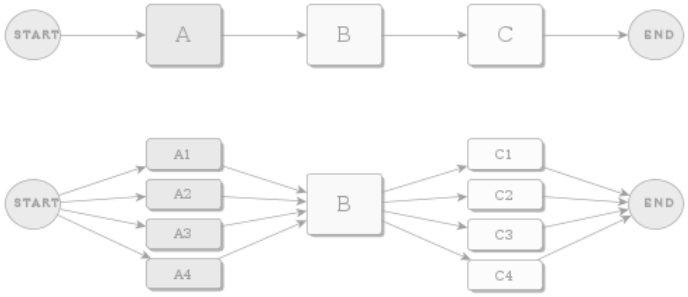
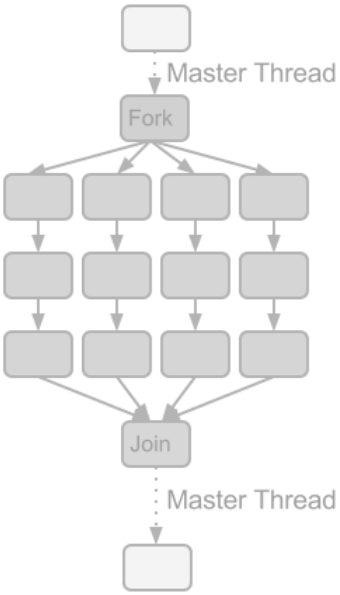
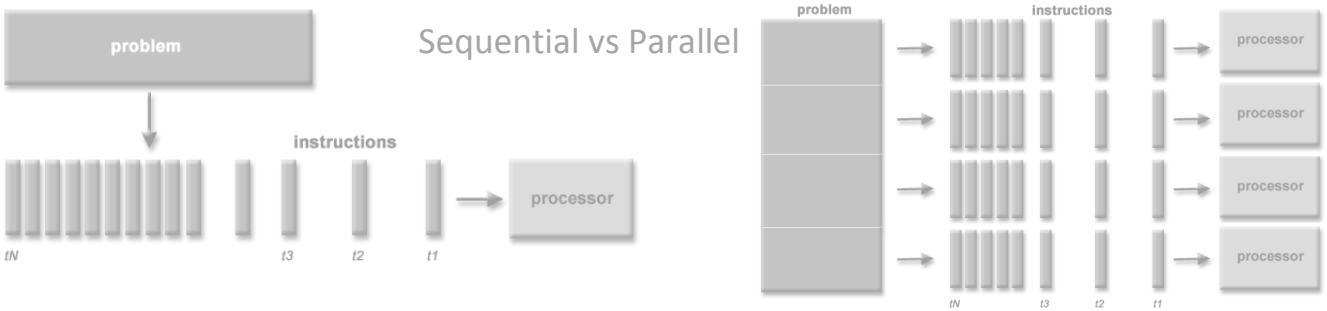
Chapter 1.

Parallel Computing: *An Introduction*







1. Introduction

The past few decades has seen tremendous advances in microprocessor technology. Clock rates of processor have increased rapidly from a few MHz to over 4.0 GHz [1]. At the same time, processors are now capable of executing multiple instructions in the same cycle. A variety of other issues have also become important over same period. Perhaps the most prominent of these is the ability of the memory system to feed data to the processor at the required rate. Parallel platform typically yield better memory system performance because they provide large aggregate caches and higher aggregate bandwidth to the memory system. Furthermore, the principals that are the heart of parallel algorithms, namely locality of reference, also lend themselves to cache friendly serial algorithms. This argument can be extended for external (secondary memory) algorithms.

Parallel algorithm has made tremendous impact on variety of areas ranging from computational simulation for scientific and engineering applications, to commercial applications in data mining and transaction processing.

1.1. Background

Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors.

Parallel computers can be roughly classified [2] according to the level at which the hardware supports parallelism, i) with multi-core and multi-processor computers having multiple processing elements within a single machine, ii) while clusters, MPPs, and grids use multiple computers to work on the same task. Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks.

Parallel computer programs are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically some of the greatest obstacles to getting good parallel program performance.

The maximum possible speed-up of a single program, as a result of parallelization, is known as **Amdahl's law**. It states that a small portion of the program which cannot be parallelized will limit the overall *speedup* available from parallelization; where **speedup** is a measure that captures the relative benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem on a single processor element to the time required to solve the same problem on parallel computer with some identical processor elements.

1.2. Flynn's Taxonomy

Flynn's taxonomy is a classification of computer architectures, proposed by Michael J. Flynn in 1966. The classification system has stuck, and has been used as a tool in design of modern processors and their functionalities. Since the rise of multiprocessing CPUs, a multiprogramming context has evolved as an extension of the classification system. The classifications defined by Flynn are based upon the number of concurrent instruction (or control) and data streams available in the architecture:

Single Instruction, Single Data stream (SISD): A sequential computer, which exploits no parallelism in either the instruction or the data streams. Single control unit (CU) fetches single Instruction Stream (IS) from memory. The CU then generates appropriate control signals to direct single processing element (PE) to operate on single Data Stream (DS) i.e. one operation at a time. Examples of SISD architecture are the traditional uniprocessor machines like a PC (currently manufactured PCs have multiple cores) or old mainframes.

Single Instruction, Multiple Data streams (SIMD): A computer, which exploits multiple data streams against a single instruction stream to perform operations that may be naturally parallelized. For example, an array processor or GPU (Graphics Processor Unit).

Multiple Instruction, Single Data stream (MISD): Multiple instructions operate on a single data stream. Uncommon architecture, which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.

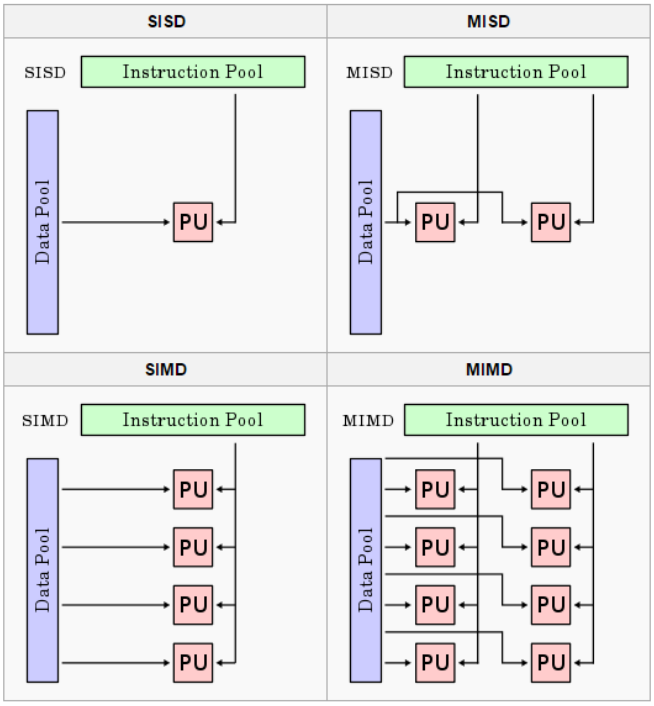


Figure 1.1: Flynn's Taxonomy

Multiple Instruction, Multiple Data streams (MIMD): Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space. A multi-core superscalar processor is an MIMD processor.

Table 1.1

	Single instruction	**Multiple instruction**	**Single program**	**Multiple program**
Single data	SISD	MISD		
Multiple data	SIMD	MIMD	SPMD	MPMD

We can further divide the MIMD category into the two categories:

Single Program, Multiple Data (SPMD): Multiple autonomous processors simultaneously executing the same program (but at independent points, rather than in the lockstep that SIMD imposes) on different data. Also referred to as 'Single Process, multiple data' - the use of this terminology for SPMD is erroneous and should be avoided, as SPMD is a parallel execution model and assumes multiple cooperating processes executing a program. SPMD is the most common style of parallel programming.

Multiple Program, Multiple Data (MPMD): Multiple autonomous processors simultaneously operating at least 2 independent programs. Typically such systems pick one node to be the "host" ("the explicit host/node programming model") or "manager" (the "Manager/Worker" strategy), which runs one program that farms out data to all the other nodes which all run a second program. Those other nodes then return their results directly to the manager. An example of this would be the Sony PlayStation 3 game console, with its SPU/PPU processor architecture.

1.3. Types of Parallelism

Parallelism can be classified as follows:

Bit-level parallelism: From the advent of 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. Word size is the amount of information the processor can manipulate per cycle. 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 processor must add two 16-bit integers, 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 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 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 recently (2003–2004), with the advent of x86-64 architectures, have 64-bitprocessors become commonplace.

Instruction-level Parallelism: A computer program, is in essence, a stream of instructions executed by a processor. These instructions can be re-ordered 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.

Modern processors have multi-stage instruction pipelines. 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. The canonical example of a pipelined processor is a RISC processor, with five stages: instruction fetch, decode, execute, memory access, and write back. The Pentium 4 processor had a 35-stage pipeline.

A canonical five-stage pipeline in a RISC machine (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).

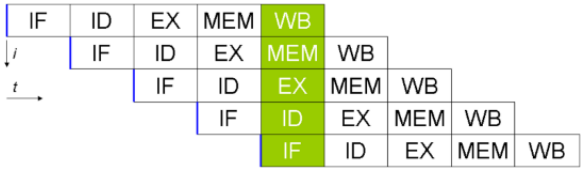


Figure 1.2: Instruction pipelining

In addition to instruction-level parallelism from pipelining, some processors can issue more than one instruction at a time. These are known as superscalar processors. Instructions can be grouped together only if there is no data dependency between them.

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". 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 simultaneously and often cooperatively. Task parallelism does not usually scale with the size of a problem.

1.4. Parallel Programming Model

In computer software, a parallel programming model is a model for writing parallel programs which can be compiled and executed. 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 they execute. The implementation of a programming model can take several forms such as libraries invoked from traditional sequential languages, language extensions, or complete new execution models.

Consensus around each programming model is important as it enables software expressed within it to be transportable between different architectures. For sequential programming architectures, the von Neumann model has facilitated this, as it provides an efficient bridge between hardware and software, meaning that high-level languages can be efficiently compiled to it and it can be efficiently implemented in hardware.

Parallel Programming Model can be classified in many ways.

1.4.1. Process Interaction

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 it can also be implicit.

Shared memory: Shared memory is an efficient means of passing data between programs. Depending on context, programs may run on a single processor or on multiple separate processors. In this model, parallel tasks share a global address space which they read and write to asynchronously. This requires protection mechanisms such as locks, semaphores and monitors to control concurrent access. Shared memory can be emulated on distributed-memory systems but non-uniform memory access (NUMA) times can come in to play. Sometimes memory is also shared between different sections of code of the same program. E.g. A For loop can create threads for each iteration which updates a variable in parallel.

Message passing: Message passing is a concept from computer science that is used extensively in the design and implementation of modern software applications; it is key to some models of concurrency and object-oriented programming. In a message passing model, parallel tasks exchange data through passing messages to one another. These communications can be asynchronous or synchronous. The Communicating Sequential Processes (CSP) formalization of message-passing employed communication channels to 'connect' processes, and led to a number of important languages such as Joyce, occam and Erlang.

Implicit: In an implicit model, no process interaction is visible to the programmer, instead the compiler and/or runtime is responsible for performing it. This is most common with domain-specific languages where the concurrency within a problem can be more prescribed.

1.4.2. Problem Decomposition

A parallel program is composed of simultaneously executing processes. Problem decomposition relates to the way in which these processes are formulated. This classification may also be referred to as algorithmic skeletons or parallel programming paradigms.

Task parallelism: A task-parallel model focuses on processes, or threads of execution. These processes will often be behaviorally distinct, which emphasizes the need for communication. Task parallelism is a natural way to express message-passing communication. It is usually classified as MIMD/MPMD or MISD.

Data parallelism: A data-parallel model focuses on performing operations on a data set which is usually regularly structured in an array. A set of tasks will operate on this data, but independently on separate partitions. In a shared memory system, the data will be accessible to all, but in a distributed-memory system it will divided between memories and worked on locally. Data parallelism is usually classified as SIMD/SPMD.

1.4.3. PRAM Model

A natural extension of the serial model of computation consists of some processors and a global memory of unbounded size that is uniformly accessible to all processors. All processors access the same address space. Processors share a common clock but may execute different instructions in same cycle. This ideal model is also referred as a Parallel Random Access Machine (PRAM). PRAMs allow concurrent access to various memory locations, depending on how simultaneous memory accesses are handled. PRAM can be classified into four subclasses.

Exclusive read exclusive write (EREW): In this class, access to a memory location is exclusive. No concurrent read or write operations are allowed. This is the weakest PRAM model, affording minimum concurrency in memory access.

Concurrent read exclusive write (CREW): In this class, multiple read access to a memory location are allowed. However, multiple write accesses to a memory location are serialized.

Exclusive read concurrent write (ERCW): Multiple write accesses are allowed to a memory locations, but multiple read accesses are serialized.

Concurrent read concurrent write (CRCW): This class allows multiple read and write accesses to a common memory location. This is the most powerful PRAM model.

Allowing concurrent read access does not create any semantic discrepancies in the program. However, concurrent write access to a memory location require arbitration. Several protocols are used to resolve concurrent writes. Those are:

Common: in which the concurrent write is allowed if all the values that the processor are attempting to write are identical.

Arbitrary: in which an arbitrary processor is allowed to proceed with the wrote operation nd the rest fail.

Priority: in which all processors are organized into a predefined prioritized list, and the processor with the highest priority succeeds and rest will fail.

Sum: in which the sum of all quantities is written. (The sum based write conflict resolution model can be extended to any associative operator defined on the quantities being written).

Several simplifying assumptions are made while considering the development of algorithms for PRAM. They are:

1. There is no limit on the number of processors in the machine.
2. Any memory location is uniformly accessible from any processor.
3. There is no limit on the amount of shared memory in the system.
4. Resource contention is absent.
5. The programs written on these machines are, in general, of type SIMD.

1.5. Parallel Processing Architecture

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism.

Multicore computing: A multicore processor is a processor that includes multiple execution units ("cores") on the same chip. Multicore processor can issue multiple instructions per cycle from multiple instruction streams.

Symmetric multiprocessing: A symmetric multiprocessor (SMP) is a computer system with multiple identical processors that share memory and connect via a bus. 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.

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.

Cluster 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. In computing and systems design a loosely coupled system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of other separate components. Sub-areas include the coupling of classes, interfaces, data, and services.

Clusters are composed of multiple standalone machines connected by a network. While machines in a cluster do not have to be symmetric, load balancing is more difficult if they are not. The most common type of cluster is the Beowulf cluster, which is a cluster implemented on multiple identical commercial off-the-shelf computers connected with a TCP/IP Ethernet local area network. The vast majority of the TOP500 supercomputers are clusters.

Massive parallel processing: 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. In a 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. Example: Blue Gene/L.

Grid computing: Grid computing is the most distributed form of parallel computing. It makes use of computers communicating over the 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 problems. Most grid computing applications use middleware, software that sits between the operating system and the application to manage network resources and standardize the software interface.

Vector processors: 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.

General-purpose computing on graphics processing units (GPGPU): General-purpose computing on graphics processing units (GPGPU) is a fairly recent trend in computer engineering research. GPUs are co-processors that have been heavily optimized for computer graphics processing. Computer graphics processing is a field dominated by data parallel operations - particularly linear algebra matrix 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 and AMD releasing programming environments with CUDA and Stream SDK respectively. The technology consortium Khronos Group has released the OpenCL specification, which is a framework for writing programs that execute across platforms consisting of CPUs and GPUs. AMD, Apple, Intel, Nvidia and others are supporting OpenCL.

1.6. Parallel Programming Language

A concurrent language is defined as one which uses the concept of simultaneously executing processes or threads of execution as a means of structuring a program. A parallel language is able to express programs that are executable on more than one processor. Some of these are described as here:

OpenMP: OpenMP (Open Multi-Processing) is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor architectures and operating systems, including Solaris, AIX, HP-UX, Linux, Mac OS X, and Windows platforms. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.

OpenCL: Open Computing Language (OpenCL) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors. OpenCL specifies a language (based on C99) for programming these devices and application programming interfaces (APIs) to control the platform and execute programs on the compute devices. OpenCL provides parallel computing using task-based and data-based parallelism. OpenCL is an open standard maintained by thenon-profit technology consortium Khronos Group. Conformant implementations are available from Altera, AMD, Apple, ARM Holdings, Creative Technology, Imagination Technologies, Intel, Nvidia, Qualcomm, Samsung, Vivante, Xilinx, and ZiiLABS.

POSIX Thread API: POSIX is an acronym for Portable Operating System Interface. The IEEE specifies a standard 1003.1c-1995, POSIX API. Also referred as Pthreads. POSIX has emerged as the standard API, supported by most vendors.

CUDA: CUDA, which stands for Compute Unified Device Architecture, is a parallel computing platform and application programming interface (API) model created by NVIDIA. It allows software developers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach known as GPGPU. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements.

The CUDA platform is designed to work with programming languages such as C, C++ and FORTRAN. This accessibility makes it easier for specialists in parallel programming to utilize GPU

resources, as opposed to previous API solutions like Direct3D and OpenGL, which required advanced skills in graphics programming. Also, CUDA supports programming frameworks such as OpenACC and OpenCL.

References

[1] "Intel Launches Devil's Canyon and Overclockable Pentium: i7-4790K, i5-4690K and G3258". Anandtech. 3 June 2014. Retrieved 29 June 2014.
[2] http://en.wikipedia.org/wiki/Parallel_computing