Recent Trends in Parallel Computing

Gurinder Singh
Department of Applied Sciences
UIET, Hoshiarpur
Hoshiarpur, Punjab, India

Aman Kaura
Department of Applied Sciences
UIET, Hoshiarpur
Hoshiarpur, Punjab, India

Abstract

In this paper we have studied the recent trends in parallel computation. A big task that cannot be handled by single CPU can be divided into small number of subtasks which can be processed simultaneously by different processor. At the end these subtasks can be combined together. A parallel computer consists of parallel computing hardware, parallel computing model, software support for parallel programming. This paper explores the different types of parallel computation.

Keywords: Parallel computation, Distributed memory, Parallel Benchmarks, Shared memory

I. INTRODUCTION

Computational requirements are ever increasing, both in the area of scientific and business computing. Silicon based processor chips are reaching their ultimate limits in processing speed, as they are constrained by speed of light and certain thermodynamic laws. In order to overcome this limitation one possibility is to connect the multiple processors working in coordination with each other. There are several different forms of parallel computing: bit-level, instruction level, data and task parallelism. In parallel computing, a computational task is broken down in several similar subtasks that can be processed independently and whose results are combined after the completion of the job. In the past years parallel machines have become significant competitors to vector machines in the quest for high performance computing.

II. Hardware Architectures for Parallel Processing

The core elements of parallel processing are CPUs. On the basis of instructions and data streams that can be processed simultaneously, computer systems can be classified into the following categories

A. Single Instruction Single Data (SISD)
B. Single Instruction Multiple Data (SIMD)
C. Multiple Instruction Single Data (MISD)
D. Multiple Instruction Multiple Data (MIMD)

A. Single Instruction Single Data (SISD)

SISD is a type of computer architecture in which a single uni-core processor executes a single instruction stream, to operate on data stored in a single memory. Most of the conventional computers are based on the SISD model. All the data and instructions have to be stored in the primary memory. The speed of the processing element is limited by the rate at which the computer can transform the information internally. Important examples of the model are Workstations, Macintosh etc. Pipelined processors and superscalar processors are common examples found in most modern SISD computers [1,2]

B. Single Instruction Multiple Data (SIMD)

A SIMD is a multiprocessor machine capable of executing the same instruction on all the CPUs, but operating on different data streams. Machine based on this model are well suited for scientific computing since they involve lot vector and matrix operation.

C. Multiple Instruction Single Data (MISD)

A MISD computing system is a multiprocessor machine capable of executing different instructions on different processing elements, but all of them operating on same data-set. Machines using this model are not useful in most of the applications.

D. Multiple Instruction Multiple Data(MIMD)

A MIMD computing system is a multiprocessor machine capable of executing multiple instructions on multiple data sets. Each processing elements in MIMD model have separate instruction and data stream and hence machines built using this model are suited for different applications. They are broadly classified into shared-memory MIMD and distributed-memory MIMD machines based on how all the processing elements are coupled to the main memory. According to the memory model the parallel computational model can be divided into three categories: shared memory computational model, distributed computational model and hierarchical memory model.

III. Shared Memory Parallel Computing Models

In shared memory architecture a number of processors are connected to a common central memory. Since all processors are sharing a single address space, the data sharing is fast but processes can corrupt each other data at the same time. So the semaphores and locks are used to save the data from corruption. There is a lack of

scalability between processing elements and memory which means that we cannot add processing elements as many as we need to a limited memory.

A. PRAM Model

PRAM model is widely used shared memory computing model for the design and analysis of parallel algorithms and was first developed by Fortune, Wyllie and Goldschlager. Small numbers of processors share a common global pool of memory. The processors are allowed to access the memory concurrently and take only one unit of time to be completed. The PRAM model has different instances. CRCW PRAM model [3] that permits simultaneous read and write to the same memory cell. CREW PRAM [3] is another model that permits simultaneous read to the same memory cell but permits only one processor to write on a cell at a time. Although the PRAM model is easy to implement, it suffers from memory and network contention problem.

IV. Distributed Memory Parallel Computing Models

There are numbers of distributed memory parallel computing models in which each different computer having their own memory are connected through a communication network. Model BSP and LogP models come under this category. These models remove the shortcomings of the shared memory computational memory.

V. Software support for Parallel programming

Designing a parallel programming is a challenging business. Two methodologies are widely used for the purpose of parallel programs. They are auto-parallelization compiler and library based software. Auto parallelization 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.

VI. Parallel Benchmarks used in HPCC

Benchmarks are freely available standardized computer programs that are mostly used by the HPC community to measure the system performance. There are mainly three types of benchmarks: synthetic benchmarks, kernel benchmarks and real application benchmarks. Synthetic benchmarks are small programs and did not involve any real computation but work out the basic functions of a machine. It compares the relative efficiency of processors. Example of this category is

Whetstone benchmark [4], Dhrystone [5] and wPrime etc. In Kernel benchmarks a part of a large program is extracted and this part of program is responsible for most of the time of that problem. Examples are LINPACK [6], NAS etc. In real application benchmarks the code segment is the application program itself. It is very useful in measuring the overall system performance but needs more time resources. Examples are Perfect Benchmarks, SPEC benchmarks etc.

VII. Conclusion

Benchmarks are freely available standardized computer programs that are mostly used by the HPC community to measure the system performance. There are mainly three types of benchmarks: synthetic benchmarks, kernel benchmarks and real application benchmarks. Synthetic benchmarks are small programs and did not involve any real computation but work out the basic functions of a machine. It compares the relative efficiency of processors. Example of this category is Whetstone benchmark [4], Dhrystone [5] and wPrime etc. In Kernel benchmarks a part of a large program is extracted and this part of program is responsible for most of the time of that problem. Examples are LINPACK [6], NAS etc. In real application benchmarks the code segment is the application program itself. It is very useful in measuring the overall system performance but needs more time resources. Examples are Perfect Benchmarks, SPEC benchmarks etc.

References

1. Quinn, Michael J. Chapter 2: Parallel Architectures, Parallel Programming in C with MPI and OpenMP. Boston: McGraw Hill, 2004, ISBN 0-07-282256-2.
2. Ibaroudene, Djaffer. Chapter 1: Motivation and History, Parallel Processing. St. Mary's University, San Antonio, TX. Spring 2008.
3. Jaja J. "An Introduction to Parallel Algorithms". Addison-Wesley, 1992.
4. Curnow H.J and Wichman B.A. "A Synthetic Benchmark". Computer Journal, vol19, no. 1, Feb 1976.
5. Weicker R.P. "DHRYSTONE: A Synthetic Systems Programming Benchmark". Communications of the ACM, vol. 27, no.10, pp. 1013-1030, October, 1984.
6. Dongrra J.J. "Performance of Various Computers Using Standard Linear Equation Software". Tech. Rep. CS-89-85, University of Tennessee and Oak Ridge National Laboratory, November, 1995.