Parallel Computer Architecture

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About this Tutorial

Parallel Computer Architecture is the method of organizing all the resources to maximize the performance and the programmability within the limits given by technology and the cost at any instance of time. It adds a new dimension in the development of computer system by using more and more number of processors.

This tutorial covers the basics related to Parallel Computer Architecture, discussing the various concepts and terminologies associated with the topic.

Audience

This tutorial has been prepared for students pursuing either a master's degree or a bachelor's degree in Computer Science, particularly those who are keen to learn about Parallel Computer Architecture.

Prerequisites

In this tutorial, all the topics have been explained from elementary level. Therefore, a beginner can understand this tutorial very easily. However if you have a prior knowledge of computer architecture in general, then it will be quite easy to grasp the concepts explained here.

Copyright & Disclaimer

Table of Contents

1. PCA – Introduction

In the last 50 years, there has been huge developments in the performance and capability of a computer system. This has been possible with the help of Very Large Scale Integration (VLSI) technology. VLSI technology allows a large number of components to be accommodated on a single chip and clock rates to increase. Therefore, more operations can be performed at a time, in parallel.

Parallel processing is also associated with data locality and data communication. **Parallel Computer Architecture** is the method of organizing all the resources to maximize the performance and the programmability within the limits given by technology and the cost at any instance of time.

Why Parallel Architecture?

Parallel computer architecture adds a new dimension in the development of computer system by using more and more number of processors. In principle, performance achieved by utilizing large number of processors is higher than the performance of a single processor at a given point of time.

Application Trends

With the advancement of hardware capacity, the demand for a well-performing application also increased, which in turn placed a demand on the development of the computer architecture.

Before the microprocessor era, high-performing computer system was obtained by exotic circuit technology and machine organization, which made them expensive. Now, highly performing computer system is obtained by using multiple processors, and most important and demanding applications are written as parallel programs. Thus, for higher performance both parallel architectures and parallel applications are needed to be developed.

To increase the performance of an application **Speedup** is the key factor to be considered. Speedup on p processors is defined as:

$$\boldsymbol{Speedup\ (p\ processors) \equiv \frac{Performance\ (p\ processors)}{Performance\ (1\ processor)}}$$

For the single fixed problem,

$$\boldsymbol{Performance\ of\ a\ computer\ system = \frac{1}{Time\ needed\ to\ complete\ the\ problem}}$$

$$\boldsymbol{Speedup}_{\text{ fixed problem}}\ \boldsymbol{(p\ processors) = \frac{Time\ (1\ processor)}{Time\ (p\ processors)}}$$

Scientific and Engineering Computing

Parallel architecture has become indispensable in scientific computing (like physics, chemistry, biology, astronomy, etc.) and engineering applications (like reservoir modeling, airflow analysis, combustion efficiency, etc.). In almost all applications, there is a huge demand for visualization of computational output resulting in the demand for development of parallel computing to increase the computational speed.

Commercial Computing

In commercial computing (like video, graphics, databases, OLTP, etc.) also high speed computers are needed to process huge amount of data within a specified time. Desktop uses multithreaded programs that are almost like the parallel programs. This in turn demands to develop parallel architecture.

Technology Trends

With the development of technology and architecture, there is a strong demand for the development of high-performing applications. Experiments show that parallel computers can work much faster than utmost developed single processor. Moreover, parallel computers can be developed within the limit of technology and the cost.

The primary technology used here is VLSI technology. Therefore, nowadays more and more transistors, gates and circuits can be fitted in the same area. With the reduction of the basic VLSI feature size, clock rate also improves in proportion to it, while the number of transistors grows as the square. The use of many transistors at once (parallelism) can be expected to perform much better than by increasing the clock rate.

Technology trends suggest that the basic single chip building block will give increasingly large capacity. Therefore, the possibility of placing multiple processors on a single chip increases.

Architectural Trends

Development in technology decides what is feasible; architecture converts the potential of the technology into performance and capability. **Parallelism** and **locality** are two methods where larger volumes of resources and more transistors enhance the performance. However, these two methods compete for the same resources. When multiple operations are executed in parallel, the number of cycles needed to execute the program is reduced.

However, resources are needed to support each of the concurrent activities. Resources are also needed to allocate local storage. The best performance is achieved by an intermediate action plan that uses resources to utilize a degree of parallelism and a degree of locality.

Generally, the history of computer architecture has been divided into four generations having following basic technologies:

- Vacuum tubes
- Transistors
- Integrated circuits
- VLSI

Till 1985, the duration was dominated by the growth in bit-level parallelism. 4-bit microprocessors followed by 8-bit, 16-bit, and so on. To reduce the number of cycles needed to perform a full 32-bit operation, the width of the data path was doubled. Later on, 64-bit operations were introduced.

The growth in **instruction-level-parallelism** dominated the mid-80s to mid-90s. The RISC approach showed that it was simple to pipeline the steps of instruction processing so that on an average an instruction is executed in almost every cycle. Growth in compiler technology has made instruction pipelines more productive.

In mid-80s, microprocessor-based computers consisted of

- An integer processing unit
- A floating-point unit
- A cache controller
- SRAMs for the cache data
- Tag storage

As chip capacity increased, all these components were merged into a single chip. Thus, a single chip consisted of separate hardware for integer arithmetic, floating point operations, memory operations and branch operations. Other than pipelining individual instructions, it fetches multiple instructions at a time and sends them in parallel to different functional units whenever possible. This type of instruction level parallelism is called **superscalar execution**.

2. PCA – Convergence of Parallel Architectures

Parallel machines have been developed with several distinct architecture. In this section, we will discuss different parallel computer architecture and the nature of their convergence.

Communication Architecture

Parallel architecture enhances the conventional concepts of computer architecture with communication architecture. Computer architecture defines critical abstractions (like user-system boundary and hardware-software boundary) and organizational structure, whereas communication architecture defines the basic communication and synchronization operations. It also addresses the organizational structure.

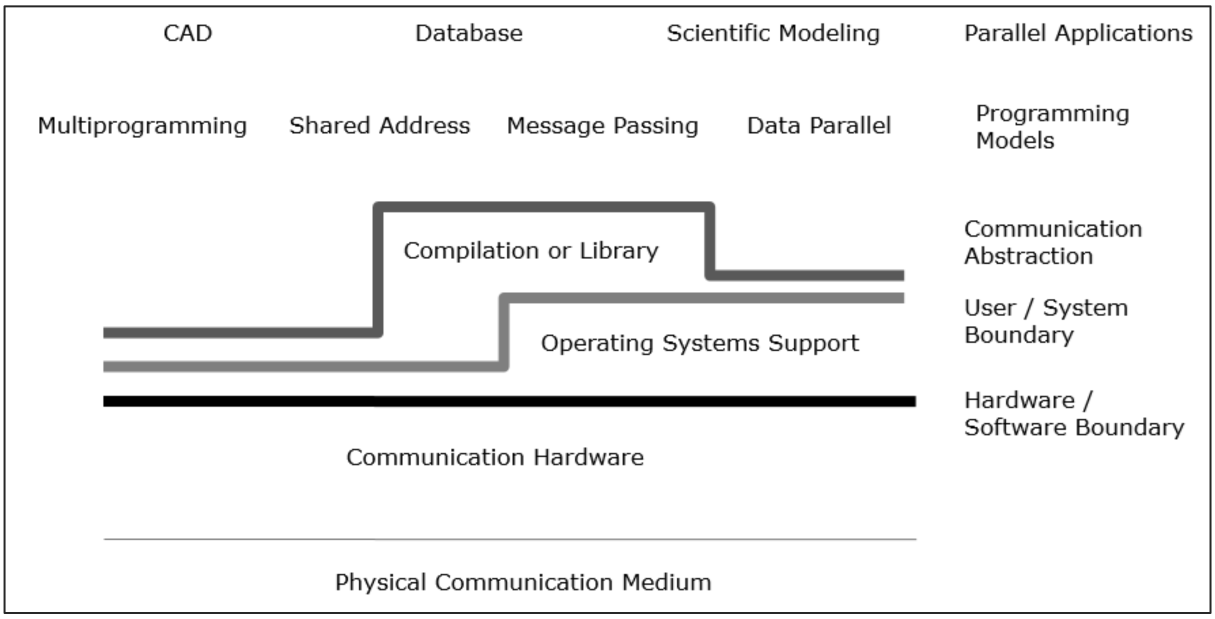

Figure : Layers of Abstraction in Parallel Computer Architecture

Programming model is the top layer. Applications are written in programming model. Parallel programming models include:

- Shared address space
- Message passing
- Data parallel programming

Shared address programming is just like using a bulletin board, where one can communicate with one or many individuals by posting information at a particular location, which is shared by all other individuals. Individual activity is coordinated by noting who is doing what task.

Message passing is like a telephone call or letters where a specific receiver receives information from a specific sender.

Data parallel programming is an organized form of cooperation. Here, several individuals perform an action on separate elements of a data set concurrently and share information globally.

Shared Memory

Shared memory multiprocessors are one of the most important classes of parallel machines. It gives better throughput on multiprogramming workloads and supports parallel programs.

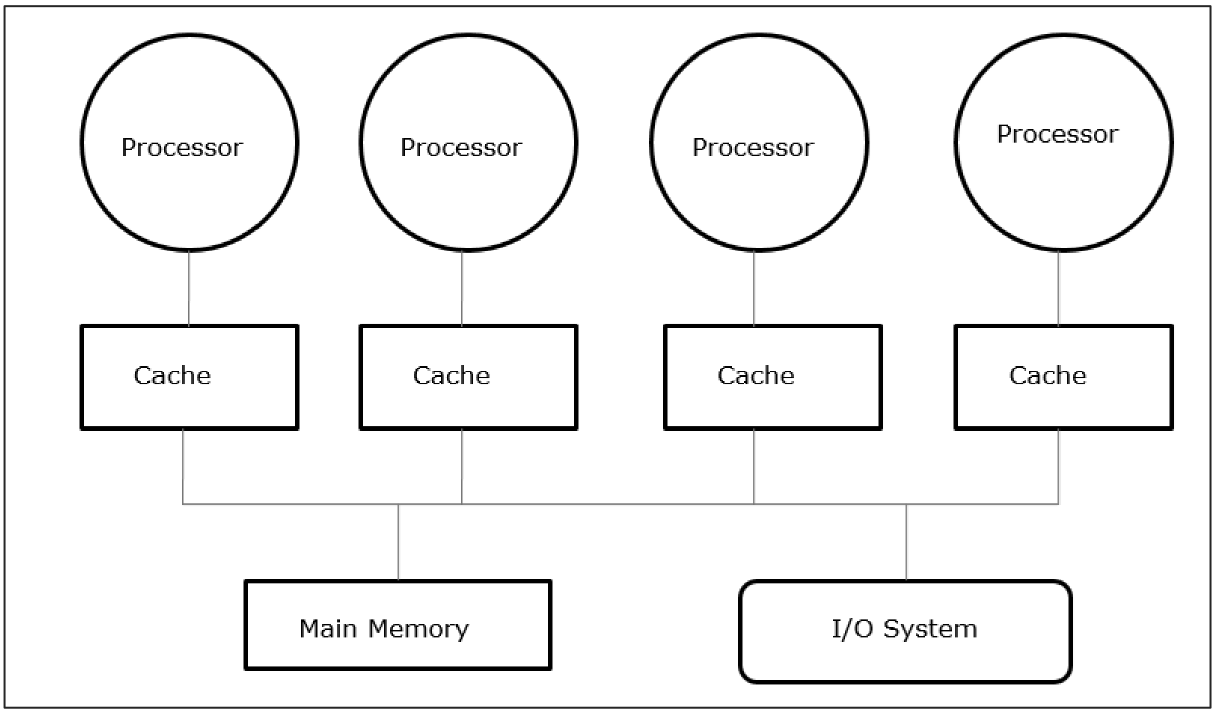

Figure : Shared Memory Multiprocessor

In this case, all the computer systems allow a processor and a set of I/O controller to access a collection of memory modules by some hardware interconnection. The memory capacity is increased by adding memory modules and I/O capacity is increased by adding devices to I/O controller or by adding additional I/O controller. Processing capacity can be increased by waiting for a faster processor to be available or by adding more processors.

All the resources are organized around a central memory bus. Through the bus access mechanism, any processor can access any physical address in the system. As all the processors are equidistant from all the memory locations, the access time or latency of all the processors is same on a memory location. This is called **symmetric multiprocessor**.

Message-Passing Architecture

Message passing architecture is also an important class of parallel machines. It provides communication among processors as explicit I/O operations. In this case, the communication is combined at the I/O level, instead of the memory system.

In message passing architecture, user communication executed by using operating system or library calls that perform many lower level actions, which includes the actual communication operation. As a result, there is a distance between the programming model and the communication operations at the physical hardware level.

Send and **receive** is the most common user level communication operations in message passing system. Send specifies a local data buffer (which is to be transmitted) and a receiving remote processor. Receive specifies a sending process and a local data buffer in which the transmitted data will be placed. In send operation, an **identifier** or a **tag** is attached to the message and the receiving operation specifies the matching rule like a specific tag from a specific processor or any tag from any processor.

The combination of a send and a matching receive completes a memory-to-memory copy. Each end specifies its local data address and a pair wise synchronization event.

Convergence

Development of the hardware and software has faded the clear boundary between the shared memory and message passing camps. Message passing and a shared address space represents two distinct programming models; each gives a transparent paradigm for sharing, synchronization and communication. However, the basic machine structures have converged towards a common organization.

Data Parallel Processing

Another important class of parallel machine is variously called: processor arrays, data parallel architecture and single-instruction-multiple-data machines. The main feature of the programming model is that operations can be executed in parallel on each element of a large regular data structure (like array or matrix).

Data parallel programming languages are usually enforced by viewing the local address space of a group of processes, one per processor, forming an explicit global space. As all the processors communicate together and there is a global view of all the operations, so either a shared address space or message passing can be used.

Fundamental Design Issues

Development of programming model only cannot increase the efficiency of the computer nor can the development of hardware alone do it. However, development in computer architecture can make the difference in the performance of the computer. We can understand the design problem by focusing on how programs use a machine and which basic technologies are provided.

In this section, we will discuss about the communication abstraction and the basic requirements of the programming model.

Communication Abstraction

Communication abstraction is the main interface between the programming model and the system implementation. It is like the instruction set that provides a platform so that the same program can run correctly on many implementations. Operations at this level must be simple.

Communication abstraction is like a contract between the hardware and software, which allows each other the flexibility to improve without affecting the work.

Programming Model Requirements

A parallel program has one or more threads operating on data. A parallel programming model defines what data the threads can **name**, which **operations** can be performed on the named data, and which order is followed by the operations.

To confirm that the dependencies between the programs are enforced, a parallel program must coordinate the activity of its threads.