UNIT 3 RECENT TRENDS IN PARALLEL COMPUTING

Structure **Page Nos.**

3.0 INTRODUCTION

This unit discusses the current trends of hardware and software in parallel computing. Though the topics about various architectures, parallel program development and parallel operating systems have already been discussed in the earlier units, here some additional topics are discussed in this unit.

3.1 OBJECTIVES

After studying this unit you should be able to describe the

- various parallel programming models;
- concept of grid computing;
- concept of cluster computing;
- IA-64 architecture, and
- concept of Hyperthreading.

3.2 RECENT PARALLEL PROGRAMMING MODELS

A model for parallel programming is an abstraction and is machine architecture independent. A model can be implemented on various hardware and memory architectures. There are several parallel programming models like Shared Memory model, Threads model, Message Passing model, Data Parallel model and Hybrid model etc.

As these models are hardware independent, the models can (theoretically) be implemented on a number of different underlying types of hardware.

The decision about which model to use in an application is often a combination of a number of factors including the available resources and the nature of the application. There is no universally best implementation of a model, though there are certainly some implementations of a model better than others. Next, we discuss briefly some popular models.

1) Shared Memory Model

In the shared-memory programming model, tasks share a common address space, which they read and write asynchronously. Various mechanisms such as locks / semaphores may be used to control access to the shared memory. An advantage of this model from the programmer's point of view is that program development can often be simplified. An important disadvantage of this model (in terms of performance) is that for this model, data management is difficult.

2) Threads Model

In this model a single process can have multiple, concurrent execution paths. The main program is scheduled to run by the native operating system. It loads and acquires all the necessary softwares and user resources to activate the process. A thread's work may best be described as a subroutine within the main program. Any thread can execute any one subroutine and at the same time it can execute other subroutine. Threads communicate with each other through global memory. This requires Synchronization constructs to insure that more than one thread is not updating the same global address at any time. Threads can be created and destroyed, but the main program remains live to provide the necessary shared resources until the application has completed. Threads are commonly associated with shared memory architectures and operating systems.

3) Message Passing Model

In the message-passing model, there exists a set of tasks that use their own local memories during computation. Multiple tasks can reside on the same physical machine as well across an arbitrary number of machines. Tasks exchange data by sending and receiving messages. In this model, data transfer usually requires cooperation among the operations that are performed by each process. For example, a send operation must have a matching receive operation.

4) Data Parallel Model

In the data parallel model, most of the parallel work focuses on performing operations on a data set. The data set is typically organised into a common structure, such as an array or a cube. A set of tasks work collectively on the same data structure with each task working on a different portion of the same data structure. Tasks perform the same operation on their partition of work, for example, “add 3 to every array element” can be one task. In shared memory architectures, all tasks may have access to the data structure through the global memory. In the distributed memory architectures, the data structure is split up and data resides as “chunks” in the local memory of each task.

5) Hybrid model

The hybrid models are generally tailormade models suiting to specific applications. Actually these fall in the category of mixed models. Such type of application-oriented models keep cropping up. Other parallel programming models also exist, and will continue to evolve corresponding to new applications.

In this types of models, any two or more parallel programming models are combined. Currently, a common example of a hybrid model is the combination of the message passing model (MPI) with either the threads model (POSIX threads) or the shared memory model (OpenMP). This hybrid model lends itself well to the increasingly common hardware environment of networked SMP machines.

Another common example of a hybrid model is combining data parallel model with message passing model. As mentioned earlier in the data parallel model, data parallel implementations (F90, HPF) on distributed memory architectures actually use message passing to transmit data transparently between tasks and the programmer.

6) Single Program Multiple Data (SPMD)

SPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models. A single program is executed by all tasks simultaneously. SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute. That is, tasks do not necessarily have to execute the entire program, they may execute only a portion of it. In this model, different tasks may use different data.

7) Multiple Program Multiple Data (MPMD)

Like SPMD, MPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.
MPMD applications typically have multiple executable object files (programs). While the application is being run in parallel, each task can be executed on the same or different program. In this model all tasks may use different data.

3.3 PARALLEL VIRTUAL MACHINE (PVM)

PVM is basically a simulation of a computer machine running parallel programs. It is a software package that permits a heterogeneous collection of Unix and/or Windows computers hooked together by a network to be used as a single large parallel computer. It includes a combination of good features of various operating systems and architectures. Thus large computational problems can be solved more cost effectively by using the combined power and memory of many computers. The software is highly portable. The source, which is available free through netlib, has been compiled on a range of computing machines from laptops to CRAY machines.

PVM enables users to exploit their existing computer hardware to solve much larger problems at minimal additional cost. Hundreds of sites around the world are using PVM to solve important scientific, industrial, and medical problems. Further, PVM's are being used as an educational tools to teach parallel programming. With tens of thousands of users, PVM has become the de facto standard for distributed computing world-wide.

Check Your Progress 1

1) Explain the operations in the following message passing operating system model
   1) Object Oriented Model
   2) Node addressed model
   3) Channel addressed model

2) Explain various Multi Processor Execution Node.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

3) What are the levels at which multitasking is exploited?

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

3.4 GRID COMPUTING

Grid Computing means applying the resources of many computers in a network simultaneously to a single problem for solving a scientific or a technical problem that requires a large number of computer processing cycles or accesses to large amounts of data. Grid computing uses software to divide and distribute pieces of a program to as many as several thousand computers. A number of corporations, professional groups and university consortia have developed frameworks and software for managing grid computing projects.

Thus, the Grid computing model allows companies to use a large number of computing resources on demand, irrespective of where they are located. Various computational tasks can be performed using a computational grid. Grid computing provides clustering of remotely distributed computing environment. The principal focus of grid computing to date has been on maximizing the use of available processor resources for compute-intensive applications. Grid computing along with storage virtualization and server virtualization enables a utility computing. Normally it has a Graphical User Interface (GUI), which is a program interface based on the graphics capabilities of the computer to create screens or windows.

Grid computing uses the resources of many separate computers connected by a network (usually the internet) to solve large-scale computation problems. The SETI@home project, launched in the mid-1990s, was the first widely-known grid computing project, and it has been followed by many others project covering tasks such as protein folding, research into drugs for cancer, mathematical problems, and climate models.

Grid computing offers a model for solving massive computational problems by making use of the unused resources (CPU cycles and/or disk storage) of large numbers of disparate, often desktop, computers treated as a virtual cluster embedded in a distributed telecommunications infrastructure. Grid computing focuses on the ability to support computation across administrative domains which sets it apart from traditional computer clusters or traditional distributed computing.

Various systems which can participate in the grid computing as platform are:
Windows 3.1, 95/98, NT, 2000 XP , DOS, OS/2, , supported by Intel (x86);
Mac OS A/UX (Unix) supported by Motorola 680 x0;
Mac OS , AIX (Unix), OS X (Unix) supported by Power PC;
HP / UX (Unix) supported by HP 9000 (PA – RISC);
Digital Unix open VMS, Windows NT spported by Compaq Alpha;
VMS Ultrix (Unix) supported by DEC VAX;
Solaris (Unix) supported by SPARC station;
AIX (Unix) supported by IBM RS / 6000;
IRIS (Unix) supported by Silicon Graphics workstation.

3.5 CLUSTER COMPUTING

The concept of clustering is defined as the use of multiple computers, typically PCs or UNIX workstations, multiple storage devices, and their interconnections, to form what appears to users as a single highly available system. Workstation clusters is a collection of loosely-connected processors, where each workstation acts as an autonomous and independent agent. The cluster operates faster than normal systems.

In general, a 4-CPU cluster is about 250~300% faster than a single CPU PC. Besides, it not only reduces computational time, but also allows the simulations of much bigger computational systems models than before. Because of cluster computing overnight

analysis of a complete 3D injection molding simulation for an extremely complicated model is possible.
Cluster workstation defined in ***Silicon Graphics*** project is as follows:

"A distributed workstation cluster should be viewed as single computing resource and not as a group of individual workstations".

The details of the cluster were: 150MHz R4400 workstations, 64MB memory, 1 GB disc per workstation, 6 x 17" monitors, Cluster operating on local 10baseT Ethernet, Computing Environment Status, PVM, MPI (LAM and Chimp), Oxford BSP Library.

The operating system structure makes it difficult to exploit the characteristics of current clusters, such as low-latency communication, huge primary memories, and high operating speed. Advances in network and processor technology have greatly changed the communication and computational power of local-area workstation clusters. Cluster computing can be used for load balancing in multi-process systems

A common use of cluster computing is to balance traffic load on high-traffic Web sites. In web operation a web page request is sent to a manager server, which then determines which of the several identical or very similar web servers to forward the request to for handling. The use of cluster computing makes this web traffic uniform.

Clustering has been available since the 1980s when it was used in DEC's VMS systems. IBM's SYSLEX is a cluster approach for a mainframe system. Microsoft, Sun Microsystems, and other leading hardware and software companies offer clustering packages for scalability as well as availability. As traffic or availability assurance increases, all or some parts of the cluster can be increased in size or number. Cluster computing can also be used as a relatively low-cost form of parallel processing for scientific and other applications that lend themselves to parallel operations

An early and well-known example was the project in which a number of off-the-shelf PCs were used to form a cluster for scientific applications.

Windows cluster is supported by the Symmetric Multiple Processors (SMP) and by Moldex3D R7.1. While the users chose to start the parallel computing, the program will automatically partition one huge model into several individual domains and distribute them over different CPUs for computing. Every computer node will exchange each other's computing data via message passing interface to get the final full model solution. The computing task is parallel and processed simultaneously. Therefore, the overall computing time will be reduced greatly.

Some famous projects on cluster computing are as follows:

i) **High Net Worth Project**: (developed by: Bill McMillan, JISC NTI/65 – The HNW Project, University of Glasgow, (billm@aero.gla.ac.uk)

The primary aims / details of the project are:

- Creation of a High Performance Computing Environment using clustered workstations
- Use of pilot facility,
- Use of spare capacity,
- Use of cluster computing environment as a Resource Management system,
- Use of ATM communications,
- Parallel usage between sites across SuperJANET,
- Promoting Awareness of Project in Community.

ii) **The primary aims/details of Load Sharing Facility Resource Management Software(LSFRMS)** are better resource utilisation by routing the task to the most appropriate system and better utilisation of busy workstations through load sharing, and also by involving idle workstations.

Types of Users of the LSFRMS:

- users whose computational requirements exceed the capabilities of their own hardware;
- users whose computational requirements are too great to be satisfied by a single workstation;
- users intending to develop code for use on a national resource;
- users who are using national resources
- users who wish to investigate less probable research scenarios.

Advantages of using clusters:

- Better resource utilisation;
- Reduced turnaround time;
- Balanced loads;
- Exploitation of more powerful hosts;
- Access to non-local resources;
- Parallel and distributed applications.

Check Your Progress 2

1) Name any three platforms which can participate in grid computing.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

2) Explain the memory organization with a cluster computing.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

3) Find out the names of famous cluster computer projects.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

4) Explain various generations of message passing multi computers.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

3.6 INTEL ARCHITECTURE – 64 (IA-64)

IA-64 (Intel Architecture-64) is a 64-bit processor architecture developed in cooperation by Intel and Hewlett-Packard, implemented by processors such as Itanium. The goal of Itanium was to produce a “post-RISC era” architecture using EPIC(Explicitly Parallel Instruction Computing).

1 EPIC Architecture

In this system a complex decoder system examines each instruction as it flows through the pipeline and sees which can be fed off to operate in parallel across the available execution units — *e.g.*, a sequence of instructions for performing the computations
A = B + C and
D = F + G

These will be independent of each other and will not affect each other, and so they can be fed into two different execution units and run in parallel. The ability to extract instruction level parallelism (ILP) from the instruction stream is essential for good performance in a modern CPU.

Predicting which code can and cannot be split up this way is a very complex task. In many cases the inputs to one line are dependent on the output from another, but only if some other condition is true. For instance, consider the slight modification of the example noted before, A = B + C; IF A==5 THEN D = F + G. In this case the calculations remain independent of the other, but the second command requires the results from the first calculation in order to know if it should be run at all.

In these cases the circuitry on the CPU typically "guesses" what the condition will be. In something like 90% of all cases, an IF will be taken, suggesting that in our example the second half of the command can be safely fed into another core. However, getting the guess wrong can cause a significant performance hit when the result has to be thrown out and the CPU waits for the results of the "right" command to be calculated. Much of the improving performance of modern CPUs is due to better prediction logic, but lately the improvements have begun to slow. Branch prediction accuracy has reached figures in excess of 98% in recent Intel architectures, and increasing this figure can only be achieved by devoting more CPU die space to the branch predictor, a self-defeating tactic because it would make the CPU more expensive to manufacture.

IA-64 instead relies on the compiler for this task. Even before the program is fed into the CPU, the compiler examines the code and makes the same sorts of decisions that would otherwise happen at "run time" on the chip itself. Once it has decided what paths to take, it gathers up the instructions it knows can be run in parallel, bundles them into one larger instruction, and then stores it in that form in the program.

Moving this task from the CPU to the compiler has several advantages. First, the compiler can spend considerably more time examining the code, a benefit the chip itself doesn't have because it has to complete the task as quickly as possible. Thus the compiler version can be considerably more accurate than the same code run on the chip's circuitry. Second, the prediction circuitry is quite complex, and offloading a prediction to the compiler reduces that complexity enormously. It no longer has to examine anything; it simply breaks the instruction apart again and feeds the pieces off to the cores. Third, doing the prediction in the compiler is a one-off cost, rather than one incurred every time the program is run.

The downside is that a program's runtime-behaviour is not always obvious in the code used to generate it, and may vary considerably depending on the actual data being processed. The out-of-order processing logic of a mainstream CPU can make decisions on the basis of actual run-time data which the compiler can only guess at. It means that it is possible for the compiler to get its prediction wrong more often than comparable (or simpler) logic placed on the CPU. Thus this design this relies heavily on the performance of the compilers. It leads to decrease in microprocessor hardware complexity by increasing compiler software complexity.

Registers: The IA-64 architecture includes a very generous set of registers. It has an 82-bit floating point and 64-bit integer registers. In addition to these registers, IA-64 adds in a register rotation mechanism that is controlled by the Register Stack Engine. Rather than the typical spill/fill or window mechanisms used in other processors, the Itanium can rotate in a set of new registers to accommodate new function parameters or temporaries. The register rotation mechanism combined with predication is also very effective in executing automatically unrolled loops.

Instruction set: The architecture provides instructions for multimedia operations and floating point operations.

The Itanium supports several bundle mappings to allow for more instruction mixing possibilities, which include a balance between serial and parallel execution modes. There was room left in the initial bundle encodings to add more mappings in future versions of IA-64. In addition, the Itanium has individually settable predicate registers to issue a kind of runtime-determined "cancel this command" directive to the respective instruction. This is sometimes more efficient than branching.

Pre-OS and runtime sub-OS functionality

In a raw Itanium, a "Processor Abstraction Layer" (PAL) is integrated into the system. When it is booted the PAL is loaded into the CPU and provides a low-level interface that abstracts some instructions and provides a mechanism for processor updates distributed via a BIOS update.
During BIOS initialization an additional layer of code, the "System Abstraction Layer" (SAL) is loaded that provides a uniform API for implementation-specific platform functions.

On top of the PAL/SAL interface sits the "Extensible Firmware Interface" (EFI). EFI is not part of the IA-64 architecture but by convention it is required on all IA-64 systems. It is a simple API for access to logical aspects of the system (storage, display, keyboard, etc) combined with a lightweight runtime environment (similar to DOS) that allows basic system administration tasks such as flashing BIOS, configuring storage adapters, and running an OS boot-loader.

Once the OS has been booted, some aspects of the PAL/SAL/EFI stack remain resident in memory and can be accessed by the OS to perform low-level tasks that are implementation-dependent on the underlying hardware.

IA-32 support

In order to support IA-32, the Itanium can switch into 32-bit mode with special jump escape instructions. The IA-32 instructions have been mapped to the Itanium's functional units. However, since, the Itanium is built primarily for speed of its EPIC-style instructions, and because it has no out-of-order execution capabilities, IA-32 code executes at a severe performance penalty compared to either the IA-64 mode or the Pentium line of processors. For example, the Itanium functional units do not automatically generate integer flags as a side effect of ordinary ALU computation, and do not intrinsically support multiple outstanding unaligned memory loads. There are also IA-32 software emulators which are freely available for Windows and Linux, and these emulators typically outperform the hardware-based emulation by around 50%. The Windows emulator is available from Microsoft, the Linux emulator is available from some Linux vendors such as Novell and from Intel itself. Given the superior performance of the software emulator, and despite the fact that IA-32 hardware accounts for less than 1% of the transistors of an Itanium 2, Intel plan to remove the circuitry from the next-generation Itanium 2 chip codenamed "Montecito".

3.7 HYPER-THREADING

Hyper-threading, officially called **Hyper-threading Technology** (**HTT**), is Intel's trademark for their implementation of the simultaneous multithreading technology on the Pentium 4 microarchitecture. It is basically a more advanced form of Super-threading that was first introduced on the Intel Xeon processors and was later added to Pentium 4 processors. The technology improves processor performance under certain workloads by providing useful work for execution units that would otherwise be idle for example during a cache miss.

Features of Hyper-threading

The salient features of hyperthrading are:

i) Improved support for multi-threaded code, allowing multiple threads to run simultaneously.

ii) Improved reaction and response time, and increased number of users a server can support.

According to Intel, the first implementation only used an additional 5% of the die area over the "normal" processor, yet yielded performance improvements of 15-30%.
Intel claims up to a 30% speed improvement with respect to otherwise identical, non-SMT Pentium 4. However, the performance improvement is very application dependent, and some programs actually slow down slightly when HTT is turned on.

This is because of the replay system of the Pentium 4 tying up valuable execution resources, thereby starving for the other thread. However, any performance degradation is unique to the Pentium 4 (due to various architectural nuances), and is not characteristic of simultaneous multithreading in general.

Functionality of hypethread Processor

Hyper-threading works by duplicating those sections of processor that store the architectural state—but not duplicating the main execution resources. This allows a Hyper-threading equipped processor to pretend to be two "logical" processors to the host operating system, allowing the operating system to schedule two threads or processes simultaneously. Where execution resources in a non-Hyper-threading capable processor are not used by the current task, and especially when the processor is stalled, a Hyper-threading equipped processor may use those execution resources to execute the other scheduled task.

Except for its performance implications, this innovation is transparent to operating systems and programs. All that is required to take advantage of Hyper-Threading is symmetric multiprocessing (SMP) support in the operating system, as the logical processors appear as standard separate processors.

However, it is possible to optimize operating system behaviour on Hyper-threading capable systems, such as the Linux techniques discussed in Kernel Traffic. For example, consider an SMP system with two physical processors that are both Hyper-Threaded (for a total of four logical processors). If the operating system's process scheduler is unaware of Hyper-threading, it would treat all four processors similarly.

As a result, if only two processes are eligible to run, it might choose to schedule those processes on the two logical processors that happen to belong to one of the physical processors. Thus, one CPU would be extremely busy while the other CPU would be completely idle, leading to poor overall performance. This problem can be avoided by improving the scheduler to treat logical processors differently from physical processors; in a sense, this is a limited form of the scheduler changes that are required for NUMA systems.

The Future of Hyperthreading

Current Pentium 4 based MPUs use Hyper-threading, but the next-generation cores, Merom, Conroe and Woodcrest will not. While some have alleged that this is because Hyper-threading is somehow energy inefficient, this is not the case. Hyper-threading is a particular form of multithreading, and multithreading is definitely on Intel roadmaps for the generation after Merom/Conroe/Woodcrest. Quite a few other low power chips use multithreading, including the PPE from the Cell processor, the CPUs in the Playstation 3 and Sun's Niagara. Regarding the future of multithreading
the real question is not whether Hyper-threading will return, as it will, but rather how it will work. Currently, Hyper-threading is identical to Simultaneous Multi-Threading, but future variants may be different. In future trends parallel codes have been easily ported to the Pilot Cluster, often with improved results and Public Domain and Commercial Resource Management Systems have been evaluated in the Pilot Cluster Environment.

The proposed enhancements in these architecture are as follows:

- High Performance language implementation
- Further developments in Heterogeneous Systems
- Management of wider domains with collaborating departments
- Faster communications
- Inter-campus computing

Check Your Progress 3

1) How is performance enhanced in IA-64 Architecture?
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

2) How is performance judged on the basis of run time behaviour?
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

3) List some data parallelism feature of a modern computer architecture.
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

3.8 SUMMARY

This unit discusses basic issues in respect of latest trends of hardware and software technology for parallel computing systems. The technology of parallel computing is the outcome of four decades of research and industrial advances in various fields like microelectronics, integration technologies, advanced processors, memory and peripheral devices, programming language development, operating system developments etc.

The high performance computers which are developed using these concepts provide fast and accurate solutions to scientific, engineering, business, social and aerospace applications.

The unit discusses some of the latest architectures. However, the learner is advised to get the knowledge about further ongoing developments from related websites.

3.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1) i) **Object Oriented Model:** This model treats multiple tasks or processor concurrently, each task using its own address space. In this model multitasking is practiced at each node.
The communication between different tasks is carried out by either at synchronous or asynchronous message passing protocol. Message passing supports IPC as well as process migration or load balancing.

ii) **Node Address Model:** In this only one task runs on each node at a time. It is implemented with asynchronous scheme. The actual topology is hidden from the user. Some models use store and forward message routing and some models use wormhole routing.

iii) **Channel Addressed Model:** This model runs one task on each node. Data are communicated by synchronous message passing for communication channel.

2) a) Multi Processor super computers are built in vector processing as well as for parallel processing across multiple processors. Various executions models are parallel execution from fine grain process level.
b) Parallel execution from coarse grain process level
c) Parallel execution to coarse grain exam level

3) Multi tasking exploits parallelism at the following level:

i) The different functional unit is pipeline together.
ii) Various functional units are used concurrently.
iii) I/O and CPU activities are overlapped.
iv) Multiple CPUs can operate on a single program to achieve minimum execution time.

In a multi tasking environment, the task and the corresponding data structure of a program are properly pertaining to parallel execution in a such a manner that conflicts will not occur.

Check Your Progress 2

1) The following platform can participate in grid computing.
i) Digital, Unit, Open, VMS
ii) AIX supported by IBM RS/6000.
iii) Solaris (Unix supported by SPARC station)

2) In cluster computer the processor are divided into several clusters. Each cluster is a UMA or NUMA multiprocessor. The clusters are connected to global shared memory model. The complete system works on NUMA model.

All processing elements belonging to it clusters are allowed to access the cluster shared memory modules.

All clusters have access to global memory. Different computers systems have specified different access right among Internet cluster memory. For example, CEDAR multi processor built at University of Illinois adopts architecture in each cluster is Alliant FX/80 multiprocessor.

3) Collect them from the website.

4) Multi computers have gone through the following generations of development. The first generally ranged from 1983-87. It was passed on Processor Board Technology using Hypercube architecture and Software controlled message switching. Examples of this generation computers are Caltech, Cosmic and Intel, PEPSC.

The second generation ranged from 1988-92. It was implemented with mesh connected architecture, hardware access routing environment in medium grain distributed computing. Examples of such systems are Intel Paragon and Parsys super node 1000.

The third generation ranged from 1983-97 and it is an era of fine grain computers like MIT, J Machine and Caltech mosaic.

Check Your Progress 3

1) The performance in IA 64 architecture is enhanced in a modern CPU due to vector prediction logic and in branch prediction technique the path which is expected to be taken as is anticipated in advance, in order to preset the required instruction in advance. Branch prediction accuracies has reached to more than 98% in recent Intel architecture, and such a high figure is achieved by devoting more CPU die space to branch prediction.

2) The run time behaviour of a program can be adjudged only at execution time. It depends on the actual data being processed. The out of order processing logic of a main stream CPU can make itself on the basis of actual run time data which only compiler can connect.

3) Various data parallelism depends as specified as to how data is access to distribute in SIMD or MIMD computer. These are as follows:

   i) **Run time automatic decomposition:** In this data are automatically distributed with no user intervention.

   ii) **Mapping specifications:** It provides the facility for user to specify the communication pattern.

   iii) **Virtual processor support:** The compiler maps virtual process dynamically and statically.

   iv) **Direct Access to sharing data:** Shared data can be directly accessed without monitor control.

3.10 FURTHER READINGS

1) Thomas L. Casavant, Pavel Tvrdik, Frantisck Plasil, *Parallel Computers: Theory and Applications*, IEEE Computer Society Press (1996)

2) Kai Hwang: *Advanced Computer Architecture: Parallelism*, Scalability and Programmability, Tata-McGraw-Hill (2001)