Journal Of Harmonized Research (JOHR)

Journal Of Harmonized Research in Engineering
1(2), 2013, 65-72

ISSN 2347 – 7393

Original Research Article

FUNCTIONING EFFICIENTLY USING PARALLEL COMPUTING

Nikita Chhillar, Nisha Yadav, Neha Jaiswal

Department of Computer Science and Engineering,
Dronacharya College of Engineering, Khentawas,
Farukhnagar, Gurgaon, India

Abstract:
Parallel Computing is very wide-ranging and large topic. It basically means use of multiple processors or computers working together on a common task. Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem. This paper consists of a conversation on parallel computing - what it is and where it's used, followed by a discussion on concepts connected with parallel computing. The subject dependencies, Flynn's taxonomy are then explored. These topics are followed by types of parallelism hardware parallelism and software parallelism.

Keywords: Parallel computing, computation, parallelism.

Introduction:

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"). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. 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 multicore processors.

For Correspondence:
nikitachhillarATyahoo.com
Received on: September 2013
Accepted after revision: October 2013
Downloaded from: www.johronline.com

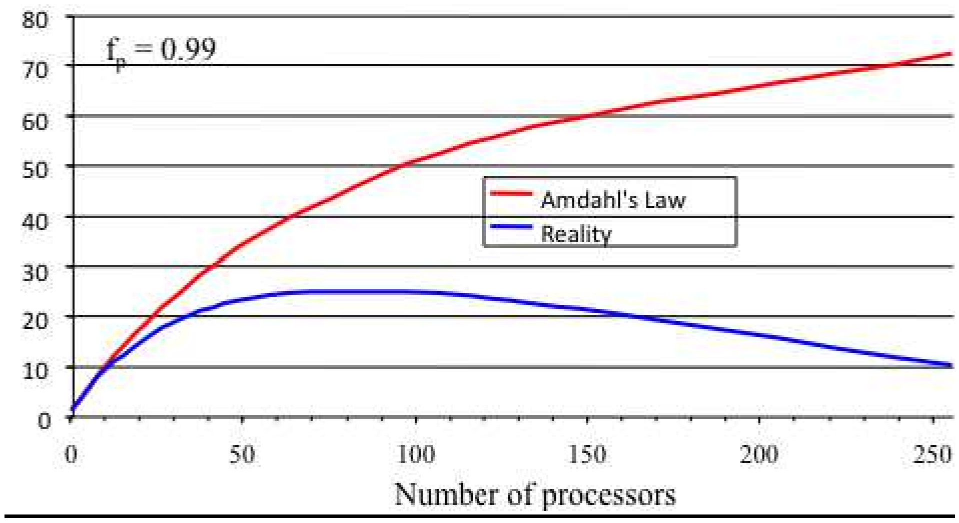
The maximum possible speed-up of a program as a result of parallelization is known as Amdahl's law.

- **Gustafson's Law:**

•Effect of multiple processors on run time of a problem with a *fixed amount of parallel work per processor.*

- **Amdahl's law:**

•All parallel programs contain: parallel sections & serial sections
•Serial sections limit the parallel effectiveness
•Amdahl's Law states this formally
–Effect of multiple processors on speed up



Uses for Parallel Computing:
Science and Engineering:
Historically, parallel computing has been considered to be "the high end of computing", and has been used to model difficult problems in many areas of science and engineering:

For eg: Atmosphere, Earth, Environment ,Physics, condensed matter, high pressure, photonics, Biotechnology, Genetic , Chemistry, Molecular Sciences ,Geology, Seismology , Engineering, Circuit Design, Microelectronics ,Defense, Weapons, Computer Science, Mathematics.

Jet Construction

Industrial and Commercial:
Today, commercial applications provide an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways.
For example: Databases, data mining , Oil exploration , web based business, Medical imaging and diagnosis, Pharmaceutical design

,Financial and economic modeling Management, Advanced graphics and virtual reality, particularly in the entertainment industry, multi-media technologies Collaborative work environments

Flynn's 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.

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 applications. Multiple-instruction-single-data (MISD) is a rarely used classification. While computer architectures to deal with this were devised (such as systolic arrays), few applications that fit this class materialized. Multiple-instruction-multiple-data (MIMD) programs are by far the most common type of parallel programs.

Flynn's taxonomy

S I S D **Single Instruction Stream** **Single Data Stream**	**S I M D** **Single Instruction Stream** **Multiple Data Stream**
M I S D **Multiple Instruction Stream** **Single Data Stream**	**M I M D** **Multiple Instruction Stream** **Multiple Data Stream**

❖ **Single Instruction, Single Data (SISD):**

- A serial (non-parallel) computer
- **Single Instruction:** Only one instruction stream is being acted on by the CPU during any one clock cycle
- **Single Data:** Only one data stream is being used as input during any one clock cycle
- Examples: older generation mainframe, minicomputer and workstations; most modern day PCs.

❖ **Single Instruction, Multiple Data (SIMD):**

- A type of parallel computer
- **Single Instruction:** All processing units execute the same instruction at any given clock cycle
- **Multiple Data:** Each processing unit can operate on a different data element and Vector Pipelines

Examples:

- o Processor Arrays: Connection Machine CM-2, MasPar MP-1 & MP-2, ILLIAC IV
- o Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10

❖ **Multiple Instruction, Single Data (MISD):**

- A type of parallel computer
- **Multiple Instructions:** Each processing unit operates on the data independently via separate instruction streams.
- **Single Data:** A single data stream is fed into multiple processing units.

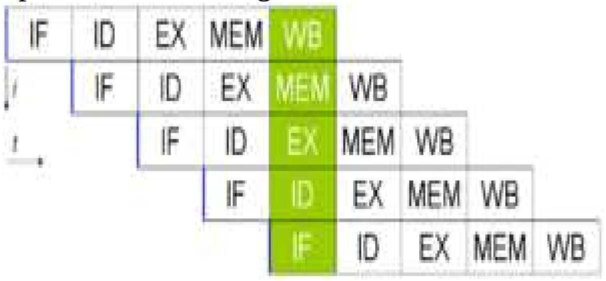
❖ **Multiple Instructions, Multiple Data (MIMD):**

- A type of parallel computer
- **Multiple Instruction:** Every processor may be executing a different instruction stream
- **Multiple Data:** Every processor may be working with a different data stream
  Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.

Types of parallelism:

✓ **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—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.

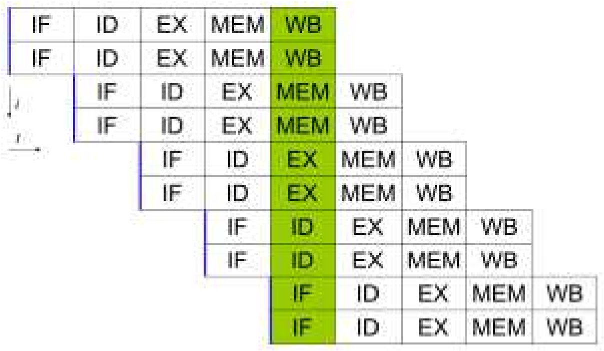


✓ **Instruction-level parallelism:**

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

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.

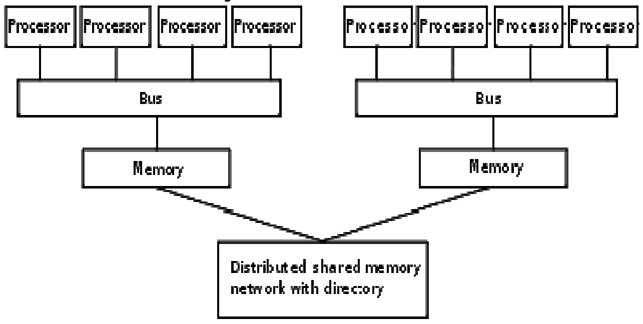


A five-stage pipelined superscalar processor, capable of issuing two instructions per cycle. It can have two instructions in each stage of the pipeline, for a total of up to 10 instructions (shown in green) being simultaneously executed.

Hardware:

Memory and communication:

Main memory in a parallel computer is either shared memory (shared between all processing elements in a single address space), or distributed memory (in which each processing element has its own local address space). 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 and 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.



A logical view of a 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 systems make use of caches—small, 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 system, which keeps track of cached values and strategically purges them, thus ensuring correct program execution. Bus snooping 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.

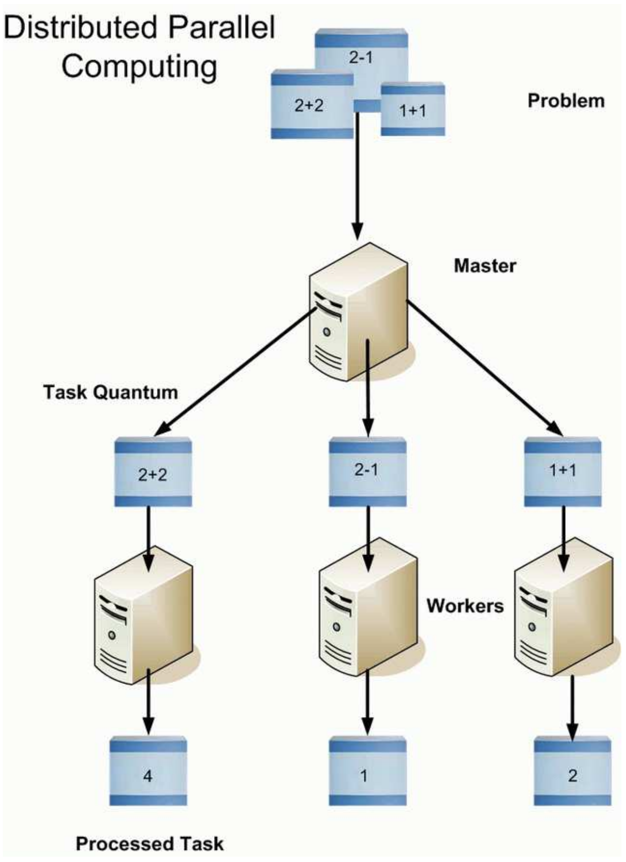
Processor–processor and processor–memory communication can be implemented in hardware in several ways, including via shared (either multiported or multiplexed) memory, a crossbar switch, a shared bus or an interconnect network of a myriad of topologies including star, ring, tree, hypercube, fat hypercube (a hypercube with more than one processor at a node), or n-dimensional mesh.

1) Classes of parallel computers:

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.

➢ ***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:*

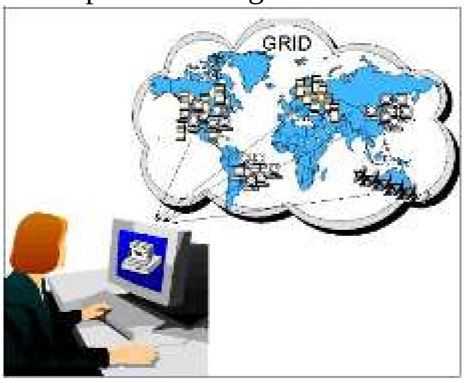
A cluster is a group of loosely coupled computers that work together closely, so that in some respects they can be regarded as a single computer. Clusters are composed of multiple standalone machines connected by a network. While machines in a cluster do not have to be symmetric, load balancing is more 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.

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."

➢ *Grid computing:*

Distributed 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. Many distributed computing applications have been created, of which SETI@home and Folding@home are the best-known examples. 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. The most common distributed computing middleware is the 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:

Within parallel computing, there are specialized parallel devices that remain niche areas of interest. While not domain-specific, they tend to be applicable to only a few classes of parallel problems.

- *Reconfigurable computing with field-programmable gate arrays:*

Reconfigurable computing is the use of a 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 such as VHDL or Verilog. However, programming in these languages can be tedious. Several vendors have created

C to HDL languages that attempt to emulate the syntax and/or semantics of the C programming language

- *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.

- *Application-specific integrated circuits:*

Several application-specific integrated circuit (ASIC) approaches have been devised for dealing with parallel applications. As a result, for a given application, an ASIC tends to outperform a general-purpose computer. This process requires a mask, which can be extremely expensive. (The smaller the transistors required for the chip, the more expensive the mask will be.) Meanwhile, performance increases in general-purpose computing over time tend to wipe out these gains in only one or two chip generations. High initial cost and the tendency to be overtaken have rendered ASICs unfeasible for most parallel computing applications.

- *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. An example vector operation is $A = B \times C$, where A, B, and C are each 64-element vectors of 64-bit floating-point numbers." They are closely related to Flynn's SIMD classification.

Software:

❖ **Parallel programming languages:**

Concurrent programming languages, libraries, APIs, and parallel programming models 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. POSIX Threads and OpenMP are two of most widely used shared memory APIs, whereas Message Passing Interface (MPI) is the most widely used message-passing system API. HMPP (Hybrid Multicore Parallel Programming) directives an Open Standard denoted Open HMPP. The Open HMPP directive-based programming model offers syntax to efficiently offload computations on hardware accelerators and to optimize data movement to/from the hardware memory. Open HMPP directives describe remote procedure call (RPC) on an accelerator device (e.g. GPU) or more generally a set of cores.

❖ **Automatic parallelization:**

Automatic parallelization of a sequential program by a compiler is the holy grail of parallel computing. Despite decades of work by compiler researchers, automatic parallelization has had only limited success Mainstream parallel programming languages remain either explicitly parallel or (at best) partially implicit, in which a programmer gives the compiler directives for parallelization. A few fully implicit parallel programming languages exist—SISAL, Parallel Haskell, System C (for FPGAs), Mitrion-C, VHDL, and Verilog.

❖ **Application check pointing:**

As a computer system grows in complexity, the mean time between failures usually decreases. Application check pointing 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; this information can be used to restore the program if the computer should fail. Application check pointing means that the program has to restart from only its last checkpoint rather than the beginning. While check pointing 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.

Conclusion:

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. **Save time and/or money:** throwing more resources at a task will shorten it's time to completion, with potential cost savings. Parallel clusters can be built from cheap, commodity components. **Provide concurrency:** A single compute resource can only do one thing at a time. Multiple computing resources can be doing many things simultaneously

References:

1. Bernstein, A. J. (1 October 1966). "Analysis of Programs for Parallel Processing". IEEE Transactions on Electronic Computers. EC-15 (5): 757–763. Doi:10.1109/PGEC.1966.264565.
2. Gottlieb, Allan; Almasi, George S. (1989). Highly parallel computing. Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1.
3. S.V. Adve et al. (November 2008). "Parallel Computing Research at Illinois: The UPCRC Agenda" (PDF).
4. Asanovic, Krste et al. (December 18, 2006). "The Landscape of Parallel Computing Research: A View from Berkeley" (PDF).
5. Hennessy, John L.; Patterson, David A., Larus, James R. (1999). Computer organization and design: the hardware/software interface (2. ed., 3rd print. ed.). San Francisco: Kaufmann. ISBN 1-55860-428-6
6. Jump up to: **a** **b** Barney, Blaise. "Introduction to Parallel Computing". Lawrence Livermore National Laboratory. Retrieved 2007-11-09
7. Singh, David Culler ; J.P. (1997). Parallel computer architecture ([Nachdr.] ed.). San Francisco: Morgan Kaufmann Publ. p. 15. ISBN 1-55860-343-3.
8. Patterson and Hennessy Gustafson, John L. (May 1988). "Reevaluating Amdahl's law". Communications of the ACM **31** (5): 532–533. doi:10.1145/42411.42415
9. Encyclopedia of Parallel Computing, Volume 4 by David Padua 2011 ISBN 0387097651
10. Asanovic, Krste, et al. (December 18, 2006). The Landscape of Parallel Computing Research: A View from Berkeley (PDF)
11. Anthes, Gry (November 19, 2001). "The Power of Parallelism". Computerworld. Retrieved 2008-01-08.

.