Parallel programming model

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Parallel_programming_model#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Parallel_programming_model#searchInput)

In [computing](https://en.wikipedia.org/wiki/Computing), a **parallel programming model** is an [abstraction](https://en.wikipedia.org/wiki/Abstraction_(software_engineering)) of [parallel computer](https://en.wikipedia.org/wiki/Parallel_computing) architecture, with which it is convenient to express [algorithms](https://en.wikipedia.org/wiki/Algorithms) and their composition in [programs](https://en.wikipedia.org/wiki/Computer_program). The value of a programming model can be judged on its *generality*: how well a range of different problems can be expressed for a variety of different architectures, and its *performance*: how efficiently the compiled programs can execute.[[1]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-1) The implementation of a parallel programming model can take the form of a [library](https://en.wikipedia.org/wiki/Library_(computing)) invoked from a [sequential language](https://en.wikipedia.org/wiki/Sequential_programming), as an extension to an existing language, or as an entirely new language.

Consensus around a particular programming model is important because it leads to different parallel computers being built with support for the model, thereby facilitating [portability](https://en.wikipedia.org/wiki/Software_portability) of software. In this sense, programming models are referred to as [*bridging*](https://en.wikipedia.org/wiki/Bridging_model) between hardware and software.[[2]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-Valiant1990-2)



**Contents**

* [1Classification of parallel programming models](https://en.wikipedia.org/wiki/Parallel_programming_model#Classification_of_parallel_programming_models)
  + [1.1Process interaction](https://en.wikipedia.org/wiki/Parallel_programming_model#Process_interaction)
    - [1.1.1Shared memory](https://en.wikipedia.org/wiki/Parallel_programming_model#Shared_memory)
    - [1.1.2Message passing](https://en.wikipedia.org/wiki/Parallel_programming_model#Message_passing)
    - [1.1.3Implicit interaction](https://en.wikipedia.org/wiki/Parallel_programming_model#Implicit_interaction)
  + [1.2Problem decomposition](https://en.wikipedia.org/wiki/Parallel_programming_model#Problem_decomposition)
    - [1.2.1Task parallelism](https://en.wikipedia.org/wiki/Parallel_programming_model#Task_parallelism)
    - [1.2.2Data parallelism](https://en.wikipedia.org/wiki/Parallel_programming_model#Data_parallelism)
    - [1.2.3Implicit parallelism](https://en.wikipedia.org/wiki/Parallel_programming_model#Implicit_parallelism)
* [2Terminology](https://en.wikipedia.org/wiki/Parallel_programming_model#Terminology)
* [3Example parallel programming models](https://en.wikipedia.org/wiki/Parallel_programming_model#Example_parallel_programming_models)
* [4See also](https://en.wikipedia.org/wiki/Parallel_programming_model#See_also)
* [5References](https://en.wikipedia.org/wiki/Parallel_programming_model#References)
* [6Further reading](https://en.wikipedia.org/wiki/Parallel_programming_model#Further_reading)

Classification of parallel programming models[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=1)]

Classifications of parallel programming models can be divided broadly into two areas: process interaction and problem decomposition.[[3]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-3)[[4]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-4)[[5]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-5)

**Process interaction**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=2)]

Process interaction relates to the mechanisms by which parallel processes are able to communicate with each other. The most common forms of interaction are shared memory and message passing, but interaction can also be implicit (invisible to the programmer).

**Shared memory**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=3)]

*Main article:*[*Shared memory (interprocess communication)*](https://en.wikipedia.org/wiki/Shared_memory_(interprocess_communication))

Shared memory is an efficient means of passing data between processes. In a shared-memory model, parallel processes share a global address space that they read and write to asynchronously. Asynchronous concurrent access can lead to [race conditions](https://en.wikipedia.org/wiki/Race_condition), and mechanisms such as [locks](https://en.wikipedia.org/wiki/Lock_(computer_science)), [semaphores](https://en.wikipedia.org/wiki/Semaphore_(programming)) and [monitors](https://en.wikipedia.org/wiki/Monitor_(synchronization)) can be used to avoid these. Conventional [multi-core processors](https://en.wikipedia.org/wiki/Multi-core_processor) directly support shared memory, which many parallel programming languages and libraries, such as [Cilk](https://en.wikipedia.org/wiki/Cilk_(programming_language)), [OpenMP](https://en.wikipedia.org/wiki/OpenMP) and [Threading Building Blocks](https://en.wikipedia.org/wiki/Threading_Building_Blocks), are designed to exploit.

**Message passing**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=4)]

*Main article:*[*Message passing*](https://en.wikipedia.org/wiki/Message_passing)

In a message-passing model, parallel processes exchange data through passing messages to one another. These communications can be asynchronous, where a message can be sent before the receiver is ready, or synchronous, where the receiver must be ready. The [Communicating sequential processes](https://en.wikipedia.org/wiki/Communicating_sequential_processes) (CSP) formalisation of message passing uses synchronous communication channels to connect processes, and led to important languages such as [Occam](https://en.wikipedia.org/wiki/Occam_(programming_language)), [Limbo](https://en.wikipedia.org/wiki/Limbo_(programming_language)) and [Go](https://en.wikipedia.org/wiki/Go_(programming_language)). In contrast, the [actor model](https://en.wikipedia.org/wiki/Actor_model) uses asynchronous message passing and has been employed in the design of languages such as [D](https://en.wikipedia.org/wiki/D_(programming_language)), [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)) and SALSA.

**Implicit interaction**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=5)]

*Main article:*[*Implicit parallelism*](https://en.wikipedia.org/wiki/Implicit_parallelism)

In an implicit model, no process interaction is visible to the programmer and instead the compiler and/or runtime is responsible for performing it. Two examples of implicit parallelism are with [domain-specific languages](https://en.wikipedia.org/wiki/Domain-specific_language) where the concurrency within high-level operations is prescribed, and with [functional programming languages](https://en.wikipedia.org/wiki/Functional_programming) because the absence of [side-effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)) allows non-dependent functions to be executed in parallel.[[6]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-ParFuncProg-6) However, this kind of parallelism is difficult to manage[[7]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-7) and functional languages such as [Concurrent Haskell](https://en.wikipedia.org/wiki/Concurrent_Haskell) and [Concurrent ML](https://en.wikipedia.org/wiki/Concurrent_ML) provide features to manage parallelism explicitly.

**Problem decomposition**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=6)]

A parallel program is composed of simultaneously executing processes. Problem decomposition relates to the way in which the constituent processes are formulated.[[8]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-8)[[9]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-9)

**Task parallelism**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=7)]

*Main article:*[*Task parallelism*](https://en.wikipedia.org/wiki/Task_parallelism)

A task-parallel model focuses on processes, or threads of execution. These processes will often be behaviourally distinct, which emphasises the need for communication. Task parallelism is a natural way to express message-passing communication. In [Flynn's taxonomy](https://en.wikipedia.org/wiki/Flynn%27s_taxonomy), task parallelism is usually classified as [MIMD](https://en.wikipedia.org/wiki/MIMD)/[MPMD](https://en.wikipedia.org/wiki/Flynn%27s_taxonomy#MPMD) or [MISD](https://en.wikipedia.org/wiki/MISD).

**Data parallelism**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=8)]

*Main article:*[*Data parallelism*](https://en.wikipedia.org/wiki/Data_parallelism)

A data-parallel model focuses on performing operations on a data set, typically a regularly structured array. A set of tasks will operate on this data, but independently on disjoint partitions. In [Flynn's taxonomy](https://en.wikipedia.org/wiki/Flynn%27s_taxonomy), data parallelism is usually classified as [MIMD](https://en.wikipedia.org/wiki/MIMD)/[SPMD](https://en.wikipedia.org/wiki/SPMD) or [SIMD](https://en.wikipedia.org/wiki/SIMD).

**Implicit parallelism**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=9)]

*Main article:*[*Implicit parallelism*](https://en.wikipedia.org/wiki/Implicit_parallelism)

As with implicit process interaction, an implicit model of parallelism reveals nothing to the programmer as the compiler, the runtime or the hardware is responsible. For example, in compilers, [automatic parallelization](https://en.wikipedia.org/wiki/Automatic_parallelization) is the process of converting sequential code into parallel code, and in computer architecture, [superscalar execution](https://en.wikipedia.org/wiki/Superscalar_processor) is a mechanism whereby [instruction-level parallelism](https://en.wikipedia.org/wiki/Instruction-level_parallelism) is exploited to perform operations in parallel.

Terminology[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=10)]

Parallel programming models are closely related to [models of computation](https://en.wikipedia.org/wiki/Model_of_computation). A model of parallel computation is an [abstraction](https://en.wikipedia.org/wiki/Abstraction) used to analyze the cost of computational processes, but it does not necessarily need to be practical, in that it can be implemented efficiently in hardware and/or software. A programming model, in contrast, does specifically imply the practical considerations of hardware and software implementation.[[10]](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_note-10)

A parallel programming language may be based on one or a combination of programming models. For example, [High Performance Fortran](https://en.wikipedia.org/wiki/High_Performance_Fortran) is based on shared-memory interactions and data-parallel problem decomposition, and [Go](https://en.wikipedia.org/wiki/Go_(programming_language)) provides mechanism for shared-memory and message-passing interaction.

Example parallel programming models[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=11)]

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Class of interaction** | **Class of decomposition** | **Example implementations** |
| [Actor model](https://en.wikipedia.org/wiki/Actor_model) | Asynchronous message passing | Task | [D](https://en.wikipedia.org/wiki/D_(programming_language)), [Erlang](https://en.wikipedia.org/wiki/Erlang_(programming_language)), [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)), SALSA |
| [Bulk synchronous parallel](https://en.wikipedia.org/wiki/Bulk_synchronous_parallel) | Shared memory | Task | [Apache Giraph](https://en.wikipedia.org/wiki/Apache_Giraph), [Apache Hama](https://en.wikipedia.org/wiki/Apache_Hama), [BSPlib](https://en.wikipedia.org/w/index.php?title=BSPlib&action=edit&redlink=1) |
| [Communicating sequential processes](https://en.wikipedia.org/wiki/Communicating_sequential_processes) | Synchronous message passing | Task | [Ada](https://en.wikipedia.org/wiki/Ada_(programming_language)), [Occam](https://en.wikipedia.org/wiki/Occam_(programming_language)), [VerilogCSP](https://en.wikipedia.org/wiki/VerilogCSP), [Go](https://en.wikipedia.org/wiki/Go_(programming_language)) |
| [Circuits](https://en.wikipedia.org/wiki/Circuit_(computer_science)) | Message passing | Task | [Verilog](https://en.wikipedia.org/wiki/Verilog), [VHDL](https://en.wikipedia.org/wiki/VHDL) |
| [Dataflow](https://en.wikipedia.org/wiki/Dataflow_programming) | Message passing | Task | [Lustre](https://en.wikipedia.org/wiki/Lustre_(programming_language)), [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow), [Apache Flink](https://en.wikipedia.org/wiki/Apache_Flink) |
| [Functional](https://en.wikipedia.org/wiki/Functional_programming) | Message passing | Task | [Concurrent Haskell](https://en.wikipedia.org/wiki/Concurrent_Haskell), [Concurrent ML](https://en.wikipedia.org/wiki/Concurrent_ML) |
| [LogP machine](https://en.wikipedia.org/wiki/LogP_machine) | Synchronous message passing | Not specified | None |
| [Parallel random access machine](https://en.wikipedia.org/wiki/Parallel_random_access_machine) | Shared memory | Data | [Cilk](https://en.wikipedia.org/wiki/Cilk_(programming_language)), [CUDA](https://en.wikipedia.org/wiki/CUDA), [OpenMP](https://en.wikipedia.org/wiki/OpenMP), [Threading Building Blocks](https://en.wikipedia.org/wiki/Threading_Building_Blocks), [XMTC](https://en.wikipedia.org/wiki/XMTC) |

See also[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=12)]

* [Automatic parallelization](https://en.wikipedia.org/wiki/Automatic_parallelization)
* [Bridging model](https://en.wikipedia.org/wiki/Bridging_model)
* [Concurrency](https://en.wikipedia.org/wiki/Concurrent_computing)
* [Degree of parallelism](https://en.wikipedia.org/wiki/Degree_of_parallelism)
* [Explicit parallelism](https://en.wikipedia.org/wiki/Explicit_parallelism)
* [List of concurrent and parallel programming languages](https://en.wikipedia.org/wiki/List_of_concurrent_and_parallel_programming_languages)
* [Parallel external memory (Model)](https://en.wikipedia.org/wiki/Parallel_external_memory_(Model))

References[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=13)]

* 1. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-1) Skillicorn, David B., "Models for practical parallel computation", International Journal of Parallel Programming, 20.2 133–158 (1991), <https://www.ida.liu.se/~chrke55/papers/modelsurvey.pdf>
  2. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-Valiant1990_2-0) Leslie G. Valiant, "A bridging model for parallel computation", Communications of the ACM, Volume 33, Issue 8, August, 1990, pages 103–111.
  3. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-3) John E. Savage, Models of Computation: Exploring the Power of Computing, 2008, Chapter 7 (Parallel Computation), <http://cs.brown.edu/~jes/book/> [Archived](https://web.archive.org/web/20161105053330/http:/cs.brown.edu/~jes/book/) 2016-11-05 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine)
  4. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-4) Ian Foster, Designing and Building Parallel Programs, 1995, Section 1.3, "A Parallel Programming Model", <http://www.mcs.anl.gov/~itf/dbpp/text/node9.html>
  5. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-5) Blaise Barney, Introduction to Parallel Computing, "Models", 2015, Lawrence Livermore National Laboratory, <https://computing.llnl.gov/tutorials/parallel_comp/#Models>
  6. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-ParFuncProg_6-0) Hammond, Kevin. Parallel functional programming: An introduction. In International Symposium on Parallel Symbolic Computation, p. 46. 1994.
  7. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-7) McBurney, D. L., and M. Ronan Sleep. "Transputer-based experiments with the ZAPP architecture." PARLE Parallel Architectures and Languages Europe. Springer Berlin Heidelberg, 1987.
  8. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-8) Ian Foster, Designing and Building Parallel Programs, 1995, Section 2.2, "Partitioning", <http://www.mcs.anl.gov/~itf/dbpp/text/node16.html>
  9. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-9) Blaise Barney, Introduction to Parallel Computing, "Partitioning", 2015, Lawrence Livermore National Laboratory, <https://computing.llnl.gov/tutorials/parallel_comp/#DesignPartitioning>
  10. [**^**](https://en.wikipedia.org/wiki/Parallel_programming_model#cite_ref-10) Skillicorn, David B., and Domenico Talia, Models and languages for parallel computation, ACM Computing Surveys, 30.2 123–169 (1998), <https://www.cs.utexas.edu/users/browne/CS392Cf2000/papers/ModelsOfParallelComputation-Skillicorn.pdf>

Further reading[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&action=edit&section=14)]

* *Blaise Barney,*[*Introduction to Parallel Computing*](https://computing.llnl.gov/tutorials/parallel_comp/)*, Lawrence Livermore National Laboratory*
* *Murray I. Cole.,*[*Algorithmic Skeletons: Structured Management of Parallel Computation*](http://homepages.inf.ed.ac.uk/mic/Pubs/skeletonbook.pdf)*(PDF), University of Glasgow*
* *J. Darlinton; M. Ghanem; H. W. To (1993),*[*"Structured Parallel Programming"*](https://www.researchgate.net/profile/Moustafa_Ghanem/publication/3557907_Structured_parallel_programming/links/00b49523c28076a748000000.pdf)*(PDF), In Programming Models for Massively Parallel Computers. IEEE Computer Society Press*
* *Ian Foster,*[*Designing and Building Parallel Programs*](http://www.mcs.anl.gov/~itf/dbpp)*, Argonne National Laboratory*

Parallel computing

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Parallel_computing#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Parallel_computing#searchInput)

*"Parallelization" redirects here. For parallelization of manifolds, see*[*Parallelization (mathematics)*](https://en.wikipedia.org/wiki/Parallelization_(mathematics))*.*

[](https://en.wikipedia.org/wiki/File:IBM_Blue_Gene_P_supercomputer.jpg)

IBM's [Blue Gene/P](https://en.wikipedia.org/wiki/Blue_Gene) [massively parallel](https://en.wikipedia.org/wiki/Massively_parallel) [supercomputer](https://en.wikipedia.org/wiki/Supercomputer)

**Parallel computing** is a type of [computation](https://en.wikipedia.org/wiki/Computing) where many calculations or the execution of [processes](https://en.wikipedia.org/wiki/Process_(computing)) are carried out simultaneously.[[1]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-1) Large problems can often be divided into smaller ones, which can then be solved at the same time. There are several different forms of parallel computing: [bit-level](https://en.wikipedia.org/wiki/Bit-level_parallelism), [instruction-level](https://en.wikipedia.org/wiki/Instruction-level_parallelism), [data](https://en.wikipedia.org/wiki/Data_parallelism), and [task parallelism](https://en.wikipedia.org/wiki/Task_parallelism). Parallelism has long been employed in [high-performance computing](https://en.wikipedia.org/wiki/High-performance_computing), but has gained broader interest due to the physical constraints preventing [frequency scaling](https://en.wikipedia.org/wiki/Frequency_scaling).[[2]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-:0-2) As power consumption (and consequently heat generation) by computers has become a concern in recent years,[[3]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-3) parallel computing has become the dominant paradigm in [computer architecture](https://en.wikipedia.org/wiki/Computer_architecture), mainly in the form of [multi-core processors](https://en.wikipedia.org/wiki/Multi-core_processor).[[4]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-View-Power-4)

Parallel computing is closely related to [concurrent computing](https://en.wikipedia.org/wiki/Concurrent_computing)—they are frequently used together, and often conflated, though the two are distinct: it is possible to have parallelism without concurrency (such as [bit-level parallelism](https://en.wikipedia.org/wiki/Bit-level_parallelism)), and concurrency without parallelism (such as multitasking by [time-sharing](https://en.wikipedia.org/wiki/Time-sharing) on a single-core CPU).[[5]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-waza-5)[[6]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-6) In parallel computing, a computational task is typically broken down into several, often many, very similar sub-tasks that can be processed independently and whose results are combined afterwards, upon completion. In contrast, in concurrent computing, the various processes often do not address related tasks; when they do, as is typical in [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing), the separate tasks may have a varied nature and often require some [inter-process communication](https://en.wikipedia.org/wiki/Inter-process_communication) during execution.

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with [multi-core](https://en.wikipedia.org/wiki/Multi-core) and [multi-processor](https://en.wikipedia.org/wiki/Symmetric_multiprocessing) computers having multiple [processing elements](https://en.wikipedia.org/wiki/Processing_element) within a single machine, while [clusters](https://en.wikipedia.org/wiki/Computer_cluster), [MPPs](https://en.wikipedia.org/wiki/Massively_parallel_(computing)), and [grids](https://en.wikipedia.org/wiki/Grid_computing) use multiple computers to work on the same task. Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks.

In some cases parallelism is transparent to the programmer, such as in bit-level or instruction-level parallelism, but explicitly [parallel algorithms](https://en.wikipedia.org/wiki/Parallel_algorithm), particularly those that use concurrency, are more difficult to write than [sequential](https://en.wikipedia.org/wiki/Sequential_algorithm) ones,[[7]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-7) because concurrency introduces several new classes of potential [software bugs](https://en.wikipedia.org/wiki/Software_bug), of which [race conditions](https://en.wikipedia.org/wiki/Race_condition) are the most common. [Communication](https://en.wikipedia.org/wiki/Computer_networking) and [synchronization](https://en.wikipedia.org/wiki/Synchronization_(computer_science)) between the different subtasks are typically some of the greatest obstacles to getting optimal parallel program performance.

A theoretical [upper bound](https://en.wikipedia.org/wiki/Upper_bound) on the [speed-up](https://en.wikipedia.org/wiki/Speedup) of a single program as a result of parallelization is given by [Amdahl's law](https://en.wikipedia.org/wiki/Amdahl%27s_law).



**Contents**

* [1Background](https://en.wikipedia.org/wiki/Parallel_computing#Background)
  + [1.1Amdahl's law and Gustafson's law](https://en.wikipedia.org/wiki/Parallel_computing#Amdahl's_law_and_Gustafson's_law)
  + [1.2Dependencies](https://en.wikipedia.org/wiki/Parallel_computing#Dependencies)
  + [1.3Race conditions, mutual exclusion, synchronization, and parallel slowdown](https://en.wikipedia.org/wiki/Parallel_computing#Race_conditions,_mutual_exclusion,_synchronization,_and_parallel_slowdown)
  + [1.4Fine-grained, coarse-grained, and embarrassing parallelism](https://en.wikipedia.org/wiki/Parallel_computing#Fine-grained,_coarse-grained,_and_embarrassing_parallelism)
  + [1.5Consistency models](https://en.wikipedia.org/wiki/Parallel_computing#Consistency_models)
  + [1.6Flynn's taxonomy](https://en.wikipedia.org/wiki/Parallel_computing#Flynn's_taxonomy)
* [2Types of parallelism](https://en.wikipedia.org/wiki/Parallel_computing#Types_of_parallelism)
  + [2.1Bit-level parallelism](https://en.wikipedia.org/wiki/Parallel_computing#Bit-level_parallelism)
  + [2.2Instruction-level parallelism](https://en.wikipedia.org/wiki/Parallel_computing#Instruction-level_parallelism)
  + [2.3Task parallelism](https://en.wikipedia.org/wiki/Parallel_computing#Task_parallelism)
  + [2.4Superword level parallelism](https://en.wikipedia.org/wiki/Parallel_computing#Superword_level_parallelism)
* [3Hardware](https://en.wikipedia.org/wiki/Parallel_computing#Hardware)
  + [3.1Memory and communication](https://en.wikipedia.org/wiki/Parallel_computing#Memory_and_communication)
  + [3.2Classes of parallel computers](https://en.wikipedia.org/wiki/Parallel_computing#Classes_of_parallel_computers)
    - [3.2.1Multi-core computing](https://en.wikipedia.org/wiki/Parallel_computing#Multi-core_computing)
    - [3.2.2Symmetric multiprocessing](https://en.wikipedia.org/wiki/Parallel_computing#Symmetric_multiprocessing)
    - [3.2.3Distributed computing](https://en.wikipedia.org/wiki/Parallel_computing#Distributed_computing)
    - [3.2.4Specialized parallel computers](https://en.wikipedia.org/wiki/Parallel_computing#Specialized_parallel_computers)
* [4Software](https://en.wikipedia.org/wiki/Parallel_computing#Software)
  + [4.1Parallel programming languages](https://en.wikipedia.org/wiki/Parallel_computing#Parallel_programming_languages)
  + [4.2Automatic parallelization](https://en.wikipedia.org/wiki/Parallel_computing#Automatic_parallelization)
  + [4.3Application checkpointing](https://en.wikipedia.org/wiki/Parallel_computing#Application_checkpointing)
* [5Algorithmic methods](https://en.wikipedia.org/wiki/Parallel_computing#Algorithmic_methods)
* [6Fault tolerance](https://en.wikipedia.org/wiki/Parallel_computing#Fault_tolerance)
* [7History](https://en.wikipedia.org/wiki/Parallel_computing#History)
* [8Biological brain as massively parallel computer](https://en.wikipedia.org/wiki/Parallel_computing#Biological_brain_as_massively_parallel_computer)
* [9See also](https://en.wikipedia.org/wiki/Parallel_computing#See_also)
* [10References](https://en.wikipedia.org/wiki/Parallel_computing#References)
* [11Further reading](https://en.wikipedia.org/wiki/Parallel_computing#Further_reading)
* [12External links](https://en.wikipedia.org/wiki/Parallel_computing#External_links)

Background[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=1)]

Traditionally, [computer software](https://en.wikipedia.org/wiki/Computer_software) has been written for [serial computation](https://en.wikipedia.org/wiki/Serial_computation). To solve a problem, an [algorithm](https://en.wikipedia.org/wiki/Algorithm) is constructed and implemented as a serial stream of instructions. These instructions are executed on a [central processing unit](https://en.wikipedia.org/wiki/Central_processing_unit) on one computer. Only one instruction may execute at a time—after that instruction is finished, the next one is executed.[[8]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-llnltut-8)

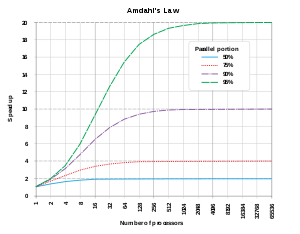
Parallel computing, on the other hand, uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.[[8]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-llnltut-8) Historically parallel computing was used for scientific computing and the simulation of scientific problems, particularly in the natural and engineering sciences, such as [meteorology](https://en.wikipedia.org/wiki/Meteorology). This led to the design of parallel hardware and software, as well as [high performance computing](https://en.wikipedia.org/wiki/High_performance_computing).[[9]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-9)

[Frequency scaling](https://en.wikipedia.org/wiki/Frequency_scaling) was the dominant reason for improvements in [computer performance](https://en.wikipedia.org/wiki/Computer_performance) from the mid-1980s until 2004. The [runtime](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)) of a program is equal to the number of instructions multiplied by the average time per instruction. Maintaining everything else constant, increasing the clock frequency decreases the average time it takes to execute an instruction. An increase in frequency thus decreases runtime for all [compute-bound](https://en.wikipedia.org/wiki/CPU_bound) programs.[[10]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-10) However, power consumption *P* by a chip is given by the equation *P* = *C* × *V* 2 × *F*, where *C* is the [capacitance](https://en.wikipedia.org/wiki/Capacitance) being switched per clock cycle (proportional to the number of transistors whose inputs change), *V* is [voltage](https://en.wikipedia.org/wiki/Voltage), and *F* is the processor frequency (cycles per second).[[11]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-11) Increases in frequency increase the amount of power used in a processor. Increasing processor power consumption led ultimately to [Intel](https://en.wikipedia.org/wiki/Intel)'s May 8, 2004 cancellation of its [Tejas and Jayhawk](https://en.wikipedia.org/wiki/Tejas_and_Jayhawk) processors, which is generally cited as the end of frequency scaling as the dominant computer architecture paradigm.[[12]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-12)

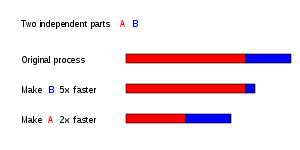
To deal with the problem of power consumption and overheating the major [central processing unit](https://en.wikipedia.org/wiki/Central_processing_unit) (CPU or processor) manufacturers started to produce power efficient processors with multiple cores. The core is the computing unit of the processor and in multi-core processors each core is independent and can access the same memory concurrently. [Multi-core processors](https://en.wikipedia.org/wiki/Multi-core_processor) have brought parallel computing to [desktop computers](https://en.wikipedia.org/wiki/Desktop_computers). Thus parallelisation of serial programmes has become a mainstream programming task. In 2012 quad-core processors became standard for [desktop computers](https://en.wikipedia.org/wiki/Desktop_computers), while [servers](https://en.wikipedia.org/wiki/Server_(computing)) have 10 and 12 core processors. From [Moore's law](https://en.wikipedia.org/wiki/Moore%27s_law) it can be predicted that the number of cores per processor will double every 18–24 months. This could mean that after 2020 a typical processor will have dozens or hundreds of cores.[[13]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-13)

An [operating system](https://en.wikipedia.org/wiki/Operating_system) can ensure that different tasks and user programmes are run in parallel on the available cores. However, for a serial software programme to take full advantage of the multi-core architecture the programmer needs to restructure and parallelise the code. A speed-up of application software runtime will no longer be achieved through frequency scaling, instead programmers will need to parallelise their software code to take advantage of the increasing computing power of multicore architectures.[[14]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-14)

**Amdahl's law and Gustafson's law**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=2)]

[](https://en.wikipedia.org/wiki/File:AmdahlsLaw.svg)

A graphical representation of [Amdahl's law](https://en.wikipedia.org/wiki/Amdahl%27s_law). The speedup of a program from parallelization is limited by how much of the program can be parallelized. For example, if 90% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 10 times no matter how many processors are used.

[](https://en.wikipedia.org/wiki/File:Optimizing-different-parts.svg)

Assume that a task has two independent parts, *A* and *B*. Part *B* takes roughly 25% of the time of the whole computation. By working very hard, one may be able to make this part 5 times faster, but this only reduces the time for the whole computation by a little. In contrast, one may need to perform less work to make part *A* be twice as fast. This will make the computation much faster than by optimizing part *B*, even though part *B'*s speedup is greater by ratio, (5 times versus 2 times).

Optimally, the [speedup](https://en.wikipedia.org/wiki/Speedup) from parallelization would be linear—doubling the number of processing elements should halve the runtime, and doubling it a second time should again halve the runtime. However, very few parallel algorithms achieve optimal speedup. Most of them have a near-linear speedup for small numbers of processing elements, which flattens out into a constant value for large numbers of processing elements.

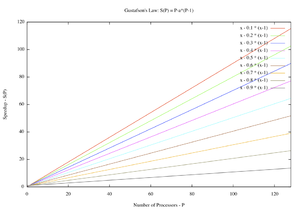
The potential speedup of an algorithm on a parallel computing platform is given by [Amdahl's law](https://en.wikipedia.org/wiki/Amdahl%27s_law)[[15]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-15)

{\displaystyle S\_{\text{latency}}(s)={\frac {1}{1-p+{\frac {p}{s}}}},}

where

* *S*latency is the potential [speedup](https://en.wikipedia.org/wiki/Speedup) in [latency](https://en.wikipedia.org/wiki/Latency_(engineering)) of the execution of the whole task;
* *s* is the speedup in latency of the execution of the parallelizable part of the task;
* *p* is the percentage of the execution time of the whole task concerning the parallelizable part of the task *before parallelization*.

Since *S*latency < 1/(1 - *p*), it shows that a small part of the program which cannot be parallelized will limit the overall speedup available from parallelization. A program solving a large mathematical or engineering problem will typically consist of several parallelizable parts and several non-parallelizable (serial) parts. If the non-parallelizable part of a program accounts for 10% of the runtime (*p* = 0.9), we can get no more than a 10 times speedup, regardless of how many processors are added. This puts an upper limit on the usefulness of adding more parallel execution units. "When a task cannot be partitioned because of sequential constraints, the application of more effort has no effect on the schedule. The bearing of a child takes nine months, no matter how many women are assigned."[[16]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-16)

[](https://en.wikipedia.org/wiki/File:Gustafson.png)

A graphical representation of [Gustafson's law](https://en.wikipedia.org/wiki/Gustafson%27s_law)

Amdahl's law only applies to cases where the problem size is fixed. In practice, as more computing resources become available, they tend to get used on larger problems (larger datasets), and the time spent in the parallelizable part often grows much faster than the inherently serial work.[[17]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-17) In this case, [Gustafson's law](https://en.wikipedia.org/wiki/Gustafson%27s_law) gives a less pessimistic and more realistic assessment of parallel performance:[[18]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-18)

{\displaystyle S\_{\text{latency}}(s)=1-p+sp.}

Both Amdahl's law and Gustafson's law assume that the running time of the serial part of the program is independent of the number of processors. Amdahl's law assumes that the entire problem is of fixed size so that the total amount of work to be done in parallel is also *independent of the number of processors*, whereas Gustafson's law assumes that the total amount of work to be done in parallel *varies linearly with the number of processors*.

**Dependencies**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=3)]

Understanding [data dependencies](https://en.wikipedia.org/wiki/Data_dependency) is fundamental in implementing [parallel algorithms](https://en.wikipedia.org/wiki/Parallel_algorithm). No program can run more quickly than the longest chain of dependent calculations (known as the [critical path](https://en.wikipedia.org/wiki/Critical_path_method)), since calculations that depend upon prior calculations in the chain must be executed in order. However, most algorithms do not consist of just a long chain of dependent calculations; there are usually opportunities to execute independent calculations in parallel.

Let *Pi* and *Pj* be two program segments. Bernstein's conditions[[19]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-19) describe when the two are independent and can be executed in parallel. For *Pi*, let *Ii* be all of the input variables and *Oi* the output variables, and likewise for *Pj*. *Pi* and *Pj* are independent if they satisfy

{\displaystyle I\_{j}\cap O\_{i}=\varnothing ,}

{\displaystyle I\_{i}\cap O\_{j}=\varnothing ,}

{\displaystyle O\_{i}\cap O\_{j}=\varnothing .}

Violation of the first condition introduces a flow dependency, corresponding to the first segment producing a result used by the second segment. The second condition represents an anti-dependency, when the second segment produces a variable needed by the first segment. The third and final condition represents an output dependency: when two segments write to the same location, the result comes from the logically last executed segment.[[20]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-20)

Consider the following functions, which demonstrate several kinds of dependencies:

1: function Dep(a, b)

2: c := a \* b

3: d := 3 \* c

4: end function

In this example, instruction 3 cannot be executed before (or even in parallel with) instruction 2, because instruction 3 uses a result from instruction 2. It violates condition 1, and thus introduces a flow dependency.

1: function NoDep(a, b)

2: c := a \* b

3: d := 3 \* b

4: e := a + b

5: end function

In this example, there are no dependencies between the instructions, so they can all be run in parallel.

Bernstein's conditions do not allow memory to be shared between different processes. For that, some means of enforcing an ordering between accesses is necessary, such as [semaphores](https://en.wikipedia.org/wiki/Semaphore_(programming)), [barriers](https://en.wikipedia.org/wiki/Barrier_(computer_science)) or some other [synchronization method](https://en.wikipedia.org/wiki/Synchronization_(computer_science)).

**Race conditions, mutual exclusion, synchronization, and parallel slowdown**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=4)]

Subtasks in a parallel program are often called [threads](https://en.wikipedia.org/wiki/Thread_(computing)). Some parallel computer architectures use smaller, lightweight versions of threads known as [fibers](https://en.wikipedia.org/wiki/Fiber_(computer_science)), while others use bigger versions known as [processes](https://en.wikipedia.org/wiki/Process_(computing)). However, "threads" is generally accepted as a generic term for subtasks.[[21]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-21) Threads will often need [synchronized](https://en.wikipedia.org/wiki/Synchronization_(computer_science)) access to an [object](https://en.wikipedia.org/wiki/Object_(computer_science)) or other [resource](https://en.wikipedia.org/wiki/Resource_management_(computing)), for example when they must update a [variable](https://en.wikipedia.org/wiki/Variable_(programming)) that is shared between them. Without synchronization, the instructions between the two threads may be interleaved in any order. For example, consider the following program:

|  |  |
| --- | --- |
| Thread A | Thread B |
| 1A: Read variable V | 1B: Read variable V |
| 2A: Add 1 to variable V | 2B: Add 1 to variable V |
| 3A: Write back to variable V | 3B: Write back to variable V |

If instruction 1B is executed between 1A and 3A, or if instruction 1A is executed between 1B and 3B, the program will produce incorrect data. This is known as a [race condition](https://en.wikipedia.org/wiki/Race_condition). The programmer must use a [lock](https://en.wikipedia.org/wiki/Lock_(computer_science)) to provide [mutual exclusion](https://en.wikipedia.org/wiki/Mutual_exclusion). A lock is a programming language construct that allows one thread to take control of a variable and prevent other threads from reading or writing it, until that variable is unlocked. The thread holding the lock is free to execute its [critical section](https://en.wikipedia.org/wiki/Critical_section) (the section of a program that requires exclusive access to some variable), and to unlock the data when it is finished. Therefore, to guarantee correct program execution, the above program can be rewritten to use locks:

|  |  |
| --- | --- |
| Thread A | Thread B |
| 1A: Lock variable V | 1B: Lock variable V |
| 2A: Read variable V | 2B: Read variable V |
| 3A: Add 1 to variable V | 3B: Add 1 to variable V |
| 4A: Write back to variable V | 4B: Write back to variable V |
| 5A: Unlock variable V | 5B: Unlock variable V |

One thread will successfully lock variable V, while the other thread will be [locked out](https://en.wikipedia.org/wiki/Software_lockout)—unable to proceed until V is unlocked again. This guarantees correct execution of the program. Locks may be necessary to ensure correct program execution when threads must serialize access to resources, but their use can greatly slow a program and may affect its [reliability](https://en.wikipedia.org/wiki/Software_quality#Reliability).[[22]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-22)

Locking multiple variables using [non-atomic](https://en.wikipedia.org/wiki/Atomic_operation) locks introduces the possibility of program [deadlock](https://en.wikipedia.org/wiki/Deadlock). An [atomic lock](https://en.wikipedia.org/wiki/Atomic_lock) locks multiple variables all at once. If it cannot lock all of them, it does not lock any of them. If two threads each need to lock the same two variables using non-atomic locks, it is possible that one thread will lock one of them and the second thread will lock the second variable. In such a case, neither thread can complete, and deadlock results.[[23]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-23)

Many parallel programs require that their subtasks [act in synchrony](https://en.wikipedia.org/wiki/Synchronization_(computer_science)). This requires the use of a [barrier](https://en.wikipedia.org/wiki/Barrier_(computer_science)). Barriers are typically implemented using a lock or a [semaphore](https://en.wikipedia.org/wiki/Semaphore_(programming)).[[24]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-24) One class of algorithms, known as [lock-free and wait-free algorithms](https://en.wikipedia.org/wiki/Lock-free_and_wait-free_algorithms), altogether avoids the use of locks and barriers. However, this approach is generally difficult to implement and requires correctly designed data structures.[[25]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-25)

Not all parallelization results in speed-up. Generally, as a task is split up into more and more threads, those threads spend an ever-increasing portion of their time communicating with each other or waiting on each other for access to resources.[[26]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-26)[[27]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-27) Once the overhead from resource contention or communication dominates the time spent on other computation, further parallelization (that is, splitting the workload over even more threads) increases rather than decreases the amount of time required to finish. This problem, known as [parallel slowdown](https://en.wikipedia.org/wiki/Parallel_slowdown),[[28]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-28) can be improved in some cases by software analysis and redesign.[[29]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-29)

**Fine-grained, coarse-grained, and embarrassing parallelism**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=5)]

Applications are often classified according to how often their subtasks need to synchronize or communicate with each other. An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; it exhibits coarse-grained parallelism if they do not communicate many times per second, and it exhibits [embarrassing parallelism](https://en.wikipedia.org/wiki/Embarrassingly_parallel) if they rarely or never have to communicate. Embarrassingly parallel applications are considered the easiest to parallelize.

**Consistency models**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=6)]

*Main article:*[*Consistency model*](https://en.wikipedia.org/wiki/Consistency_model)

Parallel programming languages and parallel computers must have a [consistency model](https://en.wikipedia.org/wiki/Consistency_model) (also known as a memory model). The consistency model defines rules for how operations on [computer memory](https://en.wikipedia.org/wiki/Computer_data_storage) occur and how results are produced.

One of the first consistency models was [Leslie Lamport](https://en.wikipedia.org/wiki/Leslie_Lamport)'s [sequential consistency](https://en.wikipedia.org/wiki/Sequential_consistency) model. Sequential consistency is the property of a parallel program that its parallel execution produces the same results as a sequential program. Specifically, a program is sequentially consistent if "the results of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program".[[30]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-30)

[Software transactional memory](https://en.wikipedia.org/wiki/Software_transactional_memory) is a common type of consistency model. Software transactional memory borrows from [database theory](https://en.wikipedia.org/wiki/Database_management_system) the concept of [atomic transactions](https://en.wikipedia.org/wiki/Atomic_commit) and applies them to memory accesses.

Mathematically, these models can be represented in several ways. Introduced in 1962, [Petri nets](https://en.wikipedia.org/wiki/Petri_net) were an early attempt to codify the rules of consistency models. Dataflow theory later built upon these, and [Dataflow architectures](https://en.wikipedia.org/wiki/Dataflow_architecture) were created to physically implement the ideas of dataflow theory. Beginning in the late 1970s, [process calculi](https://en.wikipedia.org/wiki/Process_calculi) such as [Calculus of Communicating Systems](https://en.wikipedia.org/wiki/Calculus_of_Communicating_Systems) and [Communicating Sequential Processes](https://en.wikipedia.org/wiki/Communicating_Sequential_Processes) were developed to permit algebraic reasoning about systems composed of interacting components. More recent additions to the process calculus family, such as the [π-calculus](https://en.wikipedia.org/wiki/Pi_calculus), have added the capability for reasoning about dynamic topologies. Logics such as Lamport's [TLA+](https://en.wikipedia.org/wiki/Temporal_logic_of_actions), and mathematical models such as [traces](https://en.wikipedia.org/wiki/Trace_theory) and [Actor event diagrams](https://en.wikipedia.org/wiki/Actor_model_theory), have also been developed to describe the behavior of concurrent systems.

*See also:*[*Relaxed sequential*](https://en.wikipedia.org/wiki/Relaxed_sequential)

**Flynn's taxonomy**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=7)]

[Michael J. Flynn](https://en.wikipedia.org/wiki/Michael_J._Flynn) created one of the earliest classification systems for parallel (and sequential) computers and programs, now known as [Flynn's taxonomy](https://en.wikipedia.org/wiki/Flynn%27s_taxonomy). Flynn classified programs and computers by whether they were operating using a single set or multiple sets of instructions, and whether or not those instructions were using a single set or multiple sets of data.

|  |
| --- |
| [**Flynn's taxonomy**](https://en.wikipedia.org/wiki/Flynn%27s_taxonomy) |
| **Single data stream** |
| * [SISD](https://en.wikipedia.org/wiki/SISD) * [MISD](https://en.wikipedia.org/wiki/MISD) |
| **Multiple data streams** |
| * [SIMD](https://en.wikipedia.org/wiki/SIMD) * [MIMD](https://en.wikipedia.org/wiki/MIMD) * [SPMD](https://en.wikipedia.org/wiki/SPMD) * [MPMD](https://en.wikipedia.org/wiki/MPMD) |

The single-instruction-single-data (SISD) classification is equivalent to an entirely sequential program. The single-instruction-multiple-data (SIMD) classification is analogous to doing the same operation repeatedly over a large data set. This is commonly done in [signal processing](https://en.wikipedia.org/wiki/Signal_processing) applications. Multiple-instruction-single-data (MISD) is a rarely used classification. While computer architectures to deal with this were devised (such as [systolic arrays](https://en.wikipedia.org/wiki/Systolic_array)), few applications that fit this class materialized. Multiple-instruction-multiple-data (MIMD) programs are by far the most common type of parallel programs.

According to [David A. Patterson](https://en.wikipedia.org/wiki/David_A._Patterson_(scientist)) and [John L. Hennessy](https://en.wikipedia.org/wiki/John_L._Hennessy), "Some machines are hybrids of these categories, of course, but this classic model has survived because it is simple, easy to understand, and gives a good first approximation. It is also—perhaps because of its understandability—the most widely used scheme."[[31]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-31)

Types of parallelism[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=8)]

**Bit-level parallelism**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=9)]

*Main article:*[*Bit-level parallelism*](https://en.wikipedia.org/wiki/Bit-level_parallelism)

From the advent of [very-large-scale integration](https://en.wikipedia.org/wiki/Very-large-scale_integration) (VLSI) computer-chip fabrication technology in the 1970s until about 1986, speed-up in computer architecture was driven by doubling [computer word size](https://en.wikipedia.org/wiki/Word_(data_type))—the amount of information the processor can manipulate per cycle.[[32]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-32) Increasing the word size reduces the number of instructions the processor must execute to perform an operation on variables whose sizes are greater than the length of the word. For example, where an [8-bit](https://en.wikipedia.org/wiki/8-bit) processor must add two [16-bit](https://en.wikipedia.org/wiki/16-bit) [integers](https://en.wikipedia.org/wiki/Integer), the processor must first add the 8 lower-order bits from each integer using the standard addition instruction, then add the 8 higher-order bits using an add-with-carry instruction and the [carry bit](https://en.wikipedia.org/wiki/Carry_bit) from the lower order addition; thus, an 8-bit processor requires two instructions to complete a single operation, where a 16-bit processor would be able to complete the operation with a single instruction.

Historically, [4-bit](https://en.wikipedia.org/wiki/4-bit) microprocessors were replaced with 8-bit, then 16-bit, then 32-bit microprocessors. This trend generally came to an end with the introduction of 32-bit processors, which has been a standard in general-purpose computing for two decades. Not until the early 2000s, with the advent of [x86-64](https://en.wikipedia.org/wiki/X86-64) architectures, did [64-bit](https://en.wikipedia.org/wiki/64-bit) processors become commonplace.

**Instruction-level parallelism**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=10)]

*Main article:*[*Instruction-level parallelism*](https://en.wikipedia.org/wiki/Instruction-level_parallelism)

[](https://en.wikipedia.org/wiki/File:Nopipeline.png)

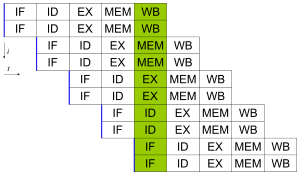
A canonical processor without [pipeline](https://en.wikipedia.org/wiki/Instruction_pipelining). It takes five clock cycles to complete one instruction and thus the processor can issue subscalar performance (IPC = 0.2 < 1).

[](https://en.wikipedia.org/wiki/File:Fivestagespipeline.png)

A canonical five-stage [pipelined](https://en.wikipedia.org/wiki/Instruction_pipelining) processor. In the best case scenario, it takes one clock cycle to complete one instruction and thus the processor can issue scalar performance (IPC = 1).

A computer program is, in essence, a stream of instructions executed by a processor. Without instruction-level parallelism, a processor can only issue less than one [instruction per clock cycle](https://en.wikipedia.org/wiki/Instructions_per_cycle) (IPC < 1). These processors are known as *subscalar* processors. These instructions can be [re-ordered](https://en.wikipedia.org/wiki/Out-of-order_execution) and combined into groups which are then executed in parallel without changing the result of the program. This is known as instruction-level parallelism. Advances in instruction-level parallelism dominated computer architecture from the mid-1980s until the mid-1990s.[[33]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-33)

All modern processors have multi-stage [instruction pipelines](https://en.wikipedia.org/wiki/Instruction_pipelining). Each stage in the pipeline corresponds to a different action the processor performs on that instruction in that stage; a processor with an *N*-stage pipeline can have up to *N* different instructions at different stages of completion and thus can issue one instruction per clock cycle (IPC = 1). These processors are known as *scalar* processors. The canonical example of a pipelined processor is a [RISC](https://en.wikipedia.org/wiki/Reduced_Instruction_Set_Computer) processor, with five stages: instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM), and register write back (WB). The [Pentium 4](https://en.wikipedia.org/wiki/Pentium_4) processor had a 35-stage pipeline.[[34]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-34)

[](https://en.wikipedia.org/wiki/File:Superscalarpipeline.svg)

A canonical five-stage [pipelined](https://en.wikipedia.org/wiki/Instruction_pipelining) [superscalar](https://en.wikipedia.org/wiki/Superscalar) processor. In the best case scenario, it takes one clock cycle to complete two instructions and thus the processor can issue superscalar performance (IPC = 2 > 1).

Most modern processors also have multiple [execution units](https://en.wikipedia.org/wiki/Execution_unit). They usually combine this feature with pipelining and thus can issue more than one instruction per clock cycle (IPC > 1). These processors are known as [*superscalar*](https://en.wikipedia.org/wiki/Superscalar) processors. Instructions can be grouped together only if there is no [data dependency](https://en.wikipedia.org/wiki/Data_dependency) between them. [Scoreboarding](https://en.wikipedia.org/wiki/Scoreboarding) and the [Tomasulo algorithm](https://en.wikipedia.org/wiki/Tomasulo_algorithm) (which is similar to scoreboarding but makes use of [register renaming](https://en.wikipedia.org/wiki/Register_renaming)) are two of the most common techniques for implementing out-of-order execution and instruction-level parallelism.

**Task parallelism**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=11)]

*Main article:*[*Task parallelism*](https://en.wikipedia.org/wiki/Task_parallelism)

Task parallelisms is the characteristic of a parallel program that "entirely different calculations can be performed on either the same or different sets of data".[[35]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-Culler124-35) This contrasts with data parallelism, where the same calculation is performed on the same or different sets of data. Task parallelism involves the decomposition of a task into sub-tasks and then allocating each sub-task to a processor for execution. The processors would then execute these sub-tasks concurrently and often cooperatively. Task parallelism does not usually scale with the size of a problem.[[36]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-Culler125-36)

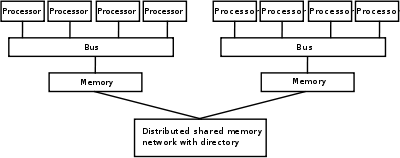
**Superword level parallelism**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=12)]

Superword level parallelism is a [vectorization](https://en.wikipedia.org/wiki/Automatic_vectorization) technique based on [loop unrolling](https://en.wikipedia.org/wiki/Loop_unwinding) and basic block vectorization. It is distinct from loop vectorization algorithms in that it can exploit [parallelism](https://en.wikipedia.org/wiki/Parallelism_(computing)) of [inline code](https://en.wikipedia.org/wiki/Inline_code), such as manipulating coordinates, color channels or in loops unrolled by hand.[[37]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-37)

Hardware[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=13)]

**Memory and communication**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=14)]

Main memory in a parallel computer is either [shared memory](https://en.wikipedia.org/wiki/Shared_memory_(interprocess_communication)) (shared between all processing elements in a single [address space](https://en.wikipedia.org/wiki/Address_space)), or [distributed memory](https://en.wikipedia.org/wiki/Distributed_memory) (in which each processing element has its own local address space).[[38]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-PH713-38) Distributed memory refers to the fact that the memory is logically distributed, but often implies that it is physically distributed as well. [Distributed shared memory](https://en.wikipedia.org/wiki/Distributed_shared_memory) and [memory virtualization](https://en.wikipedia.org/wiki/Memory_virtualization) combine the two approaches, where the processing element has its own local memory and access to the memory on non-local processors. Accesses to local memory are typically faster than accesses to non-local memory. On the [supercomputers](https://en.wikipedia.org/wiki/Supercomputers), distributed shared memory space can be implemented using the programming model such as [PGAS](https://en.wikipedia.org/wiki/Partitioned_global_address_space). This model allows processes on one compute node to transparently access the remote memory of another compute node. All compute nodes are also connected to an external shared memory system via high-speed interconnect, such as [Infiniband](https://en.wikipedia.org/wiki/Infiniband), this external shared memory system is known as [burst buffer](https://en.wikipedia.org/wiki/Burst_buffer), which is typically built from arrays of [non-volatile memory](https://en.wikipedia.org/wiki/Non-volatile_memory) physically distributed across multiple I/O nodes.

[](https://en.wikipedia.org/wiki/File:Numa.svg)

A logical view of a [non-uniform memory access](https://en.wikipedia.org/wiki/Non-uniform_memory_access) (NUMA) architecture. Processors in one directory can access that directory's memory with less latency than they can access memory in the other directory's memory.

Computer architectures in which each element of main memory can be accessed with equal [latency](https://en.wikipedia.org/wiki/Memory_latency) and [bandwidth](https://en.wikipedia.org/wiki/Bandwidth_(computing)) are known as [uniform memory access](https://en.wikipedia.org/wiki/Uniform_memory_access) (UMA) systems. Typically, that can be achieved only by a [shared memory](https://en.wikipedia.org/wiki/Shared_memory_(interprocess_communication)) system, in which the memory is not physically distributed. A system that does not have this property is known as a [non-uniform memory access](https://en.wikipedia.org/wiki/Non-uniform_memory_access) (NUMA) architecture. Distributed memory systems have non-uniform memory access.

Computer systems make use of [caches](https://en.wikipedia.org/wiki/CPU_cache)—small and fast memories located close to the processor which store temporary copies of memory values (nearby in both the physical and logical sense). Parallel computer systems have difficulties with caches that may store the same value in more than one location, with the possibility of incorrect program execution. These computers require a [cache coherency](https://en.wikipedia.org/wiki/Cache_coherency) system, which keeps track of cached values and strategically purges them, thus ensuring correct program execution. [Bus snooping](https://en.wikipedia.org/wiki/Bus_sniffing) is one of the most common methods for keeping track of which values are being accessed (and thus should be purged). Designing large, high-performance cache coherence systems is a very difficult problem in computer architecture. As a result, shared memory computer architectures do not scale as well as distributed memory systems do.[[38]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-PH713-38)

Processor–processor and processor–memory communication can be implemented in hardware in several ways, including via shared (either multiported or [multiplexed](https://en.wikipedia.org/wiki/Multiplexing)) memory, a [crossbar switch](https://en.wikipedia.org/wiki/Crossbar_switch), a shared [bus](https://en.wikipedia.org/wiki/Bus_(computing)) or an interconnect network of a myriad of [topologies](https://en.wikipedia.org/wiki/Network_topology) including [star](https://en.wikipedia.org/wiki/Star_network), [ring](https://en.wikipedia.org/wiki/Ring_network), [tree](https://en.wikipedia.org/wiki/Tree_(graph_theory)), [hypercube](https://en.wikipedia.org/wiki/Hypercube_graph), fat hypercube (a hypercube with more than one processor at a node), or [n-dimensional mesh](https://en.wikipedia.org/wiki/Mesh_networking).

Parallel computers based on interconnected networks need to have some kind of [routing](https://en.wikipedia.org/wiki/Routing) to enable the passing of messages between nodes that are not directly connected. The medium used for communication between the processors is likely to be hierarchical in large multiprocessor machines.

**Classes of parallel computers**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=15)]

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism. This classification is broadly analogous to the distance between basic computing nodes. These are not mutually exclusive; for example, clusters of symmetric multiprocessors are relatively common.

**Multi-core computing**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=16)]

*Main article:*[*Multi-core processor*](https://en.wikipedia.org/wiki/Multi-core_processor)

A multi-core processor is a processor that includes multiple [processing units](https://en.wikipedia.org/wiki/Central_processing_unit) (called "cores") on the same chip. This processor differs from a [superscalar](https://en.wikipedia.org/wiki/Superscalar) processor, which includes multiple [execution units](https://en.wikipedia.org/wiki/Execution_unit) and can issue multiple instructions per clock cycle from one instruction stream (thread); in contrast, a multi-core processor can issue multiple instructions per clock cycle from multiple instruction streams. [IBM](https://en.wikipedia.org/wiki/IBM)'s [Cell microprocessor](https://en.wikipedia.org/wiki/Cell_(microprocessor)), designed for use in the [Sony](https://en.wikipedia.org/wiki/Sony) [PlayStation 3](https://en.wikipedia.org/wiki/PlayStation_3), is a prominent multi-core processor. Each core in a multi-core processor can potentially be superscalar as well—that is, on every clock cycle, each core can issue multiple instructions from one thread.

[Simultaneous multithreading](https://en.wikipedia.org/wiki/Simultaneous_multithreading) (of which Intel's [Hyper-Threading](https://en.wikipedia.org/wiki/Hyper-Threading) is the best known) was an early form of pseudo-multi-coreism. A processor capable of concurrent multithreading includes multiple execution units in the same processing unit—that is it has a superscalar architecture—and can issue multiple instructions per clock cycle from *multiple* threads. [Temporal multithreading](https://en.wikipedia.org/wiki/Temporal_multithreading) on the other hand includes a single execution unit in the same processing unit and can issue one instruction at a time from *multiple* threads.

**Symmetric multiprocessing**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=17)]

*Main article:*[*Symmetric multiprocessing*](https://en.wikipedia.org/wiki/Symmetric_multiprocessing)

A symmetric multiprocessor (SMP) is a computer system with multiple identical processors that share memory and connect via a [bus](https://en.wikipedia.org/wiki/Bus_(computing)).[[39]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-HP549-39) [Bus contention](https://en.wikipedia.org/wiki/Bus_contention) prevents bus architectures from scaling. As a result, SMPs generally do not comprise more than 32 processors.[[40]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-40) Because of the small size of the processors and the significant reduction in the requirements for bus bandwidth achieved by large caches, such symmetric multiprocessors are extremely cost-effective, provided that a sufficient amount of memory bandwidth exists.[[39]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-HP549-39)

**Distributed computing**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=18)]

*Main article:*[*Distributed computing*](https://en.wikipedia.org/wiki/Distributed_computing)

A distributed computer (also known as a distributed memory multiprocessor) is a distributed memory computer system in which the processing elements are connected by a network. Distributed computers are highly scalable. The terms "[concurrent computing](https://en.wikipedia.org/wiki/Concurrent_computing)", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them.[[41]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-41) The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel.[[42]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-42)

**Cluster computing**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=19)]

*Main article:*[*Computer cluster*](https://en.wikipedia.org/wiki/Computer_cluster)

[](https://en.wikipedia.org/wiki/File:Beowulf.jpg)

A [Beowulf cluster](https://en.wikipedia.org/wiki/Beowulf_(computing))

A cluster is a group of loosely coupled computers that work together closely, so that in some respects they can be regarded as a single computer.[[43]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-43) Clusters are composed of multiple standalone machines connected by a network. While machines in a cluster do not have to be symmetric, [load balancing](https://en.wikipedia.org/wiki/Load_balancing_(computing)) is more difficult if they are not. The most common type of cluster is the [Beowulf cluster](https://en.wikipedia.org/wiki/Beowulf_(computing)), which is a cluster implemented on multiple identical [commercial off-the-shelf](https://en.wikipedia.org/wiki/Commercial_off-the-shelf) computers connected with a [TCP/IP](https://en.wikipedia.org/wiki/TCP/IP) [Ethernet](https://en.wikipedia.org/wiki/Ethernet) [local area network](https://en.wikipedia.org/wiki/Local_area_network).[[44]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-44) Beowulf technology was originally developed by [Thomas Sterling](https://en.wikipedia.org/wiki/Thomas_Sterling_(computing)) and [Donald Becker](https://en.wikipedia.org/wiki/Donald_Becker). 87% of all [Top500](https://en.wikipedia.org/wiki/TOP500) supercomputers are clusters.[[45]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-45) The remaining are Massively Parallel Processors, explained below.

Because grid computing systems (described below) can easily handle embarrassingly parallel problems, modern clusters are typically designed to handle more difficult problems—problems that require nodes to share intermediate results with each other more often. This requires a high bandwidth and, more importantly, a low-[latency](https://en.wikipedia.org/wiki/Latency_(engineering)) interconnection network. Many historic and current supercomputers use customized high-performance network hardware specifically designed for cluster computing, such as the Cray Gemini network.[[46]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-46) As of 2014, most current supercomputers use some off-the-shelf standard network hardware, often [Myrinet](https://en.wikipedia.org/wiki/Myrinet), [InfiniBand](https://en.wikipedia.org/wiki/InfiniBand), or [Gigabit Ethernet](https://en.wikipedia.org/wiki/Gigabit_Ethernet).

**Massively parallel computing**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=20)]

*Main article:*[*Massively parallel (computing)*](https://en.wikipedia.org/wiki/Massively_parallel_(computing))

[](https://en.wikipedia.org/wiki/File:BlueGeneL_cabinet.jpg)

A cabinet from [IBM](https://en.wikipedia.org/wiki/IBM)'s [Blue Gene/L](https://en.wikipedia.org/wiki/Blue_Gene) massively parallel [supercomputer](https://en.wikipedia.org/wiki/Supercomputer)

A massively parallel processor (MPP) is a single computer with many networked processors. MPPs have many of the same characteristics as clusters, but MPPs have specialized interconnect networks (whereas clusters use commodity hardware for networking). MPPs also tend to be larger than clusters, typically having "far more" than 100 processors.[[47]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-47) In an MPP, "each CPU contains its own memory and copy of the operating system and application. Each subsystem communicates with the others via a high-speed interconnect."[[48]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-48)

[IBM](https://en.wikipedia.org/wiki/IBM)'s [Blue Gene/L](https://en.wikipedia.org/wiki/Blue_Gene), the fifth fastest [supercomputer](https://en.wikipedia.org/wiki/Supercomputer) in the world according to the June 2009 [TOP500](https://en.wikipedia.org/wiki/TOP500) ranking, is an MPP.

**Grid computing**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=21)]

*Main article:*[*Grid computing*](https://en.wikipedia.org/wiki/Grid_computing)

Grid computing is the most distributed form of parallel computing. It makes use of computers communicating over the [Internet](https://en.wikipedia.org/wiki/Internet) to work on a given problem. Because of the low bandwidth and extremely high latency available on the Internet, distributed computing typically deals only with [embarrassingly parallel](https://en.wikipedia.org/wiki/Embarrassingly_parallel) problems. [Many distributed computing applications](https://en.wikipedia.org/wiki/List_of_distributed_computing_projects) have been created, of which [SETI@home](https://en.wikipedia.org/wiki/SETI@home) and [Folding@home](https://en.wikipedia.org/wiki/Folding@home) are the best-known examples.[[49]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-49)

Most grid computing applications use [middleware](https://en.wikipedia.org/wiki/Middleware) (software that sits between the operating system and the application to manage network resources and standardize the software interface). The most common distributed computing middleware is the [Berkeley Open Infrastructure for Network Computing](https://en.wikipedia.org/wiki/Berkeley_Open_Infrastructure_for_Network_Computing) (BOINC). Often, distributed computing software makes use of "spare cycles", performing computations at times when a computer is idling.

**Specialized parallel computers**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=22)]

Within parallel computing, there are specialized parallel devices that remain niche areas of interest. While not [domain-specific](https://en.wikipedia.org/wiki/Domain-specific_programming_language), they tend to be applicable to only a few classes of parallel problems.

**Reconfigurable computing with field-programmable gate arrays**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=23)]

[Reconfigurable computing](https://en.wikipedia.org/wiki/Reconfigurable_computing) is the use of a [field-programmable gate array](https://en.wikipedia.org/wiki/Field-programmable_gate_array) (FPGA) as a co-processor to a general-purpose computer. An FPGA is, in essence, a computer chip that can rewire itself for a given task.

FPGAs can be programmed with [hardware description languages](https://en.wikipedia.org/wiki/Hardware_description_language) such as [VHDL](https://en.wikipedia.org/wiki/VHDL) or [Verilog](https://en.wikipedia.org/wiki/Verilog). However, programming in these languages can be tedious. Several vendors have created [C to HDL](https://en.wikipedia.org/wiki/C_to_HDL) languages that attempt to emulate the syntax and semantics of the [C programming language](https://en.wikipedia.org/wiki/C_programming_language), with which most programmers are familiar. The best known C to HDL languages are [Mitrion-C](https://en.wikipedia.org/wiki/Mitrionics), [Impulse C](https://en.wikipedia.org/wiki/Impulse_C), [DIME-C](https://en.wikipedia.org/w/index.php?title=DIME-C&action=edit&redlink=1), and [Handel-C](https://en.wikipedia.org/wiki/Handel-C). Specific subsets of [SystemC](https://en.wikipedia.org/wiki/SystemC) based on C++ can also be used for this purpose.

AMD's decision to open its [HyperTransport](https://en.wikipedia.org/wiki/HyperTransport) technology to third-party vendors has become the enabling technology for high-performance reconfigurable computing.[[50]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-DAmour-50) According to Michael R. D'Amour, Chief Operating Officer of [DRC Computer Corporation](https://en.wikipedia.org/w/index.php?title=DRC_Computer_Corporation&action=edit&redlink=1), "when we first walked into AMD, they called us 'the [socket](https://en.wikipedia.org/wiki/CPU_socket) stealers.' Now they call us their partners."[[50]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-DAmour-50)

**General-purpose computing on graphics processing units (GPGPU)**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=24)]

*Main article:*[*GPGPU*](https://en.wikipedia.org/wiki/GPGPU)

[](https://en.wikipedia.org/wiki/File:NvidiaTesla.jpg)

Nvidia's [Tesla GPGPU card](https://en.wikipedia.org/wiki/Nvidia_Tesla)

General-purpose computing on [graphics processing units](https://en.wikipedia.org/wiki/Graphics_processing_unit) (GPGPU) is a fairly recent trend in computer engineering research. GPUs are co-processors that have been heavily optimized for [computer graphics](https://en.wikipedia.org/wiki/Computer_graphics) processing.[[51]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-51) Computer graphics processing is a field dominated by data parallel operations—particularly [linear algebra](https://en.wikipedia.org/wiki/Linear_algebra) [matrix](https://en.wikipedia.org/wiki/Matrix_(mathematics)) operations.

In the early days, GPGPU programs used the normal graphics APIs for executing programs. However, several new programming languages and platforms have been built to do general purpose computation on GPUs with both [Nvidia](https://en.wikipedia.org/wiki/Nvidia) and [AMD](https://en.wikipedia.org/wiki/AMD) releasing programming environments with [CUDA](https://en.wikipedia.org/wiki/CUDA) and [Stream SDK](https://en.wikipedia.org/wiki/AMD_FireStream#Software_Development_Kit) respectively. Other GPU programming languages include [BrookGPU](https://en.wikipedia.org/wiki/BrookGPU), [PeakStream](https://en.wikipedia.org/wiki/PeakStream), and [RapidMind](https://en.wikipedia.org/wiki/RapidMind). Nvidia has also released specific products for computation in their [Tesla series](https://en.wikipedia.org/wiki/Nvidia_Tesla). The technology consortium Khronos Group has released the [OpenCL](https://en.wikipedia.org/wiki/OpenCL) specification, which is a framework for writing programs that execute across platforms consisting of CPUs and GPUs. [AMD](https://en.wikipedia.org/wiki/AMD), [Apple](https://en.wikipedia.org/wiki/Apple_Inc.), [Intel](https://en.wikipedia.org/wiki/Intel), [Nvidia](https://en.wikipedia.org/wiki/Nvidia) and others are supporting [OpenCL](https://en.wikipedia.org/wiki/OpenCL).

**Application-specific integrated circuits**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=25)]

*Main article:*[*Application-specific integrated circuit*](https://en.wikipedia.org/wiki/Application-specific_integrated_circuit)

Several [application-specific integrated circuit](https://en.wikipedia.org/wiki/Application-specific_integrated_circuit) (ASIC) approaches have been devised for dealing with parallel applications.[[52]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-52)[[53]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-53)[[54]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-54)

Because an ASIC is (by definition) specific to a given application, it can be fully optimized for that application. As a result, for a given application, an ASIC tends to outperform a general-purpose computer. However, ASICs are created by [UV photolithography](https://en.wikipedia.org/wiki/Photolithography). This process requires a mask set, which can be extremely expensive. A mask set can cost over a million US dollars.[[55]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-55) (The smaller the transistors required for the chip, the more expensive the mask will be.) Meanwhile, performance increases in general-purpose computing over time (as described by [Moore's law](https://en.wikipedia.org/wiki/Moore%27s_law)) tend to wipe out these gains in only one or two chip generations.[[50]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-DAmour-50) High initial cost, and the tendency to be overtaken by Moore's-law-driven general-purpose computing, has rendered ASICs unfeasible for most parallel computing applications. However, some have been built. One example is the PFLOPS [RIKEN MDGRAPE-3](https://en.wikipedia.org/wiki/RIKEN_MDGRAPE-3) machine which uses custom ASICs for [molecular dynamics](https://en.wikipedia.org/wiki/Molecular_dynamics) simulation.

**Vector processors**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=26)]

*Main article:*[*Vector processor*](https://en.wikipedia.org/wiki/Vector_processor)

[](https://en.wikipedia.org/wiki/File:Cray_1_IMG_9126.jpg)

The [Cray-1](https://en.wikipedia.org/wiki/Cray-1) is a vector processor

A vector processor is a CPU or computer system that can execute the same instruction on large sets of data. Vector processors have high-level operations that work on linear arrays of numbers or vectors. An example vector operation is *A* = *B* × *C*, where *A*, *B*, and *C* are each 64-element vectors of 64-bit [floating-point](https://en.wikipedia.org/wiki/Floating-point) numbers.[[56]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-PH751-56) They are closely related to Flynn's SIMD classification.[[56]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-PH751-56)

[Cray](https://en.wikipedia.org/wiki/Cray) computers became famous for their vector-processing computers in the 1970s and 1980s. However, vector processors—both as CPUs and as full computer systems—have generally disappeared. Modern [processor instruction sets](https://en.wikipedia.org/wiki/Instruction_set) do include some vector processing instructions, such as with [Freescale Semiconductor](https://en.wikipedia.org/wiki/Freescale_Semiconductor)'s [AltiVec](https://en.wikipedia.org/wiki/AltiVec) and [Intel](https://en.wikipedia.org/wiki/Intel)'s [Streaming SIMD Extensions](https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions) (SSE).

Software[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=27)]

**Parallel programming languages**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=28)]

*Main article:*[*List of concurrent and parallel programming languages*](https://en.wikipedia.org/wiki/List_of_concurrent_and_parallel_programming_languages)

[Concurrent programming languages](https://en.wikipedia.org/wiki/List_of_concurrent_and_parallel_programming_languages), [libraries](https://en.wikipedia.org/wiki/Library_(computing)), [APIs](https://en.wikipedia.org/wiki/Application_programming_interface), and [parallel programming models](https://en.wikipedia.org/wiki/Parallel_programming_model) (such as [algorithmic skeletons](https://en.wikipedia.org/wiki/Algorithmic_skeleton)) have been created for programming parallel computers. These can generally be divided into classes based on the assumptions they make about the underlying memory architecture—shared memory, distributed memory, or shared distributed memory. Shared memory programming languages communicate by manipulating shared memory variables. Distributed memory uses [message passing](https://en.wikipedia.org/wiki/Message_passing). [POSIX Threads](https://en.wikipedia.org/wiki/POSIX_Threads) and [OpenMP](https://en.wikipedia.org/wiki/OpenMP) are two of the most widely used shared memory APIs, whereas [Message Passing Interface](https://en.wikipedia.org/wiki/Message_Passing_Interface) (MPI) is the most widely used message-passing system API.[[57]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-57) One concept used in programming parallel programs is the [future concept](https://en.wikipedia.org/wiki/Futures_and_promises), where one part of a program promises to deliver a required datum to another part of a program at some future time.

[CAPS entreprise](https://en.wikipedia.org/w/index.php?title=CAPS_entreprise&action=edit&redlink=1) and [Pathscale](https://en.wikipedia.org/w/index.php?title=Pathscale&action=edit&redlink=1) are also coordinating their effort to make [hybrid multi-core parallel programming](https://en.wikipedia.org/w/index.php?title=Hybrid_multi-core_parallel_programming&action=edit&redlink=1) (HMPP) directives an open standard called [OpenHMPP](https://en.wikipedia.org/wiki/OpenHMPP). The OpenHMPP directive-based programming model offers a syntax to efficiently offload computations on hardware accelerators and to optimize data movement to/from the hardware memory. OpenHMPP directives describe [remote procedure call](https://en.wikipedia.org/wiki/Remote_procedure_call) (RPC) on an accelerator device (e.g. GPU) or more generally a set of cores. The directives annotate [C](https://en.wikipedia.org/wiki/C_(programming_language)) or [Fortran](https://en.wikipedia.org/wiki/Fortran) codes to describe two sets of functionalities: the offloading of procedures (denoted codelets) onto a remote device and the optimization of data transfers between the CPU main memory and the accelerator memory.

The rise of consumer GPUs has led to support for [compute kernels](https://en.wikipedia.org/wiki/Compute_kernel), either in graphics APIs (referred to as [compute shaders](https://en.wikipedia.org/wiki/Compute_shader)), in dedicated APIs (such as [OpenCL](https://en.wikipedia.org/wiki/OpenCL)), or in other language extensions.

**Automatic parallelization**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=29)]

*Main article:*[*Automatic parallelization*](https://en.wikipedia.org/wiki/Automatic_parallelization)

Automatic parallelization of a sequential program by a [compiler](https://en.wikipedia.org/wiki/Compiler) is the "holy grail" of parallel computing, especially with the aforementioned limit of processor frequency. Despite decades of work by compiler researchers, automatic parallelization has had only limited success.[[58]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-58)

Mainstream parallel programming languages remain either [explicitly parallel](https://en.wikipedia.org/wiki/Explicit_parallelism) or (at best) [partially implicit](https://en.wikipedia.org/wiki/Implicit_parallelism), in which a programmer gives the compiler [directives](https://en.wikipedia.org/wiki/Directive_(programming)) for parallelization. A few fully implicit parallel programming languages exist—[SISAL](https://en.wikipedia.org/wiki/SISAL), Parallel [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)), [SequenceL](https://en.wikipedia.org/wiki/SequenceL), [System C](https://en.wikipedia.org/wiki/SystemC) (for [FPGAs](https://en.wikipedia.org/wiki/FPGA)), [Mitrion-C](https://en.wikipedia.org/w/index.php?title=Mitrion-C&action=edit&redlink=1), [VHDL](https://en.wikipedia.org/wiki/VHDL), and [Verilog](https://en.wikipedia.org/wiki/Verilog).

**Application checkpointing**[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=30)]

*Main article:*[*Application checkpointing*](https://en.wikipedia.org/wiki/Application_checkpointing)

As a computer system grows in complexity, the [mean time between failures](https://en.wikipedia.org/wiki/Mean_time_between_failures) usually decreases. [Application checkpointing](https://en.wikipedia.org/wiki/Application_checkpointing) is a technique whereby the computer system takes a "snapshot" of the application—a record of all current resource allocations and variable states, akin to a [core dump](https://en.wikipedia.org/wiki/Core_dump)—; this information can be used to restore the program if the computer should fail. Application checkpointing means that the program has to restart from only its last checkpoint rather than the beginning. While checkpointing provides benefits in a variety of situations, it is especially useful in highly parallel systems with a large number of processors used in [high performance computing](https://en.wikipedia.org/wiki/High_performance_computing).[[59]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-59)

Algorithmic methods[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=31)]

As parallel computers become larger and faster, we are now able to solve problems that had previously taken too long to run. Fields as varied as [bioinformatics](https://en.wikipedia.org/wiki/Bioinformatics) (for [protein folding](https://en.wikipedia.org/wiki/Protein_folding) and [sequence analysis](https://en.wikipedia.org/wiki/Sequence_analysis)) and economics (for [mathematical finance](https://en.wikipedia.org/wiki/Mathematical_finance)) have taken advantage of parallel computing. Common types of problems in parallel computing applications include:[[60]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-60)

* Dense [linear algebra](https://en.wikipedia.org/wiki/Linear_algebra)
* Sparse linear algebra
* Spectral methods (such as [Cooley–Tukey fast Fourier transform](https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm))
* [*N*-body problems](https://en.wikipedia.org/wiki/N-body_problem) (such as [Barnes–Hut simulation](https://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation))
* [structured grid](https://en.wikipedia.org/wiki/Regular_grid) problems (such as [Lattice Boltzmann methods](https://en.wikipedia.org/wiki/Lattice_Boltzmann_methods))
* [Unstructured grid](https://en.wikipedia.org/wiki/Unstructured_grid) problems (such as found in [finite element analysis](https://en.wikipedia.org/wiki/Finite_element_analysis))
* [Monte Carlo method](https://en.wikipedia.org/wiki/Monte_Carlo_method)
* [Combinational logic](https://en.wikipedia.org/wiki/Combinational_logic) (such as [brute-force cryptographic techniques](https://en.wikipedia.org/wiki/Brute_force_attack))
* [Graph traversal](https://en.wikipedia.org/wiki/Graph_traversal) (such as [sorting algorithms](https://en.wikipedia.org/wiki/Sorting_algorithm))
* [Dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming)
* [Branch and bound](https://en.wikipedia.org/wiki/Branch_and_bound) methods
* [Graphical models](https://en.wikipedia.org/wiki/Graphical_model) (such as detecting [hidden Markov models](https://en.wikipedia.org/wiki/Hidden_Markov_model) and constructing [Bayesian networks](https://en.wikipedia.org/wiki/Bayesian_network))
* [Finite-state machine](https://en.wikipedia.org/wiki/Finite-state_machine) simulation

Fault tolerance[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=32)]

*Further information:*[*Fault-tolerant computer system*](https://en.wikipedia.org/wiki/Fault-tolerant_computer_system)

Parallel computing can also be applied to the design of [fault-tolerant computer systems](https://en.wikipedia.org/wiki/Fault-tolerant_computer_system), particularly via [lockstep](https://en.wikipedia.org/wiki/Lockstep_(computing)) systems performing the same operation in parallel. This provides [redundancy](https://en.wikipedia.org/wiki/Redundancy_(engineering)) in case one component fails, and also allows automatic [error detection](https://en.wikipedia.org/wiki/Error_detection) and [error correction](https://en.wikipedia.org/wiki/Error_correction) if the results differ. These methods can be used to help prevent single-event upsets caused by transient errors.[[61]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-61) Although additional measures may be required in embedded or specialized systems, this method can provide a cost-effective approach to achieve n-modular redundancy in commercial off-the-shelf systems.

History[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=33)]

*Main article:*[*History of computing*](https://en.wikipedia.org/wiki/History_of_computing)

[](https://en.wikipedia.org/wiki/File:ILLIAC_4_parallel_computer.jpg)

[ILLIAC IV](https://en.wikipedia.org/wiki/ILLIAC_IV), "the most infamous of supercomputers"[[62]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-infamous-62)

The origins of true (MIMD) parallelism go back to [Luigi Federico Menabrea](https://en.wikipedia.org/wiki/Luigi_Federico_Menabrea) and his *Sketch of the*[*Analytic Engine*](https://en.wikipedia.org/wiki/Analytic_Engine)*Invented by*[*Charles Babbage*](https://en.wikipedia.org/wiki/Charles_Babbage).[[63]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-63)[[64]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-PH753-64)[[65]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-65)

In April 1958, Stanley Gill (Ferranti) discussed parallel programming and the need for branching and waiting.[[66]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-66) Also in 1958, IBM researchers [John Cocke](https://en.wikipedia.org/wiki/John_Cocke) and [Daniel Slotnick](https://en.wikipedia.org/wiki/Daniel_Slotnick) discussed the use of parallelism in numerical calculations for the first time.[[67]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-G_Wilson-67) [Burroughs Corporation](https://en.wikipedia.org/wiki/Burroughs_Corporation) introduced the D825 in 1962, a four-processor computer that accessed up to 16 memory modules through a [crossbar switch](https://en.wikipedia.org/wiki/Crossbar_switch).[[68]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-68) In 1967, Amdahl and Slotnick published a debate about the feasibility of parallel processing at American Federation of Information Processing Societies Conference.[[67]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-G_Wilson-67) It was during this debate that [Amdahl's law](https://en.wikipedia.org/wiki/Amdahl%27s_law) was coined to define the limit of speed-up due to parallelism.

In 1969, [Honeywell](https://en.wikipedia.org/wiki/Honeywell) introduced its first [Multics](https://en.wikipedia.org/wiki/Multics) system, a symmetric multiprocessor system capable of running up to eight processors in parallel.[[67]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-G_Wilson-67) [C.mmp](https://en.wikipedia.org/wiki/C.mmp), a multi-processor project at [Carnegie Mellon University](https://en.wikipedia.org/wiki/Carnegie_Mellon_University) in the 1970s, was among the first multiprocessors with more than a few processors. The first bus-connected multiprocessor with snooping caches was the [Synapse N+1](https://en.wikipedia.org/w/index.php?title=Synapse_N%2B1&action=edit&redlink=1) in 1984.[[64]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-PH753-64)

SIMD parallel computers can be traced back to the 1970s. The motivation behind early SIMD computers was to amortize the [gate delay](https://en.wikipedia.org/wiki/Propagation_delay) of the processor's [control unit](https://en.wikipedia.org/wiki/Control_unit) over multiple instructions.[[69]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-69) In 1964, Slotnick had proposed building a massively parallel computer for the [Lawrence Livermore National Laboratory](https://en.wikipedia.org/wiki/Lawrence_Livermore_National_Laboratory).[[67]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-G_Wilson-67) His design was funded by the [US Air Force](https://en.wikipedia.org/wiki/US_Air_Force), which was the earliest SIMD parallel-computing effort, [ILLIAC IV](https://en.wikipedia.org/wiki/ILLIAC_IV).[[67]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-G_Wilson-67) The key to its design was a fairly high parallelism, with up to 256 processors, which allowed the machine to work on large datasets in what would later be known as [vector processing](https://en.wikipedia.org/wiki/Vector_processor). However, ILLIAC IV was called "the most infamous of supercomputers", because the project was only one-fourth completed, but took 11 years and cost almost four times the original estimate.[[62]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-infamous-62) When it was finally ready to run its first real application in 1976, it was outperformed by existing commercial supercomputers such as the [Cray-1](https://en.wikipedia.org/wiki/Cray-1).

Biological brain as massively parallel computer[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=34)]

In the early 1970s, at the [MIT Computer Science and Artificial Intelligence Laboratory](https://en.wikipedia.org/wiki/MIT_Computer_Science_and_Artificial_Intelligence_Laboratory), [Marvin Minsky](https://en.wikipedia.org/wiki/Marvin_Minsky) and [Seymour Papert](https://en.wikipedia.org/wiki/Seymour_Papert) started developing the [*Society of Mind*](https://en.wikipedia.org/wiki/Society_of_Mind) theory, which views the biological brain as [massively parallel computer](https://en.wikipedia.org/wiki/Massively_parallel). In 1986, Minsky published *The Society of Mind*, which claims that “mind is formed from many little agents, each mindless by itself”.[[70]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-70) The theory attempts to explain how what we call intelligence could be a product of the interaction of non-intelligent parts. Minsky says that the biggest source of ideas about the theory came from his work in trying to create a machine that uses a robotic arm, a video camera, and a computer to build with children's blocks.[[71]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-71)

Similar models (which also view the biological brain as a massively parallel computer, i.e., the brain is made up of a constellation of independent or semi-independent agents) were also described by:

* Thomas R. Blakeslee,[[72]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-72)
* [Michael S. Gazzaniga](https://en.wikipedia.org/wiki/Michael_Gazzaniga),[[73]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-73)[[74]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-74)
* [Robert E. Ornstein](https://en.wikipedia.org/wiki/Robert_E._Ornstein),[[75]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-75)
* [Ernest Hilgard](https://en.wikipedia.org/wiki/Ernest_Hilgard),[[76]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-76)[[77]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-77)
* [Michio Kaku](https://en.wikipedia.org/wiki/Michio_Kaku),[[78]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-78)
* [George Ivanovich Gurdjieff](https://en.wikipedia.org/wiki/George_Gurdjieff),[[79]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-79)
* Neurocluster Brain Model.[[80]](https://en.wikipedia.org/wiki/Parallel_computing#cite_note-80)

See also[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=35)]

* [Computer multitasking](https://en.wikipedia.org/wiki/Computer_multitasking)
* [Concurrency (computer science)](https://en.wikipedia.org/wiki/Concurrency_(computer_science))
* [Content Addressable Parallel Processor](https://en.wikipedia.org/wiki/Content_Addressable_Parallel_Processor)
* [List of distributed computing conferences](https://en.wikipedia.org/wiki/List_of_distributed_computing_conferences)
* [List of important publications in concurrent, parallel, and distributed computing](https://en.wikipedia.org/wiki/List_of_important_publications_in_concurrent,_parallel,_and_distributed_computing)
* [Manchester dataflow machine](https://en.wikipedia.org/wiki/Dataflow_architecture)
* [Manycore](https://en.wikipedia.org/wiki/Manycore)
* [Parallel programming model](https://en.wikipedia.org/wiki/Parallel_programming_model)
* [Serializability](https://en.wikipedia.org/wiki/Serializability)
* [Synchronous programming](https://en.wikipedia.org/wiki/Synchronous_programming)
* [Transputer](https://en.wikipedia.org/wiki/Transputer)
* [Vector processing](https://en.wikipedia.org/wiki/Vector_processing)

References[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=36)]

* 1. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-1) *Gottlieb, Allan; Almasi, George S. (1989).*[*Highly parallel computing*](http://dl.acm.org/citation.cfm?id=160438)*. Redwood City, Calif.: Benjamin/Cummings.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-8053-0177-9*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-8053-0177-9)*.*
  2. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-:0_2-0) S.V. Adve *et al.* (November 2008). ["Parallel Computing Research at Illinois: The UPCRC Agenda"](https://graphics.cs.illinois.edu/sites/default/files/upcrc-wp.pdf) [Archived](https://web.archive.org/web/20180111165735/https:/graphics.cs.illinois.edu/sites/default/files/upcrc-wp.pdf) 2018-01-11 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine) (PDF). Parallel@Illinois, University of Illinois at Urbana-Champaign. "The main techniques for these performance benefits—increased clock frequency and smarter but increasingly complex architectures—are now hitting the so-called power wall. The [computer industry](https://en.wikipedia.org/wiki/Computer_industry) has accepted that future performance increases must largely come from increasing the number of processors (or cores) on a die, rather than making a single core go faster."
  3. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-3) Asanovic *et al.* Old [conventional wisdom]: Power is free, but [transistors](https://en.wikipedia.org/wiki/Transistor) are expensive. New [conventional wisdom] is [that] power is expensive, but transistors are "free".
  4. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-View-Power_4-0) Asanovic, Krste *et al.* (December 18, 2006). ["The Landscape of Parallel Computing Research: A View from Berkeley"](http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf) (PDF). University of California, Berkeley. Technical Report No. UCB/EECS-2006-183. "Old [conventional wisdom]: Increasing clock frequency is the primary method of improving processor performance. New [conventional wisdom]: Increasing parallelism is the primary method of improving processor performance… Even representatives from Intel, a company generally associated with the 'higher clock-speed is better' position, warned that traditional approaches to maximizing performance through maximizing clock speed have been pushed to their limits."
  5. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-waza_5-0) "Concurrency is not Parallelism", *Waza conference* Jan 11, 2012, [Rob Pike](https://en.wikipedia.org/wiki/Rob_Pike) ([slides](https://talks.golang.org/2012/waza.slide)) ([video](https://vimeo.com/49718712))
  6. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-6) [*"Parallelism vs. Concurrency"*](https://wiki.haskell.org/Parallelism_vs._Concurrency)*. Haskell Wiki.*
  7. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-7) [*Hennessy, John L.*](https://en.wikipedia.org/wiki/John_L._Hennessy)*;*[*Patterson, David A.*](https://en.wikipedia.org/wiki/David_Patterson_(computer_scientist))*;*[*Larus, James R.*](https://en.wikipedia.org/wiki/James_Larus)*(1999).*[*Computer organization and design: the hardware/software interface*](https://archive.org/details/computerorganiz000henn)*(2. ed., 3rd print. ed.). San Francisco: Kaufmann.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-55860-428-5*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-55860-428-5)*.*
  8. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-llnltut_8-0) [***b***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-llnltut_8-1) *Barney, Blaise.*[*"Introduction to Parallel Computing"*](https://www.llnl.gov/computing/tutorials/parallel_comp/)*. Lawrence Livermore National Laboratory. Retrieved 2007-11-09.*
  9. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-9) *Thomas Rauber; Gudula Rünger (2013). Parallel Programming: for Multicore and Cluster Systems. Springer Science & Business Media. p. 1.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*9783642378010*](https://en.wikipedia.org/wiki/Special:BookSources/9783642378010)*.*
  10. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-10) *Hennessy, John L.; Patterson, David A. (2002). Computer architecture / a quantitative approach (3rd ed.). San Francisco, Calif.: International Thomson. p. 43.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-55860-724-8*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-55860-724-8)*.*
  11. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-11) *Rabaey, Jan M. (1996). Digital integrated circuits : a design perspective. Upper Saddle River, N.J.: Prentice-Hall. p. 235.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-13-178609-7*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-13-178609-7)*.*
  12. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-12) *Flynn, Laurie J. (8 May 2004).*[*"Intel Halts Development Of 2 New Microprocessors"*](https://www.nytimes.com/2004/05/08/business/08chip.html?ex=1399348800&en=98cc44ca97b1a562&ei=5007)*. New York Times. Retrieved 5 June 2012.*
  13. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-13) *Thomas Rauber; Gudula Rünger (2013). Parallel Programming: for Multicore and Cluster Systems. Springer Science & Business Media. p. 2.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*9783642378010*](https://en.wikipedia.org/wiki/Special:BookSources/9783642378010)*.*
  14. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-14) *Thomas Rauber; Gudula Rünger (2013). Parallel Programming: for Multicore and Cluster Systems. Springer Science & Business Media. p. 3.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*9783642378010*](https://en.wikipedia.org/wiki/Special:BookSources/9783642378010)*.*
  15. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-15) *Amdahl, Gene M. (1967).*[*"Validity of the single processor approach to achieving large scale computing capabilities"*](http://dl.acm.org/citation.cfm?id=160438)*. Proceeding AFIPS '67 (Spring) Proceedings of the April 18–20, 1967, Spring Joint Computer Conference: 483–485.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/1465482.1465560*](https://doi.org/10.1145%2F1465482.1465560)*.*
  16. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-16) *Brooks, Frederick P. (1996).*[*The mythical man month essays on software engineering*](https://archive.org/details/mythicalmonth00broo)*(Anniversary ed., repr. with corr., 5. [Dr.] ed.). Reading, Mass. [u.a.]: Addison-Wesley.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-201-83595-3*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-201-83595-3)*.*
  17. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-17) *Michael McCool; James Reinders; Arch Robison (2013). Structured Parallel Programming: Patterns for Efficient Computation. Elsevier. p. 61.*
  18. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-18) *Gustafson, John L. (May 1988).*[*"Reevaluating Amdahl's law"*](https://web.archive.org/web/20070927040654/http:/www.scl.ameslab.gov/Publications/Gus/AmdahlsLaw/Amdahls.html)*. Communications of the ACM.****31****(5): 532–533.*[*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.509.6892*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.509.6892)*.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/42411.42415*](https://doi.org/10.1145%2F42411.42415)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*33937392*](https://api.semanticscholar.org/CorpusID:33937392)*. Archived from*[*the original*](http://www.scl.ameslab.gov/Publications/Gus/AmdahlsLaw/Amdahls.html)*on 2007-09-27.*
  19. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-19) *Bernstein, A. J. (1 October 1966). "Analysis of Programs for Parallel Processing". IEEE Transactions on Electronic Computers. EC-15 (5): 757–763.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/PGEC.1966.264565*](https://doi.org/10.1109%2FPGEC.1966.264565)*.*
  20. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-20) *Roosta, Seyed H. (2000). Parallel processing and parallel algorithms : theory and computation. New York, NY [u.a.]: Springer. p. 114.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-387-98716-3*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-387-98716-3)*.*
  21. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-21) [*"Processes and Threads"*](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684841(v=vs.85).aspx)*. Microsoft Developer Network. Microsoft Corp. 2018. Retrieved 2018-05-10.*
  22. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-22) *Krauss, Kirk J (2018).*[*"Thread Safety for Performance"*](http://www.developforperformance.com/ThreadSafetyForPerformance.html)*. Develop for Performance. Retrieved 2018-05-10.*
  23. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-23) *Tanenbaum, Andrew S. (2002-02-01).*[*Introduction to Operating System Deadlocks*](http://www.informit.com/articles/article.aspx?p=25193)*. Informit. Pearson Education, Informit. Retrieved 2018-05-10.*
  24. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-24) *Cecil, David (2015-11-03).*[*"Synchronization internals – the semaphore"*](https://www.embedded.com/design/operating-systems/4440752/Synchronization-internals----the-semaphore)*. Embedded. AspenCore. Retrieved 2018-05-10.*
  25. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-25) *Preshing, Jeff (2012-06-08).*[*"An Introduction to Lock-Free Programming"*](http://preshing.com/20120612/an-introduction-to-lock-free-programming/)*. Preshing on Programming. Retrieved 2018-05-10.*
  26. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-26) [*"What's the opposite of "embarrassingly parallel"?"*](https://stackoverflow.com/questions/806569/whats-the-opposite-of-embarrassingly-parallel)*. StackOverflow. Retrieved 2018-05-10.*
  27. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-27) *Schwartz, David (2011-08-15).*[*"What is thread contention?"*](https://stackoverflow.com/questions/1970345/what-is-thread-contention)*. StackOverflow. Retrieved 2018-05-10.*
  28. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-28) *Kukanov, Alexey (2008-03-04).*[*"Why a simple test can get parallel slowdown"*](https://software.intel.com/en-us/blogs/2008/03/04/why-a-simple-test-can-get-parallel-slowdown)*. Retrieved 2015-02-15.*
  29. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-29) *Krauss, Kirk J (2018).*[*"Threading for Performance"*](http://www.developforperformance.com/ThreadingForPerformance.html)*. Develop for Performance. Retrieved 2018-05-10.*
  30. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-30) *Lamport, Leslie (1 September 1979). "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs". IEEE Transactions on Computers.****C-28****(9): 690–691.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/TC.1979.1675439*](https://doi.org/10.1109%2FTC.1979.1675439)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*5679366*](https://api.semanticscholar.org/CorpusID:5679366)*.*
  31. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-31) Patterson and Hennessy, p. 748.
  32. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-32) *Singh, David Culler; J.P. (1997). Parallel computer architecture([Nachdr.] ed.). San Francisco: Morgan Kaufmann Publ. p. 15.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-55860-343-1*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-55860-343-1)*.*
  33. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-33) Culler et al. p. 15.
  34. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-34) [Patt, Yale](https://en.wikipedia.org/wiki/Yale_Patt) (April 2004). "[The Microprocessor Ten Years From Now: What Are The Challenges, How Do We Meet Them?](http://users.ece.utexas.edu/~patt/Videos/talk_videos/cmu_04-29-04.wmv) [Archived](https://web.archive.org/web/20080414141000/http:/users.ece.utexas.edu/~patt/Videos/talk_videos/cmu_04-29-04.wmv)2008-04-14 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine) (wmv). Distinguished Lecturer talk at [Carnegie Mellon University](https://en.wikipedia.org/wiki/Carnegie_Mellon_University). Retrieved on November 7, 2007.
  35. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-Culler124_35-0) Culler et al. p. 124.
  36. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-Culler125_36-0) Culler et al. p. 125.
  37. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-37) *Samuel Larsen; Saman Amarasinghe.*[*"Exploiting Superword Level Parallelism with Multimedia Instruction Sets"*](http://groups.csail.mit.edu/cag/slp/SLP-PLDI-2000.pdf)*(PDF).*
  38. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-PH713_38-0) [***b***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-PH713_38-1) Patterson and Hennessy, p. 713.
  39. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-HP549_39-0) [***b***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-HP549_39-1) Hennessy and Patterson, p. 549.
  40. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-40) Patterson and Hennessy, p. 714.
  41. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-41) [Ghosh (2007)](https://en.wikipedia.org/wiki/Distributed_computing#CITEREFGhosh2007), p. 10. [Keidar (2008)](https://en.wikipedia.org/wiki/Distributed_computing#CITEREFKeidar2008).
  42. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-42) [Lynch (1996)](https://en.wikipedia.org/wiki/Distributed_computing#CITEREFLynch1996), p. xix, 1–2. [Peleg (2000)](https://en.wikipedia.org/wiki/Distributed_computing#CITEREFPeleg2000), p. 1.
  43. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-43) [What is clustering?](http://www.webopedia.com/TERM/c/clustering.html) Webopedia computer dictionary. Retrieved on November 7, 2007.
  44. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-44) [Beowulf definition.](https://www.pcmag.com/encyclopedia_term/0,2542,t=Beowulf&i=38548,00.asp) *PC Magazine*. Retrieved on November 7, 2007.
  45. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-45) [*"List Statistics | TOP500 Supercomputer Sites"*](https://www.top500.org/statistics/list/)*. www.top500.org. Retrieved 2018-08-05.*
  46. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-46) ["Interconnect"](https://www.nersc.gov/users/computational-systems/hopper/configuration/interconnect/) [Archived](https://web.archive.org/web/20150128133120/https:/www.nersc.gov/users/computational-systems/hopper/configuration/interconnect/) 2015-01-28 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine).
  47. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-47) Hennessy and Patterson, p. 537.
  48. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-48) [MPP Definition.](https://www.pcmag.com/encyclopedia_term/0,,t=mpp&i=47310,00.asp) *PC Magazine*. Retrieved on November 7, 2007.
  49. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-49) *Kirkpatrick, Scott (2003). "COMPUTER SCIENCE: Rough Times Ahead". Science.****299****(5607): 668–669.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1126/science.1081623*](https://doi.org/10.1126%2Fscience.1081623)*.*[*PMID*](https://en.wikipedia.org/wiki/PMID_(identifier))[*12560537*](https://pubmed.ncbi.nlm.nih.gov/12560537)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*60622095*](https://api.semanticscholar.org/CorpusID:60622095)*.*
  50. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-DAmour_50-0) [***b***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-DAmour_50-1) [***c***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-DAmour_50-2) D'Amour, Michael R., Chief Operating Officer, [DRC Computer Corporation](https://en.wikipedia.org/w/index.php?title=DRC_Computer_Corporation&action=edit&redlink=1). "Standard Reconfigurable Computing". Invited speaker at the University of Delaware, February 28, 2007.
  51. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-51) Boggan, Sha'Kia and Daniel M. Pressel (August 2007). [GPUs: An Emerging Platform for General-Purpose Computation](http://www.arl.army.mil/arlreports/2007/ARL-SR-154.pdf)[Archived](https://web.archive.org/web/20161225073248/http:/www.arl.army.mil/arlreports/2007/ARL-SR-154.pdf) 2016-12-25 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine) (PDF). ARL-SR-154, U.S. Army Research Lab. Retrieved on November 7, 2007.
  52. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-52) Maslennikov, Oleg (2002). ["Systematic Generation of Executing Programs for Processor Elements in Parallel ASIC or FPGA-Based Systems and Their Transformation into VHDL-Descriptions of Processor Element Control Units".](https://doi.org/10.1007%2F3-540-48086-2_30) *Lecture Notes in Computer Science*, **2328/2002:** p. 272.
  53. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-53) *Shimokawa, Y.; Fuwa, Y.; Aramaki, N. (18–21 November 1991). "A parallel ASIC VLSI neurocomputer for a large number of neurons and billion connections per second speed". International Joint Conference on Neural Networks.****3****: 2162–2167.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/IJCNN.1991.170708*](https://doi.org/10.1109%2FIJCNN.1991.170708)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-7803-0227-3*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-7803-0227-3)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*61094111*](https://api.semanticscholar.org/CorpusID:61094111)*.*
  54. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-54) *Acken, Kevin P.; Irwin, Mary Jane; Owens, Robert M. (July 1998). "A Parallel ASIC Architecture for Efficient Fractal Image Coding". The Journal of VLSI Signal Processing.****19****(2): 97–113.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1023/A:1008005616596*](https://doi.org/10.1023%2FA%3A1008005616596)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*2976028*](https://api.semanticscholar.org/CorpusID:2976028)*.*
  55. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-55) Kahng, Andrew B. (June 21, 2004) "[Scoping the Problem of DFM in the Semiconductor Industry](http://www.future-fab.com/documents.asp?grID=353&d_ID=2596) [Archived](https://web.archive.org/web/20080131221732/http:/www.future-fab.com/documents.asp?grID=353&d_ID=2596) 2008-01-31 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine)." University of California, San Diego. "Future design for manufacturing (DFM) technology must reduce design [non-recoverable expenditure] cost and directly address manufacturing [non-recoverable expenditures]—the cost of a mask set and probe card—which is well over $1 million at the 90 nm technology node and creates a significant damper on semiconductor-based innovation."
  56. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-PH751_56-0) [***b***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-PH751_56-1) Patterson and Hennessy, p. 751.
  57. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-57) The [Sidney Fernbach Award given to MPI inventor Bill Gropp](http://awards.computer.org/ana/award/viewPastRecipients.action?id=16)[Archived](https://web.archive.org/web/20110725191103/http:/awards.computer.org/ana/award/viewPastRecipients.action?id=16) 2011-07-25 at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine) refers to MPI as "the dominant HPC communications interface"
  58. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-58) *Shen, John Paul; Mikko H. Lipasti (2004). Modern processor design : fundamentals of superscalar processors (1st ed.). Dubuque, Iowa: McGraw-Hill. p. 561.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-07-057064-1*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-07-057064-1)*. However, the holy grail of such research—automated parallelization of serial programs—has yet to materialize. While automated parallelization of certain classes of algorithms has been demonstrated, such success has largely been limited to scientific and numeric applications with predictable flow control (e.g., nested loop structures with statically determined iteration counts) and statically analyzable memory access patterns. (e.g., walks over large multidimensional arrays of float-point data).*
  59. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-59) *Encyclopedia of Parallel Computing, Volume 4* by David Padua 2011 [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0387097651](https://en.wikipedia.org/wiki/Special:BookSources/0387097651) page 265
  60. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-60) Asanovic, Krste, et al. (December 18, 2006). ["The Landscape of Parallel Computing Research: A View from Berkeley"](http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf) (PDF). University of California, Berkeley. Technical Report No. UCB/EECS-2006-183. See table on pages 17–19.
  61. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-61) Dobel, B., Hartig, H., & Engel, M. (2012) "Operating system support for redundant multithreading". *Proceedings of the Tenth ACM International Conference on Embedded Software*, 83–92. [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1145/2380356.2380375](https://doi.org/10.1145%2F2380356.2380375)
  62. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-infamous_62-0) [***b***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-infamous_62-1) Patterson and Hennessy, pp. 749–50: "Although successful in pushing several technologies useful in later projects, the ILLIAC IV failed as a computer. Costs escalated from the $8 million estimated in 1966 to $31 million by 1972, despite the construction of only a quarter of the planned machine . It was perhaps the most infamous of supercomputers. The project started in 1965 and ran its first real application in 1976."
  63. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-63) [Menabrea, L. F.](https://en.wikipedia.org/wiki/Luigi_Federico_Menabrea) (1842). [*Sketch of the Analytic Engine Invented by Charles Babbage*](http://www.fourmilab.ch/babbage/sketch.html). Bibliothèque Universelle de Genève. Retrieved on November 7, 2007. quote: "when a long series of identical computations is to be performed, such as those required for the formation of numerical tables, the machine can be brought into play so as to give several results at the same time, which will greatly abridge the whole amount of the processes."
  64. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-PH753_64-0) [***b***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-PH753_64-1) Patterson and Hennessy, p. 753.
  65. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-65) R.W. Hockney, C.R. Jesshope. [*Parallel Computers 2: Architecture, Programming and Algorithms, Volume 2*](https://books.google.com/books?id=6HcBQ67-Fb4C). 1988. p. 8 quote: "The earliest reference to parallelism in computer design is thought to be in General L. F. Menabrea's publication in… 1842, entitled *Sketch of the Analytical Engine Invented by Charles Babbage*".
  66. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-66) "Parallel Programming", S. Gill, *The Computer Journal* Vol. 1 #1, pp2-10, British Computer Society, April 1958.
  67. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-G_Wilson_67-0) [***b***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-G_Wilson_67-1) [***c***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-G_Wilson_67-2) [***d***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-G_Wilson_67-3) [***e***](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-G_Wilson_67-4) *Wilson, Gregory V. (1994).*[*"The History of the Development of Parallel Computing"*](http://ei.cs.vt.edu/~history/Parallel.html)*. Virginia Tech/Norfolk State University, Interactive Learning with a Digital Library in Computer Science. Retrieved 2008-01-08.*
  68. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-68) *Anthes, Gry (November 19, 2001).*[*"The Power of Parallelism"*](https://web.archive.org/web/20080131205427/http:/www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=65878)*.*[*Computerworld*](https://en.wikipedia.org/wiki/Computerworld)*. Archived from*[*the original*](http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=65878)*on January 31, 2008. Retrieved 2008-01-08.*
  69. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-69) Patterson and Hennessy, p. 749.
  70. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-70) *Minsky, Marvin (1986).*[*The Society of Mind*](https://archive.org/details/societyofmind00marv/page/17)*. New York: Simon & Schuster. pp.*[*17*](https://archive.org/details/societyofmind00marv/page/17)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-671-60740-1*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-671-60740-1)*.*
  71. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-71) *Minsky, Marvin (1986).*[*The Society of Mind*](https://archive.org/details/societyofmind00marv/page/29)*. New York: Simon & Schuster. pp.*[*29*](https://archive.org/details/societyofmind00marv/page/29)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-671-60740-1*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-671-60740-1)*.*
  72. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-72) [*Blakeslee, Thomas*](https://en.wikipedia.org/w/index.php?title=Thomas_R._Blakeslee&action=edit&redlink=1)*(1996).*[*Beyond the Conscious Mind. Unlocking the Secrets of the Self*](https://archive.org/details/beyondconsciousm00blak)*. pp.*[*6–7*](https://archive.org/details/beyondconsciousm00blak/page/6)*.*
  73. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-73) [*Gazzaniga, Michael*](https://en.wikipedia.org/wiki/Michael_S._Gazzaniga)*;*[*LeDoux, Joseph*](https://en.wikipedia.org/wiki/Joseph_E._LeDoux)*(1978). The Integrated Mind. pp. 132–161.*
  74. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-74) [*Gazzaniga, Michael*](https://en.wikipedia.org/wiki/Michael_S._Gazzaniga)*(1985).*[*The Social Brain. Discovering the Networks of the Mind*](https://archive.org/details/socialbraindisco0000gazz)*. pp.*[*77–79*](https://archive.org/details/socialbraindisco0000gazz/page/77)*.*
  75. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-75) [*Ornstein, Robert*](https://en.wikipedia.org/wiki/Robert_Ornstein)*(1992).*[*Evolution of Consciousness: The Origins of the Way We Think*](https://archive.org/details/evolutionofconsc0000orns)*. pp.*[*2*](https://archive.org/details/evolutionofconsc0000orns/page/2)*.*
  76. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-76) [*Hilgard, Ernest*](https://en.wikipedia.org/w/index.php?title=Ernest_Ropiequet_%22Jack%22_Hilgard&action=edit&redlink=1)*(1977). Divided consciousness: multiple controls in human thought and action. New York: Wiley.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-471-39602-4*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-471-39602-4)*.*
  77. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-77) [*Hilgard, Ernest*](https://en.wikipedia.org/w/index.php?title=Ernest_Ropiequet_%22Jack%22_Hilgard&action=edit&redlink=1)*(1986). Divided consciousness: multiple controls in human thought and action (expanded edition). New York: Wiley.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-471-80572-4*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-471-80572-4)*.*
  78. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-78) [*Kaku, Michio*](https://en.wikipedia.org/wiki/Michio_Kaku)*(2014).*[*The Future of the Mind*](https://en.wikipedia.org/wiki/The_Future_of_the_Mind)*.*
  79. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-79) [*Ouspenskii, Pyotr*](https://en.wikipedia.org/wiki/Pyotr_Demianovich_Ouspenskii)*(1992). "Chapter 3". In Search of the Miraculous. Fragments of an Unknown Teaching. pp. 72–83.*
  80. [**^**](https://en.wikipedia.org/wiki/Parallel_computing#cite_ref-80) [*"Official Neurocluster Brain Model site"*](http://neuroclusterbrain.com/)*. Retrieved July 22,2017.*

Further reading[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=37)]

* *Rodriguez, C.; Villagra, M.; Baran, B. (29 August 2008). "Asynchronous team algorithms for Boolean Satisfiability". Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd: 66–69.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/BIMNICS.2007.4610083*](https://doi.org/10.1109%2FBIMNICS.2007.4610083)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*15185219*](https://api.semanticscholar.org/CorpusID:15185219)*.*
* Sechin, A.; Parallel Computing in Photogrammetry. GIM International. #1, 2016, pp. 21–23.

External links[[edit](https://en.wikipedia.org/w/index.php?title=Parallel_computing&action=edit&section=38)]

**Listen to this article**



MENU

0:00

Spoken Wikipedia icon

[This audio file](https://en.wikipedia.org/wiki/File:En-Parallel_computing.ogg) was created from a revision of this article dated 2013-08-21, and does not reflect subsequent edits.

(

* [Audio help](https://en.wikipedia.org/wiki/Wikipedia:Media_help)
* [More spoken articles](https://en.wikipedia.org/wiki/Wikipedia:Spoken_articles)

)

|  |  |
| --- | --- |
|  | Wikibooks has a book on the topic of: [***Distributed Systems***](https://en.wikibooks.org/wiki/Distributed_Systems) |

* [Instructional videos on CAF in the Fortran Standard by John Reid (see Appendix B)](http://www.nd.edu/~dbalsara/Numerical-PDE-Course)
* [Parallel computing](https://curlie.org/Computers/Parallel_Computing/) at [Curlie](https://en.wikipedia.org/wiki/Curlie)
* [Lawrence Livermore National Laboratory: Introduction to Parallel Computing](https://www.llnl.gov/computing/tutorials/parallel_comp/)
* [Designing and Building Parallel Programs, by Ian Foster](http://www-unix.mcs.anl.gov/dbpp/)
* [Internet Parallel Computing Archive](https://web.archive.org/web/20021012122919/http:/wotug.ukc.ac.uk/parallel/)
* [Parallel processing topic area at IEEE Distributed Computing Online](https://web.archive.org/web/20110928211245/http:/dsonline.computer.org/portal/site/dsonline/index.jsp?pageID=dso_level1_home&path=dsonline%2Ftopics%2Fparallel&file=index.xml&xsl=generic.xsl)
* [Parallel Computing Works Free On-line Book](http://www.new-npac.org/projects/cdroms/cewes-1998-05/copywrite/pcw/book.html)
* [Frontiers of Supercomputing Free On-line Book Covering topics like algorithms and industrial applications](http://ark.cdlib.org/ark:/13030/ft0f59n73z/)
* [Universal Parallel Computing Research Center](https://web.archive.org/web/20081020052247/http:/www.upcrc.illinois.edu/)
* [Course in Parallel Programming at Columbia University (in collaboration with IBM T.J. Watson X10 project)](https://web.archive.org/web/20100122110043/http:/ppppcourse.ning.com/)
* [Parallel and distributed Gröbner bases computation in JAS](https://arxiv.org/PS_cache/arxiv/pdf/1008/1008.0011v1.pdf), see also [Gröbner basis](https://en.wikipedia.org/wiki/Gr%C3%B6bner_basis)
* [Course in Parallel Computing at University of Wisconsin-Madison](https://web.archive.org/web/20110826190010/http:/sbel.wisc.edu/Courses/ME964/2011/index.htm)
* [Berkeley Par Lab: progress in the parallel computing landscape](http://research.microsoft.com/en-us/collaboration/parlab/contents.aspx), Editors: David Patterson, Dennis Gannon, and Michael Wrinn, August 23, 2013
* [The trouble with multicore](https://spectrum.ieee.org/computing/software/the-trouble-with-multicore), by David Patterson, posted 30 Jun 2010
* [The Landscape of Parallel Computing Research: A View From Berkeley](https://web.archive.org/web/20080705232106/http:/view.eecs.berkeley.edu/) (one too many dead link at this site)
* [Introduction to Parallel Computing](https://computing.llnl.gov/tutorials/parallel_comp/)
* [Coursera: Parallel Programming](https://www.coursera.org/learn/parprog1)