Middle-East Journal of Scientific Research
24 (Recent Innovations in Engineering, Technology, Management & Applications): 179-185, 2016
ISSN 1990-9233; © IDOSI Publications, 2016
DOI: 10.5829/idosi.mejsr.2016.24.RIETMA129

Review on Parallel Processing Hardware and Software

[1]S. Vijaya Kumar and [2]Dr. Vidyaa ThulasiRaman

[1]Department of Computer Science,
Periyar University, Salem, TN, India
[2]Department of Computer Science,
Government Arts and Science College (W), Bargur, Tamilnadu, India

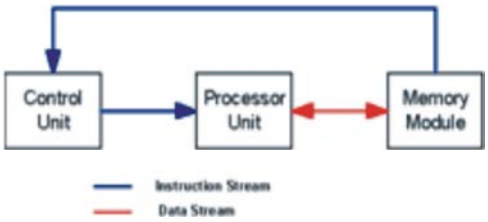
Abstract: In this paper a review of parallel processing hardware's and software's were carried out that represents all the aspects of parallel computing system. Parallel processing uses multiple processors to accomplish a single task which can be divided into independent subtasks. Parallel computers consist of multiple processors, a parallel computing architecture and a software's support for parallel programming. All these hardware's and software's support the efficient use of multiple processors. This paper explains all the aspects of parallel processing and the areas in which parallel processing can be applied.

Key words: Parallel computing · Parallel computing hardware · Multicore processor · GPU · Clusters · Parallel computing software

INTRODUCTION

There are two primary reasons for using parallel computing:

- To minimize the execution time
- To Solve larger problems



Definition: Parallel computing is the act of solving a problem of size n by dividing its domain into k≥2 (with k∈N) parts and solving them with p physical processors, simultaneously [1].

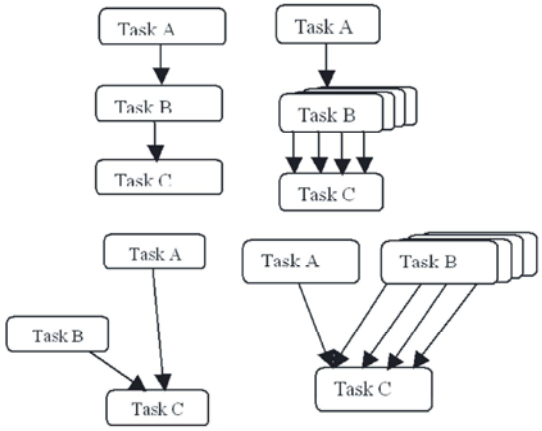
A processor has its maximum limits in its processing speed. This limitation can be overcome by increasing the number of processors connected with each other to share the data and message passing with each other to solve a grand problem [2]. Parallelism can be achieved if more than one processor simultaneously executes the independent segments of a problem. A large problem can be divided into multiple independent tasks of nearly same size by applying an appropriate task partitioning technique and each of the tasks will be executed on different processors simultaneously. Assigning the tasks to the processors is not just the solution of the problem in parallel. Some faster processor may sit idle while a slower processor may busy in a parallel computing system resulting slower computing speed due to the uneven distribution of tasks. Parallel computation can be used for the efficient use of processors with hardware and software support.

Sequential vs Parallel Algorithm: Sequential algorithm consists of different steps in which every subsequent step is depend on the current step. But in the parallel algorithm none other steps is dependent on another step. In designing parallel algorithms one has to consider splitting up of problem into independent subtask. The two categories of splitting a problem can be Data parallelism and Task Parallelise

Data Parallelism: Data parallelism [1] can be accomplished with algorithms that allow its data set to easily be divided into a subset of non-contiguous,

Corresponding Author: S. Vijaya Kumar, PhD Research Scholar, Department of Computer Science, Periyar University, Salem, TN, India.

discrete values that can be processed independently of each other. The results of one subset can be combined with another to get a final result.



a) Original Algorithm b) Data Parallelism c) Task Parallelism d) Data and Task Parallelism

Fig. 2.1: Task and Data Parallelism

Task Parallelism: Task parallelism [1] is accomplished with algorithms that allow splitting problem into independent subtask and each subtask can be processed independently of each other with a separate processer or thread. It can be executed in parallel without interfering with any other thread.

Data and Task Parallelism: It is an hybrid of Data parallelism and Task Parallelism in which partition of data as well as problem can be divided independently. Figure 2.1 shows the difference between the each parallelism.

Parallel Computing Hardware Architecture: Michael J. Flynn proposed [3] a classifications of CPU architectures. There were four classifications, three of which enables parallel processing.

Flynn's Taxonomy

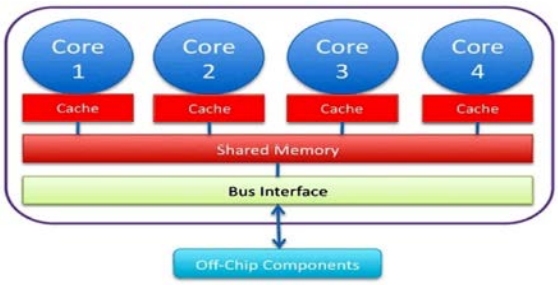


Fig. 2.2: SISD Architecture

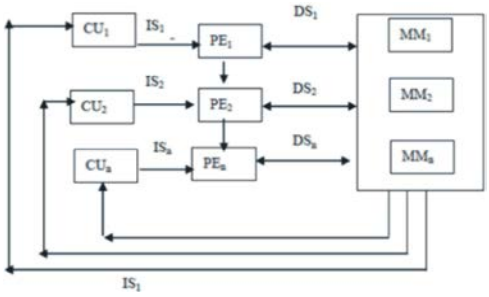


Fig. 2.3: SIMD Architecture

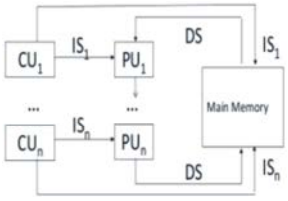


Fig. 2.4: MISD Architecture

Single Instruction-Single Data (SISD): A SISD architecture shown in Figure 2.2 is a scalar unicore processor executing oneinstruction on a register with only one data element enabling no parallelism at all.

Single Instruction-Multiple Data (SIMD): A SIMD architecture shown in Figure 2.3 enable execution of one instruction on several data elements. This can be achieved with only one core working on a register packed with multiple values, e.g., the SSE extension to Intel's x86 architecture.

Multiple Instructions-Single Data (MISD): A MISD architecture shown in Figure 2.4 do different instructions on one data element. This architecture is very uncommon, but a multicore processor can mimic such behavior.

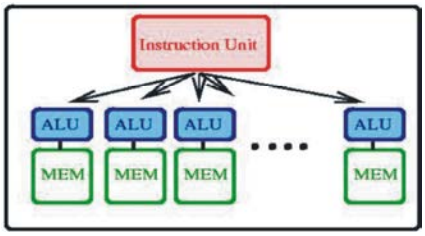


Fig. 2.5: MIMD Architecture

Multiple Instructions–Multiple Data (MIMD): A MIMD architecture shown in Figure 2.5 is able to execute multiple (different) instructions on multiple data elements. A multicore processor is a true MIMD architecture.

Multicore Processors: One of the first multicore processor for general-purpose use was IBM's POWER4. It was released in October, 2001. One configuration of this processor had two cores, but it also came in a second configuration with four cores. In 2005, Intel and AMD followed with their first multicore processors. Intel's Pentium D was released in April and the Athlon 64 X2 in May. Both the Pentium D and Athlon 64 X2 were dual core processors. This was the start of new era in processor development and most new architectures are symmetric and homogenous, where every processing core is equal and has the exact same properties. Different cores in symmetric multicore processors also usually have equal access to the processors memory space. This means that two threads that share paged virtual memory can access the same memory positions. This is analogous the shared data elements or resources. When these two threads execute in parallel on two different cores and shares a memory positions, some of these addresses might be cached for faster access. As some processor caches are functional units local to each core, it means that a processor altering some memory only does so in their own local cache and thus outdating the value in the other core's cache. To ensure that this does not happen, cache coherence protocols have been developed. In April 2010, Intel announced the SCC (Single-Chip Cloud Computer). The SCC is a 48-core processor, but Intel states the chip and its technology is able to scale up beyond 100 cores.

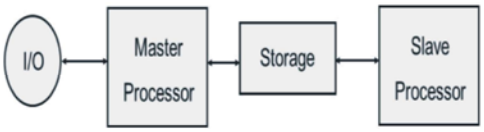


Fig. 2.6: Multicore Processor

GPU: Graphic Processing Units: GPUs [1] have become an important tool for offloading the CPU in processing graphics. Graphic algorithms are often embarrassingly parallel and are able to scale with a great number of threads executed in parallel and because of this, GPUs often have many parallel cores. Because the purpose of the GPU is to process graphics, the GPU has a special-purpose design in contrast to the CPU's general purpose design. GPUs are used for producing multimedia content and support the commonly used data types in such algorithms. GPUs have also been proven to be effective at solving other task and because of this, GPGPU (General-Purpose computation on Graphics Processing Units) have become popular in several computational fields, e.g., in scientific computing. Even though some new CPUs and System-on-Chip (SoC) solutions have a GPU on die, GPUs are often mounted on a discrete card connected to the CPU, for example via a PCIExpress bus. Utilizing GPUs to solve general data parallel problems can give great performance gains, but programming GPUs efficiently requires in depth knowledge of the specific architecture. There a two prominent ways of programming GPUs, Nvidia's propriertary approach with CUDA and the open and unified approach using OpenCL.

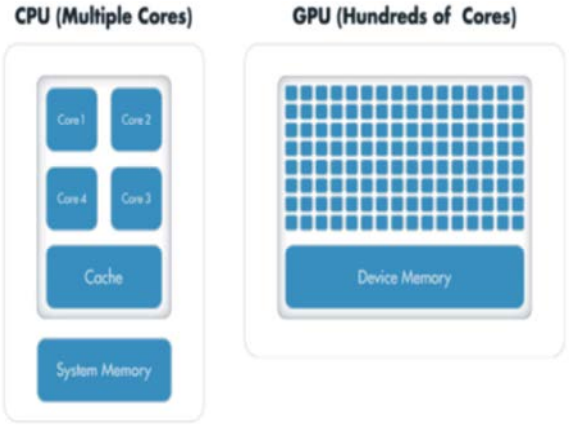


Fig. 2.7: GPU with Hundreds of Cores

Asymmetric Processors: Asymmetric processors have a heterogeneous processing cores implemented on one discrete chip. An example of an asymmetric processor architecture is the Cell Broadband Engine Architecture (CellBEA) from Sony Computer Entertainment, Toshiba and IBM. A Cell CPU includes one or more PowerPC Processor Elements (PPE) and one or more Synergistic Processor Elements (SPE). In case of the Cell BEA, the PPE is a 64-bit processor with a Power ISA for general purpose processing and for managing the system and SPEs. An SPE includes a Synergistic Processor Unit (SPU), which has its own ISA with special-purpose vector registers and instructions that are able to do SIMD operations. The SPE has 128 registers which all are 128 bits, this enables operations to be executed on sixteen 8-bit, eight 16bit, four 32-bit and two 64-bit values. The SPU has no access to the main memory or to PPE CPU features and as mentioned, data transfers must be explicitly expressed by the developer. The most famous implementation of the Cell Broadband Engine Architecture is perhaps the processor shipped with Sony's PlayStation 3, usually referred to as the Cell Broadband Engine (Cell BE). CPUs with asymmetric cores that do specialized

operations have been more frequent in embedded applications, like DSPs in Texas Instruments' OMAP3 series or the mentioned SIMD processors in game consoles, but AMD's Fusion and Intel's Sandy Bridge now bring asymmetric architectures to laptops and desktop computers.

Grid Computing Environment: The grid computing [4] is basically a way to execute jobs across a distributed set of processors. Grid computing offers sharing of resources over geographically distributed locations. Furthermore, collaborative nature of grids leads to the concept of virtual organizations consisting of a dynamic set of data and resources to solve a specific task. The architecture of grid computing is shown in Figure 2.9. Grid computing divides a large program into sub-programs and assigns each sub-program to an individual processor. Each processor would now process the sub-program and would return the end result. Even if one processor fails, the result would not get affected because the task will be reassigned to other processor. A variety of resources may be shared, including computers, storage devices, sensors, scientific instruments, network, software or data. In Grid, every processing node is autonomous i.e. it has its own administration and control and behaves like an independent entity. Grid Computing Grids can further be categorized in two; Computational grids-these are grids that primarily focus on intensive and complex computations, data grids-grids for management and control of sharing of vast amounts of data. Distributed or grid computing, traditionally is a special type of parallel computing relying on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet. This is in contrast to traditional notion of a supercomputer that has many processors connected by a local high-speed computer bus.

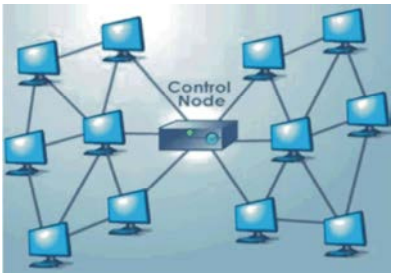


Fig. 2.8: Grid Computing

Cluster Computing Environment: Cluster computing [4] is a platform that combines commercial off-the-shelf computers with a high speed network to form a single powerful super computer. The use of clusters as computing platform is not just limited to scientific and engineering applications; there are many business applications that get benefit from the use of clusters. Basically, a cluster is a type of parallel or distributed computer system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource. When several computers are connected to a high speed network they consume a single resource pool and these grouped computers are called clusters. Cluster Computing Computers in a cluster network are attached in tightly coupled fashion i.e. they all use the same subnet and follow the same domain and networked with very high bandwidth connection and they all use the same hardware and software. In cluster computing environment users request is distributed among various computers in a network. The advantage of doing this is to increase the processing speed.

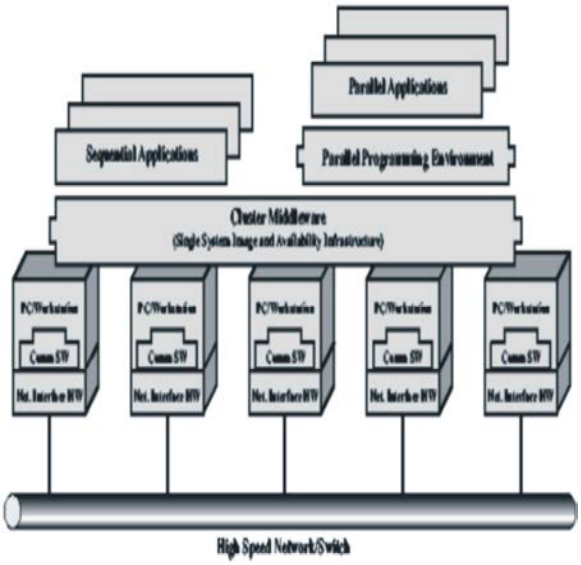


Fig. 2.9: Cluster Computer

Software Support for Parallel Programming: Designing a parallel programming is always challenging matter. More and more focus is imposed on designing parallel programming design. Two methodologies are widely used for the purpose of parallel programs [3]. They are auto-parallelization compiler and library based software. Auto parallelization does its work in two fashions.

First one is complete automatic compiler which finds the parallelism during the compilation of source code. This approach mainly aims to parallelize the loops like for and do. Second one is program directed which uses compiler directives to make the code parallel. The library based parallel software embedded its library to the sequential programming languages to support the parallel execution of a problem. MPI and OpenMP are the most widely acceptable standard for parallel programming. MPI is a common message passing library which has the primitive function like send() /receive() by which MPI process communicate with the other process through message passing. OpenMP is a parallel framework for supporting compiler directives, library support routine. Apart form this LINDA, CPS, P4 etc are the example of parallel programming software.

Currently, there are several relatively popular and sometimes developmental, parallel programming implementations based on the Data Parallel / PGAS model.

Coarray Fortran: A small set of extensions to Fortran 95 [5] for SPMD parallel programming. Compiler dependent. Coarrays are an exciting parallel programming extension for Fortran. They are special variables that can be shared across multiple instances of the same program, which are called 'images'. The main advantage of Coarrays is the high level of integration with the Fortran language itself, making programs vastly more readable than subroutine calls to parallel libraries. Synchronization primitives are also provided.

Unified Parallel C (UPC): An extension to the C programming language for SPMD parallel programming. Compiler dependent. Unified Parallel C (UPC) [6] is an extension of the C programming language designed for high performance computing on large-scale parallel machines. The language provides a uniform programming model for both shared and distributed memory hardware. The programmer is presented with a single shared, partitioned address space, where variables may be directly read and written by any processor, but each variable is physically associated with a single processor. UPC uses a Single Program Multiple Data (SPMD) model of computation in which the amount of parallelism is fixed at program startup time, typically with a single thread of execution per processor.

Global Arrays: Provides a shared memory style programming environment in the context of distributed array data structures. Public domain library with C and Fortran77 bindings. Global Arrays (GA) [7] is a Partitioned Global Address Space (PGAS) programming model. It provides primitives for one-sided communication (Get, Put, Accumulate) and Atomic Operations (read increment). It supports blocking and non-blocking primtives and supports location consistency. Starting GA v5.4, ARMCI has been deprecated and replaced with Communication runtime for Extreme Scale (ComEx). GA uses ComEx to abstract inter-node communication operations. The default ComEx ports use MPI --- which makes GA and ComEx portable for high-end systems. Most ComEx implementations use on-node shared memory for faster intra-node communication.

X10: A PGAS based parallel programming language being developed by IBM at the Thomas J. Watson Research Center [8]. X10 is a modern language in the strongly typed, object-oriented programming tradition. Its design draws on the experience of team members with foundational models of concurrency, programming language design and semantics, type systems, static analysis, compilers, runtime systems, virtual machines. Our goals were simple-design a language that fundamentally focuses on concurrency and distribution and is capable of running with good performance at scale, while building on the established productivity of object-oriented languages. In this, we sought to span two distinct programming language traditions - the old tradition of statically linked, ahead-of-time compiled languages such as Fortran, C, C++ and the more modern dynamically linked, VM based languages such as Java, C#, F#. X10 supports both compilation to the JVM and, separately, compilation to native code. It runs on the PERCS machine (Power 7 CPUs, P7IH interconnect), on Blue Gene machines, on clusters of commodity nodes, on laptops; on AIX, Linux, MacOS; on PAMI and MPI; on Ethernet and Infiniband.

Chapel: Chapel is an open source parallel programming language project being led by Cray. Chapel is an emerging programming language designed for productive parallel computing at scale [9]. Chapel's design and implementation have been undertaken with portability in mind, permitting Chapel to run on multicore desktops and laptops, commodity clusters and the cloud.

Matlab: Using Parallel Computing Toolbox™ we can solve computationally and data-intensive problems using multicore processors, GPUs and computer clusters [10]. High-level construct parallel for-loops, special array

types and parallelized numerical algorithms we can parallelize MATLAB® applications without CUDA or MPI programming. We can use the toolbox with Simulink® to run multiple simulations of a model in parallel. The toolbox can be used as full processing power of multicore desktops by executing applications on workers (MATLAB computational engines) that run locally. Without changing the code, we can run the same applications on a computer cluster or a grid computing service (using MATLAB Distributed Computing Server™). We can run parallel applications interactively or in batch.

Key Features:

- Parallel for-loops (parfor) for running task-parallel algorithms on multiple processors
- Support for CUDA-enabled NVIDIA GPUs
- Full use of multicore processors on the desktop via workers that run locally
- Computer cluster and grid support (with MATLAB Distributed Computing Server)
- Interactive and batch execution of parallel applications
- Distributed arrays and spmd (single-program-multiple-data) for large dataset handling and data-parallel algorithms

Cuda: CUDA® is a parallel computing platform and programming model invented by NVIDIA [11]. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit. GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems. Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs.

Hadoop: Hadoop, an open source project from The Apache Software Foundation, emerged from the needs of companies such as Google, Yahoo, AOL and Facebook. Hadoop in a cloud enables parallel processing spread across these many servers, speeding job completion [12]. Hadoop can seriously boost performance in data search and processing scenarios, such as retail chain data mining that seeks trends across millions of individual retail store purchases, or security information that intelligence agencies collect from a wide variety of sources to detect terrorist activity patterns.

CONCLUSION

In this paper we have presented the parallel processing hardware and software environment. Parallel computing is a broad topic which as various hardware and software support. The goal of Grids and clusters in hardware design is resource sharing, reduction in tasks execution time and allowance of simultaneous access of a single resource. The performance measure of the parallel computing can be measured as

Performance can be measured as speed-up (S):

Running Time of Sequential Algorithm

S = (Running Time of Sequential Algorithm) / (Running Time of Parallel Algorithm)

The advantage of parallel processing is depends on the CPU-bound. That is if majority of task depends only on CPU, not on the I/O cycles.

REFERENCES

1. Cristobal, A. Navarro, Nancy Hitschfeld-Kahler and Luis Mateu, 2014. “A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures” Commun. Comput. Phys. doi: 10.4208/cicp.110113.010813a, 15(2): 285-329 February 2014.
2. Rafiqul Zaman Khan and Md Firoj Ali, 2012. “Current Trends in Parallel Computing “International Journal of Computer Applications (0975-8887) 59(2), December 2012.
3. Gurinder Singh and Aman Kaura, 2016. “Recent Trends in Parallel Computing” Gian Jyoti E-journal, 6(1) (Jan-Apr 2016) ISSN 2250-348X 12th National Conference on ‘Digital India: Key Essentials To Drive Vision into Reality’ Saturday, 19th December, 2015 at GJIMT, Sector-54, Mohali-160055, Punjab, India.
4. Suri, P.K. and Sumit Mittal, 2012. “A Comparative Study of various Computing Processing Environments: A Review” International Journal of Computer Science and Information Technologies, 3(5): 5215-5218.
5. http://www.g95.org/coarray.shtml#overview
6. https://upc-lang.org
7. http://hpc.pnl.gov/globalarrays

8. http://researcher.ibm.com/researcher/view_group.php?id=536
9. chapel.cray.com/overview.html
10. http://in.mathworks.com/products/parallel-computing
11. http://www.nvidia.com/object/cuda_home_new.html
12. http://hadoop.apache.org
00. Felician ALECU., 2009. "The Impact of Parallel Processing on Operating Systems" Oeconomics of Knowledge, 1(1), 3Q 2009.