MODULE 1: PARALLELISM FUNDAMENTALS
WEEK 1

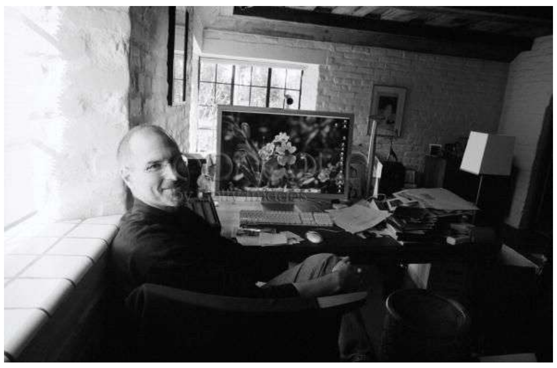
Learning Outcomes:

After completing this course you are expected to demonstrate the following:

1. Define parallelism, its origin, and multiple simultaneous computations which includes in the study of parallelism fundamentals.

A. Engage

Trivia



There is an ever-increasing appetite among some types of computer users for faster and faster machines. This was epitomized in a statement by the late Steve Jobs, founder/CEO of Apple and Pixar. He noted that when he was at Apple in the 1980s, he was always worried that some other company would come out with a faster machine than his. But later at Pixar, whose graphics work requires extremely fast computers, he was always hoping someone would produce faster machines, so that he could use them!

B. Explore

Video Title: **Parallel Computing Explained In 3 Minutes**
YouTube Link:**http://youtube.com/watch?v=q7sgzDH1cR8**
Module Video Filename: **Week 1 -Parallel Computing Explained In 3 Minutes**

C. Explain

As the Pixar example shows, highly computation-intensive applications like computer graphics also have a need for these fast parallel computers. No one wants to wait hours just to generate a single image, and the use of parallel processing machines can speed things up considerably. For example, consider ray tracing operations. Here our code follows the path of a ray of light in a scene, accounting for refection and absorption of the light by various objects. Suppose the image is to consist of 1,000 rows of pixels, with 1,000 pixels per row. In order to attack this problem in a parallel processing manner with, say, 25 processors, we could divide the image into 25 squares of size 200x200, and have each processor do the computations for its square.

D. Elaborate

Introduction

A few years ago, parallel computers could be found only in research laboratories. Today they are widely available commercially. The field of parallel processing has matured to the point wherecomputer science undergraduates should study it. Parallelism covers a wide spectrum of material, from hardware design of adders to the analysis of theoretical models of parallel computation. In fact, aspects of parallel processing could be incorporated into every computer science course in the curriculum. This text introduces the important principles of parallel processing.This chapter introduces important terms and concepts. Also, to set the stage for later chapters, a brief history of parallelism is included. Parallel machines provide a wonderful opportunity for applications with large computational requirements. Effective use of these machines, though, requires a keen understanding of how they work. This chapter provides an overview of both the software and hardware.

What is Parallel Computing?

Parallel Computing is a type of computation in which many calculations are performed at the same time.

Basic principle: computation can be divided into smaller subprograms, each of which can be solved simultaneously.

Assumption: we have parallel hardware at our dispossal, which is capable of executing these computations in parallel.

Brief History of Computers and Parallel Computing

Parallel computing was present since the early days of computing. parallel computing assumes the existence of some sort of parallel hardware which is capable of undertaking this computation simultaneously that is in parallel having said that we will take a short glance into the history of parallel computing to better understand its origin despite its recent rise in popularity parallel execution was present ever since the early days of computing when computers were still mechanical devices it is difficult to identify the exact conception of the first parallel computing machine but we can agree on the following in the 19th century Charles Babbage invented the concept of a programmable computer which he called analytical engine the analytical engine was able to execute variuos user of written programs for this Babbage is credited for being a pioneer in the field of programmable computers after Babbge presented his ideas about the analytical engine and italian general and mathematician Luigi Menabrea down a description of Babbage's computer menabrea envisioned an extension in which independent computations occur simultaneously at the same time his idea is to speed up Babbage's analytical engine modern computers based on transistor technology appeared much later in the 20th century researchers from IBM Daniel Slotnick, John Kock and Jean Andale to mention a few ware one of the early evangelist of parallel processing Andals famous quote comes from this era he said;

For over a decade prophets have voiced the contention that the organization of a sigle computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers. Gene Andahl, 1967.

Regardless of Amdahl's visionary views at the time parallel computing did not become mainstream but it attained a niche with the high – performance computing community. In high performance computing in the main area of focus our parallel algorithms for physics and biology simulations better forecasting cryptography and applications that require a lot of processing power performance requirements of regular users were satisfied by increasing the clock frequency of the CPU which meant the CPU could execute a higher number of instructions per second at the beginning of the 21st century it became obvious that scaling the processor frequency is no longer feasible after a certain point the reason is that the power required for the processor starts to grow non linearly with with frequency this is known as the power wall so instead of increasing CPU clock frequency, processor vendors turn to providing multiple CPU cores on the same processor chip, each capable of executing separate instruction streams amd named these new divises as multi – core processors. It is interesting to note that these three stories share a common theme whether the need for faster computation arose from the slugishness of a mechanical computer limitations of the available technology or the physical boundaries imposed by the powerwall additional computer power was provided through parallelization.

In the late 1970s and early 1980s, distributed memory architectures were investigated by a wide range of groups. However, it was not until the early to mid-1980s that machines containing more than one processor and conforming to the Multiple Instruction stream Multiple Data stream (MIMD) classification were available commercially. At this stage, it is worth distinguishing between the two major MIMD categories, namely Shared Memory MIMD (SM-MIMD) and Distributed Memory MIMD (DM-MIMD) computers. The SM-MIMD group are considered to be tightly coupled, whereby the instruction streams can be programmed to work together on the solution of a single problem, whilst the DM-MIMD are regarded as loosely coupled and employ the relatively slow message passing approach. The CRAY X-MP, containing essentially 2 CRAY 1 processors was announced in 1982 and was the first multi-processor to be shipped (Dongarra et al, 1991) and by 1984 a 4 processor version was also available. Each machine used pipelining to achieve very high performance and the machines had a peak performance of 420 and840 Mflop/s, respectively. In 1985, the Intel iPSC (Personal Super Computer) hypercube, the commercial variant of the Cosmic Cube, entered the supercomputing market in sizes of 25, 26 and 27 and the largest system was rated at 8 Mflop/s. However, by 1986 Intel was offering improved node performance by the inclusion of a pipelined processor. This increased the peak performance of each node to 20 Mflop/s at the expense of reducing the total number of nodes to 64. However, the overall peak performance had now increased to approximately 430 Mflop/s in 64 bit arithmetic.

It was previously stated that the component speed had shown a thousand fold increase between 1950and 1975: Has there been a similar dramatic increase since 1975? Figure 1.0 illustrates a small number of some well-known computer systems in an attempt to provide an answer this question. From figure 1.0, itis evident that the component speed, as measured by the clock cycle time, has not improved in such a dramatic fashion, particularly for the vector processors. Since 1976 (the year the Cray 1 was delivered) the clock cycle has been reduced from 12.5 ns to 2.2 ns for the Cray T90, which was first delivered.

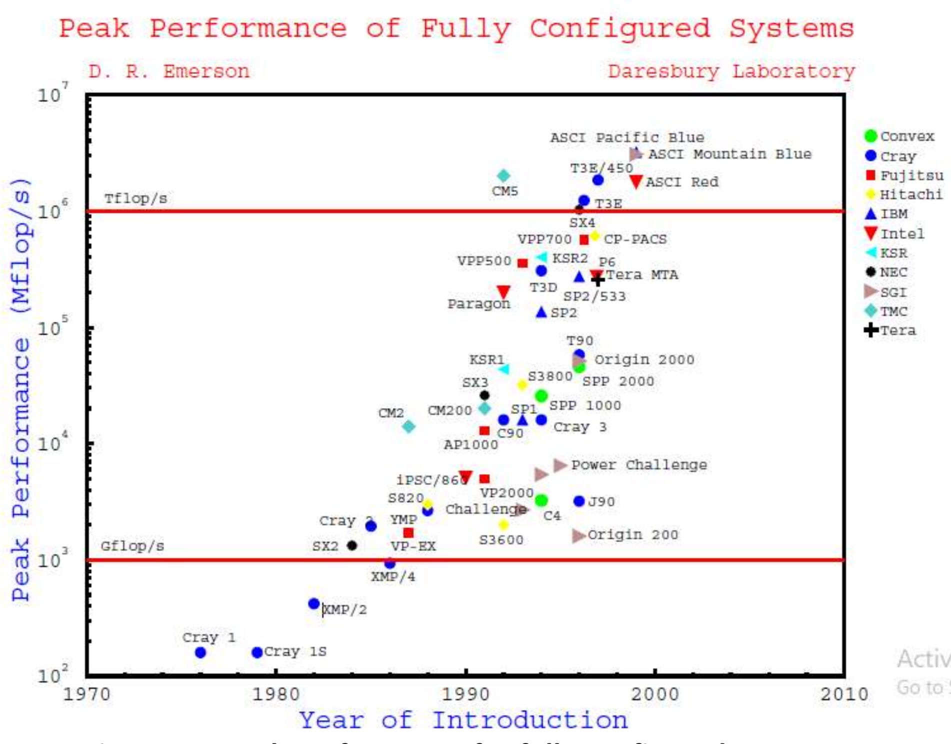


Figure 1.0 Peak Performance for fully configured Systems

Figure 1.0 clearly shows that the vector supercomputers no longer dominate in terms of peak performance. Many of the top systems belong in the Massively Parallel Processing (MPP) class and contain inexpensive off-the-shelf components. Another factor that is evident is that parallelism is necessary to achieve performances on this scale and the dramatic improvement since 1976, from 160 Mflop/s, to machines delivering over 1 Tflop/s (1,000,000 Mflop/s) is phenomenal. The fastest machine in the world today (1997), as measured by the Linpack benchmark, is Intel’s ASCI Option Red machine at Sandia.This position was formerly held for a considerable time by the Fujitsu VPP500 Numerical Wind Tunnel (NWT). The memory of this machine was physically distributed and each processor has 256 MBytes of memory. It is, therefore, a DM-MIMD machine with 140 vector processors although it could have up to 222 processors, as indicated by the performance quoted in figure 3. The progress indicated in figure 3 has not been made without cost. In 1994, two major vendors, Thinking Machines Corporation and Kendall Square Research, filed for bankruptcy. This was followed in March 1995 by Seymore Cray’s company, Cray Computer Corporation. In November, 1995, Convex Computer Corporation became a subsidiary of Hewlett-Packard and in February, 1996, Cray Research merged with Silicon Graphics. Other recent “mergers” have been Quadrics Alenia with Meiko. Hence, the supercomputing world is extremely volatile. However, from the foregoing discussion, it is evident that parallel processing is here to stay and, at present, remains the only way to achieve high performance in computing.

Platforms

For parallel computing, one must always keep in mind what hardware and software environments we will be working in. Our hardware platforms here will be multicore, GPU and clusters. For software we will use C/C++, OpenMP, MPI, CUDA and R.

What is Parallelism?

The first term to define is “parallelism.” When computer scientists talk of parallel processing, they are not talking of processing “two straight lines that never meet.” They are discussing several computational activities happening at the same time.

Parallelism- several activities happening at the same time.

Parallelism need not refer to computing. For example, writing notes while listening to a lecture are parallel activities. However, we must be careful because the phrase "at the same time" is imprecise. For example, a uni-processor VAX might appear to be computing at the same time for several time shared users because the processor handles instructions so fast. Here we have the illusion of instructions being executed simultaneously, i. e., the illusion of parallelism. However, since the processor is executing instructions for only one job at a time, it is not true parallelism.

What is Parallel Computing?
Computations that use multi-processor computers and/or several independent computers interconnected in some way, working together on a common task.

• **Examples:** CRAY T3E, IBM-SP, SGI-3K, Cluster of Workstations.

In general, we would like to design a parallel program in which it is easy to vary granularity: i.e. **a scalable program design**.

The best choice would be a fully connected network in which each processor has a direct link to every other processor. Unfortunately, this type of network would be very expensive and difficult to scale. Instead, processors are arranged in some variation of a grid, torus, hypercube, etc. Key issues in network design are the network bandwidth and the network latency. The bandwidth is the number of bits that can be transmitted in unit time, given as bits/sec. The network latency is the time to make a message transfer through the network.

1. Static Interconnects
• Consist of point-to-point links between processors
• Can make parallel system expansion easy
• Some processors may be "closer" than others
• Examples: Line/Ring, Mesh/Torus, Tree, Hypercube

Why use Parallel Programming?
Parallel programming is much harder than sequential programming. Separating sequential computations into parallel subcomputations can be challenging, or even impossible. Usually this means identifying independent parts of the program and executing at the same time however, some problems cannot be divided as shown in this figure. The second source of difficulty is ensuring the correctness we will see that many new types of errors can arise in parallel programming making the programmers life harder with these increased the complexities by them would be butter writing parallel programs at all as shown in earlier anecdotes the main benefit of parallelization is increased program performance when we write a parallel program we expect it to be faster than its sequential counterpart thus speed-up is the main reason why we bother paying for this complexity.

A major source of speedup is the parallelizing of operations. Parallel operations can be either within processor, such as with pipelining or having several ALUs within a processor, or between processor, in which many processors work on different parts of a problem in parallel. Our focus here is on between-processor operations.

For example, the Registrar's office at UC Davis uses shared-memory multiprocessors for processing its on-line registration work. Online registration involves an enormous amount of database computation. In order to handle this computation reasonably quickly, the program partitions the work to be done, assigning different portions of the database to different processors. The database filedhas contributed greatly to the commercial success of large shared-memory machines.

Parallel Program uses parallel hardware to execute computation more quickly. Efficiency is its main concern.

Concurrent Program may or may not execute multiple executions at the same time. It improves modularity, responsiveness or maintainability.

The main concerns of parallel and concurrent programs are as follows;

Parallel Program:
1. Division into subprograms
2. Optimal use of parallel hardware

Concurrent Program:
1. When can an execution start
2. How can information exchange occur
3. How to manage access to shared resources

On a parallel computer, user applications are executed as processes, tasks or threads. The traditional definition of process is a program in execution. To achieve an improvement in speed through the use of parallelism, it is necessary to divide the computation into tasks or processes that can be executed simultaneously. The size of a process can be described by its granularity.

1. **Fine-grain -** In fine granularity, a process might consist of a fewinstructions, or perhaps even one instruction.

2. **Medium-grain -** Medium granularity describes the middle ground between fine-grain and coarse grain.

3. **Course-grain -** In course granularity, each process contains a large number of sequential instructions and takes a substantial time toexecute.

Sometimes **granularity** is defined as the size of the computation between communication or synchronization points. Generally, we want to increase the granularity to reduce the cost of process creation and inter process communication, but of course this will likely reduce the number of concurrent processes and the amount of parallelism. A suitable compromise has to be made.

Parallel Programming Models
A parallel computer system should be flexible and easy to use and should exhibit good programmability in supporting various parallel algorithms.

1. **Explicit parallelism** means that parallelism is explicitly specified in the source code by the programmer using special language constructs, compiler directives or library function calls. If the programmer does not explicitly specify parallelism, but lets the compiler and the run-time support system automatically exploit it, we have the implicit parallelism.

2. **Implicit Parallelism** means that Parallelizing Compilers Automatic parallelization of sequential programs do not exploit functional parallelism Compiler, performs

dependence analysis on a sequential program's source data and then – using a suite of program transformation techniques – converts the sequential code into a native parallel code. Some performance studies indicate, however, that the parallelizing compilers are not very effective.

Although many explicit programming models have been proposed, three models have become dominant ones: data parallel, message passing and shared variable.

3. **Data parallel** Execute the same instruction or program segment over different data sets simultaneously on multiple computing nodes. Has a single thread of control. Parallelism is exploited at data set level. No functional parallelism available.

4. **Message-passing model** (Multithreading) a message-passing program consists of multiple processes, each of which has its own thread of control and may execute different code. Both control parallelism (MPMD – Multiple-Program-Multiple-Data) and data parallelism (SPMD – Single-Program-Multiple-Data) are supported.

    1. **Asynchronous** – the processes of a message-passingprogram execute asynchronously.Separate address space - the processes of a parallelprogram reside in different address spaces.

    2. **Explicit interactions** – the programmer must solve all theinteraction issues, including data mapping, communication and synchronization. Scales well, especially if data is well distributed.

Shared variable Similar to data-parallel model, in that it has single address space. Similar to message-passing model, in that it is multithreading and asynchronous. Data reside in a single, shared address space and does not have to be explicitly allocated. Communication is done implicitly through shared reads and writes of variables Synchronization is explicit.

Types of Parallelism: Two Extremes

1. Data parallel
a. Each processor performs the same task on different data
b. Data mapping is critical
c. Programmed with HPF or message passing
d. Example – grid problems

2. Task parallel
a. Each processor performs a different task
b. More difficult to balance load
c. Commonly programmed with message passing
d. Example – signal processing

Most applications fall somewhere on the continuum between these two extremes

E. Evaluate

ASSESSMENT:
Instructions: You may write your answer on the Answer Sheet (AS) provided in this module
Point : 2 points each

CONTENT FOR ASSESSMENT:

1. Computations that use multi-processor computers and/or several independent computers interconnected in some way, working together on a common task.
2. Consist of point-to-point links between processors.
3. In course granularity, each process contains a large number of sequential instructions and takes a substantial time to execute.
4. It means Parallelizing Compilers Automatic parallelization of sequential programs do not exploit functional parallelism Compiler.
5. Execute the same instruction or program segment over different data sets simultaneously on multiple computing nodes.

References:

1. *Introduction to Parallel Computing Using Advanced Architectures andAlgorithms - https://www.researchgate.net/publication/321151707*
2. *Lecture Notes on Parallel Computation -Stefan Boeriu, Kai-Ping Wang and John C. Bruch Jr. Office of Information Technology andDepartment of Mechanical and Environmental Engineering University of California, Santa Barbara, CA*
3. *Programming on Parallel Machines - Norm Matlo University of California, Davis*

Facilitated By:		
Name	:	
MS Teams Account (email)	:	
Smart Phone Number	:	