ADVANCES AT A GLANCE IN PARALLEL COMPUTING

RAJKUMAR SHARMA
Computer Centre, Vikram University
Ujjain (MP) INDIA
rksujn@rediffmail.com

Abstract

In the history of computational world, sequential uni-processor computers have been exploited for years to solve scientific and business problems. To satisfy the demand of compute & data hungry applications, it was observed that better response time can be achieved only through parallelism. Large computational problems were partitioned and solved by using multiple CPUs in parallel. Computing performance was further improved by adopting multi-core architecture which provides hardware parallelism through use of multiple cores. Efficient resource utilization of a parallel computing environment by using software and hardware parallelism is a major research challenge. The present hardware technologies provide freedom to algorithm developers for control & management of resources through software codes, such as threads-to-cores mapping in recent multi-core processors. In this paper, a survey is presented since beginning of parallel computing up to the use of present state-of-art multi-core processors.

1. Introduction

Applying more than on processors to a single computational problem gave rise to *Parallel and Distributed Computing* which opened up the doors for many computing possibilities for researchers and High Performance Computing (HPC) community. In the journey of Parallel and Distributed Computing, several milestones have been achieved and deployed using contemporary technologies like Dedicated Parallel Machines, SMP clusters, Super Computing, Network of Workstations, Commodity Clusters etc. to today's Grid and Cloud Computing. Advanced developments in network technologies and availability of cheap but sophisticated processors also boosted the use of scalable distributed computing as a powerful computational to exploit scattered resources. Algorithm developers can now control and manage hardware resources through software codes, such as threads-to-cores mapping in recent multi-core processors. Since the introduction of multi-core processors, the way of computing has been changed, as the processor itself provides a built-in parallelism by the cores. The exploitation of multi-core processors is more complex than uni-core processors.

Today's hardware design is based on complex algorithms. Incorporation of underlying hardware information is necessary in computational problem's algorithm design to produce efficient computing results and better resource utilization. This approach, however, is machine dependent and imposes an extra burden to algorithm developer to have deep knowledge of underlying hardware. There exist trade-off between execution performance and parallel architecture abstraction to developers. In this paper, we address important research issues related to parallel computing which include parallel computing architectures, parallel algorithms, parallel programming languages, parallel programming tools & environments etc.

2. Parallel Computing Architectures

Uni-processor sequential computers were not able to produce results of large computational problems within desirable time limit. Execution performance was improved by using more than one processor in parallel. An application can be executed on such an interconnected multiprocessor in a variety of combinations of 'instruction' and 'data'. A well known classification of parallel architecture by Flynn is – Single Instruction Stream, Single Data Stream (SISD), Single Instruction Stream, Multiple Data Stream (SIMD), Multiple Instruction Stream, Single Data Stream (MISD) and Multiple Instruction Stream, Multiple Data Stream (MIMD). All types except MIMD, have given better performance in their contemporary time. But nowadays all popular distributed systems are based on MIMD architecture. The two categories of MIMD architecture are – *Shared Memory Systems (tightly coupled)* and *Distributed Memory Systems (loosely coupled)* as shown in Figure 1.1. To exploit these computing resources efficiently, *hybrid systems* are also in practice, which uses both shared as well as distributed memory for running a parallel application.

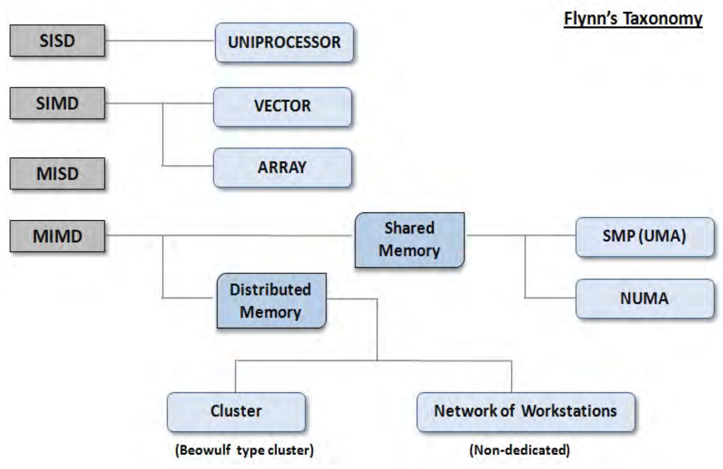

Fig 1.1 Flynn's Taxonomy for Parallel Architecture

In tightly coupled systems, a single system-wide primary memory (address space) is shared by all the processors whereas in loosely coupled systems, the processors do not share memory and each processor has its own local memory as shown in Figure 1.2. Usually, tightly coupled systems are referred to as ***Parallel Processing Systems*** and loosely coupled systems are referred to as ***Distributed Computing Systems*** [Sinha, 2002].

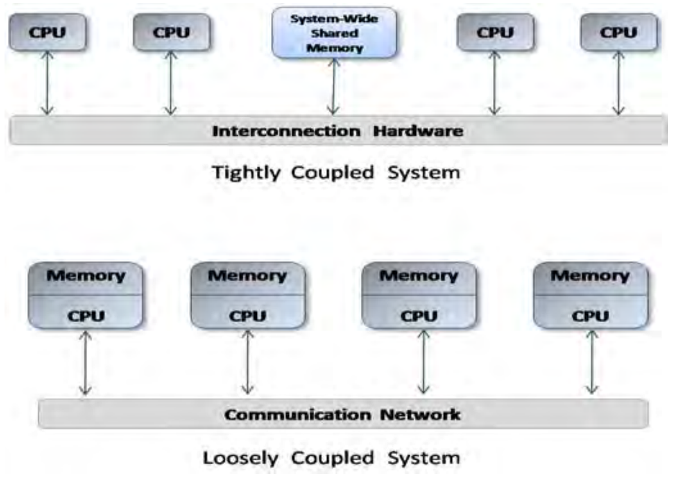

Fig. 1.2 Tightly Coupled and Loosely Coupled Systems

A decade ago, as the processors clock speed reached at 2.0 GHz to 3.0 GHz, it was projected by the processors' manufacturer that future processors will run at 10 GHz to 30 GHz and be capable of processing 1 trillion operations per second. This could not happen because as clock speed increased, the processors started consuming more power and emitting more heat. The CPUs speed could not accelerate without extra ordinary cooling and consequently hit a clock speed barrier [Sharma, 2011]. More heat dissipation was against energy efficient eco-friendly green computing. Therefore to further improve processors performance, multi-core architecture introduced to improve computing performance by providing hardware parallelism through more CPU cores, each having restrained clock speed and is known as *chip multiprocessing* (CMP). Prior to CMP, HPC community used *Symmetric Multi Processor* (SMP), but CMP proved to be faster than SMP as cores in CMP communicate with fast interconnect on same die whereas in SMP, processors communicate through motherboard. Launching of multi-core processors has been a breakthrough in *High Performance Computing* (HPC). Cluster of tiny multi-core processor nodes produces better results as compared to expensive SMP cluster and have attracted attention from HPC community due to their better performance-to-cost ratio for parallel processing applications. While more processor cores rendered effective execution results, multi-core technology inaugurated an extra layer of complexity for programming issues. To exploit each core in a multi-core environment, application software should be optimized by using multithreading. Multi-core processors can even degrade the performance for single threaded application due to reduction in clock speed. CPU cores and other computing resources of multi-core cluster should be utilized efficiently to provide optimal results for parallel applications.

Commodity cluster of multi-core processors is becoming more popular than traditional SMP cluster. Both scientific & business applications can be benefited from multi-core processors. Execution time can be minimized by running multiple threads on multiple cores. Multiple cores are effective for data parallel applications where same code can run through multiple threads on different sets of data as well as for functionally decomposed computation intensive tasks where each task run in parallel on different cores [Mamidala, 2010].

Computing performance can be improved by increasing number of physical processors. A system consists of more than one processors connected by an interconnection link, as shown in Figure 1.3, is *multiprocessor system*. A physical processor contains different resources including general purpose CPU registers, interrupt controller registers, execution units, caches, buses etc. [Roberts, 2006] Logical processors can be created which share common execution resources. This technique is known as simultaneous multi-threading (SMT). Implementation of SMT by Intel Inc. is known as *Hyper-Threading (HT) Technology* [INTEL]. In this technology, two threads are executed on a single core arranged in time slice manner or driven by interrupt mechanism. For the operating system perception, hyper threading is treated as separate cores. This allows application's multiple threads to schedule on logical processors or virtual cores as they would on multiprocessors systems, as shown in Figure 1.4.

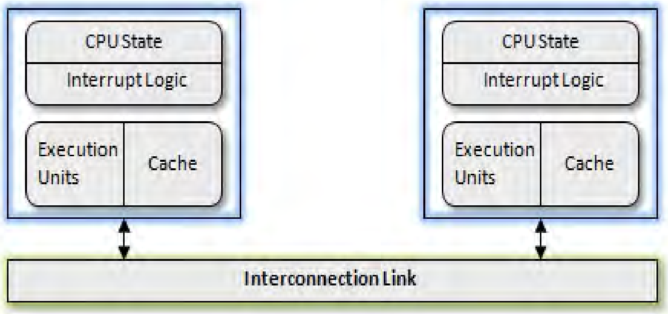

Fig. 1.3 Multiprocessors (SMP) System

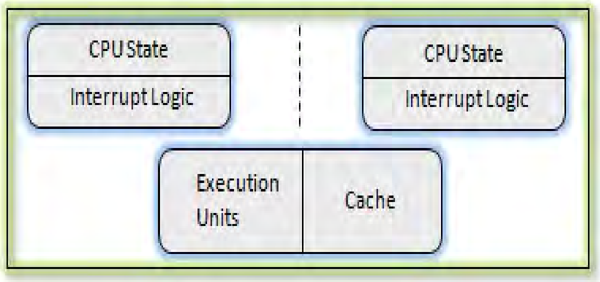

Fig. 1.4 Hyper-Threading Technology (Two Logical Cores)

In multi-core processors, there are two or more execution cores inside a single processors die. These cores have their own set of execution resources, as shown in Figure 1.5.

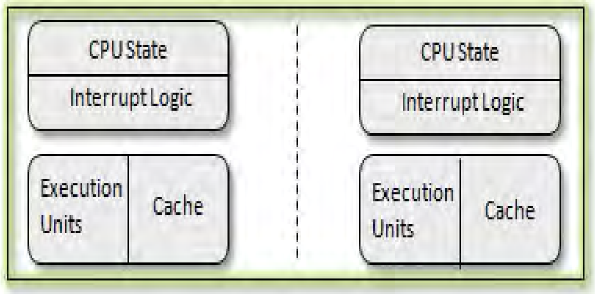

Fig. 1.5 Multi-core Architecture (Two Separate Physical Cores)

Dedicated cores in multi-core processors provide better performance than hyper threading. In the recent processor design, Intel Inc. combines hyper threading technique with multi-core processors, effectively increasing the number of logical processors by twice the number of executions cores.

3. Parallel Algorithms

3.1. *Amdahl's Law*

In parallel computing, a large problem is divided into smaller sub-tasks and these sub-tasks are carried out in parallel on several processors. Speedup refers to how much a parallel algorithm is faster than a corresponding sequential algorithm. In fact, a large mathematical or engineering problem may consists of several sequential and parallelizable portions. The speedup achieved in computing performance after parallelizing most of the problems is not ideal or linear speedup. According to ***Amdahl's Law***, a sequential portion of the problem will limit the overall speedup achieved from parallelization. Moreover, depending on the nature of problem, after a certain point, increase in number of processors will not improve the computing performance.

3.2. *Fine-grained and Course-grained Problems*

The performance of a parallel algorithm largely depends on the nature of problem, whether the problem is fine-grained problem or course-grained problem. A parallel task can be divided into a number of smaller sub-tasks, that can be carried out in parallel on different processors. A computation problem is said to be fine-grained when sub-tasks are dependent on the results of other sub-tasks. Sub-tasks more frequently communicate with each other and a higher level of synchronization among processor is required to solve such problems. In a computation problem, when each sub-task is independent of all other tasks, than it is called a course- grained computation problem. Sub-tasks less frequently communicate with each other in this category. A third type also exists known as 'embarrassingly parallel' in which sub-tasks never communicate with each other.

3.3. *Design Approach*

One approach to develop a parallel algorithm of a computational problem is to convert its existing sequential algorithm by identifying inherent parallelism in the problem. In another approach, a new parallel algorithm is developed independent of any existing sequential algorithm. In terms of methods, several approach are used depending on the dividing criteria such as divide and conquer, partitioning, pipelining etc.

4. Parallel Programming Languages and Tools

Parallel programming languages and tools largely depend on the underlying architecture being used to run parallel application, viz., shared memory architecture and distributed memory architecture [Kasim, 2008] [Chen, 2012].

4.1. *Languages and Tools for Shared Memory Architectures*

OpenMP is a popular programming option on shared memory systems. OpenMP programming standards consist of compiler directives that define and identify parallel region of the code that can run as threads. Some programs use proprietary compiler directives to form parallelism through threads whereas OpenMP provides a higher level of abstraction to programmers and create parallelism in a fork-and-join programming model [Jost et al., 2003]. In this model program begins sequential execution as a single process or thread. When the directive for parallel region is found, a single thread becomes master thread and creates several other slave threads to execute parallel tasks. At the end of parallel region, all threads are synchronized & joined to produce clubbed results.

In OpenMP programming paradigm, all threads use shared memory which creates possibility of memory contention among threads. This issue is resolved by implementing memory coherence protocol for data consistency. The other popular programming options for shared memory systems are *Portable Operating System Interface (POSIX) Threads* and *Compute Unified Device Architecture (CUDA).*

4.2. *Languages and Tools for Distributed Memory Architectures*

The communication among two or more processors in distributed memory systems are carried out through *Message Passing*. MPI is a popular programming option for distributed memory systems. MPI (stands for ***M**essage **P**assing **I**nterface*) is a specification of message passing libraries for the researchers, developers and users. By itself, it is not a library - but rather the specification of what such a library should be. The goal of the Message Passing Interface is to provide portable, efficient and flexible standard for a wide use of writing message passing programs. The first version of MPI (later called MPI-1) came in existence in 1994. The second version MPI-2, included some more programming issues, was released in 1996 [1], [2].

MPICH is a portable implementation of the full MPI-1 specification for a wide variety of parallel and distributed computing environments. MPICH contains, along with the MPI library itself, a programming environment for working with MPI programs. The programming environment includes a startup mechanism and a profiling library for studying the performance of MPI programs. Interface specifications have been defined for C/C++ and Fortran programs. MPICH2 is a portable implementation of the full MPI-2 specification. Both portable implementation also include :

- Tracing and logfile tools based on the MPI profiling interface, including a scalable logfile format (SLOG).
- Parallel performance visualization tools (*Jumpshot*).
- Extensive correctness and performance tests.

The other popular programming options for distributed memory systems are *Unified Parallel C (UPC)* and *Fortress*.

Both the programming models, viz., OpenMP and MPI, can be used within same program as hybrid MPI+OpenMP paradigm which is suitable for architectures consisting of both shared and distributed memory such as cluster of multi-core processors [Sharma, 2011], [Rabenseifner, 2003]. MPI can be used to provide process level parallelism across nodes while OpenMP can be used to implement loop level parallelism within a node by using compiler directives [Roberts, 2006],

4.3. *Parallel Benchmarks*

By running a benchmark application, relative performance is evaluated in a parallel computing environment. There are several popular benchmarks are available to conduct standard tests and trials. LAPACK (Linear Algebra Package) is a popular benchmark software library for numerical linear algebra. It includes several standard routines such as solving systems of linear equations, eigenvalue problems etc. NAS Parallel Benchmarks (NPB) are a set of benchmarks targeting performance evaluation of highly parallel supercomputers. They are developed and maintained by the NASA Advanced Supercomputing (NAS) Division.

5. Conclusion

Many large computational problems cannot be solved within desired time constraint with sequential computing (using a uni-processor computer). To achieve speedup in computing, more than one processor (computer) are used to solve a large problem, in form of *Parallel and Distributed Computing*. The processors of *Parallel Systems* are generally tightly coupled, i.e.,use shared memory, whereas the processors of *Distributed Systems* are loosely coupled, i.e., use distributed memory and may be scattered in wide geographical area. Distributed systems such as Network of Workstations (NOW), SMP Cluster etc. provide cost-effective solution to large computational problem then dedicated parallel systems.

The multi-core architecture has put an opportunity and challenge to exploit all available cores present in the processor. Cluster of recent multi-core processors is as powerful as earlier days supercomputers. Various parallel programming languages and tools are available for different computing environments. Due to improved computing performance of parallel applications, the demand of parallel programmers is expected to increase in future.

References

[1] Guo Liang Chen, Yun Quan Zhang, “Survey on Parallel Computing,” Journal of Computer Science & Technology, Sept. 2012, Vol. 21, No. 5, pp. 665-673.

[2] Intel Multi-core Series Processors, Available online at: http://www.intel.com/ products/processor/core2quad/.

[3] G. Jost, H. Jin, D. Mey, and F. Hatay, “Comparing the OpenMP, MPI, and Hybrid Programming Paradigms on an SMP Cluster,”The Fifth European Workshop on OpenMP (EWOMP03), Sep.2003.

[4] Henry Kasim, Verdi March, “Survey on Parallel Programming Model,” Proceeding of the International Conference on Information Processing, China, 2008, pp. 266-275.

[5] A. Mamidala et al., “MPI Collectives on Modern Multicore Clusters : Performance Optimizations and Communication Characteristics,” 10th IEEE International Conference on Cluster, Cloud and Grid Computing, Melbourne, May 2010.

[6] J. Roberts and S. Akhtar, “Multi-Core Programming : Increasing Performance through Software Multithreading,” Technical Report, Available online at: http://www.intel.com/intelpress.

[7] Rajkumar Sharma and Priyesh Kanungo, “Performance Evaluation of MPI and Hybrid MPI+OpenMP Programming Paradigms on Multi-Core Processors Cluster,” IEEE International Conference on Recent Trends in Information Systems, Jadavpur University, Kolkata, December 2011, pp. 137-140.

[8] Pradeep K. Sinha, “Distributed Operating System: Concepts and Design,” Prentice Hall of India, 2002.

[9] R. Rabenseifner, “Hybrid Parallel Programming: Performance Problems and Chances,” 45th CUG Conference, Columbus, May 2003, Available online at: http://www.cug.org.

[10] The Message Passing Interface (MPI) standard, http://www.mcs.anl.gov/research/ projects/mpi.

[11] W. Gropp, Ewing Lusk and Rajeev Thakur. ‘Using MPI-2 : Advanced Features of the Message-Passing Interface’. http: // www.mcs.anl.gov/ research/ projects/ mpi/usingmpi2, 2010.