Prepared by Diwas KC

Notes on Parallel and Distributing Computing, First Semester MSC CSIT

1

Foundations

1.1 Parallel and Distributed Computing: the scene, the Props, the Players

1.1.1 A Perspective

The majority of today's computers are conceptually very similar. Their architecture and models more or less follows the same basic design principle formulated by **John Von Neumann** in the early 1940's. The von neumann approach centered around the simple idea of a control unit that fetches an instruction and its operands from memory unit and sends them to a processing unit. In processing unit, the instruction is executed, and the result is sent back to memory.

In such computer, the instructions are executed **sequentially**, where the cpu executes one instruction at a time. But there was a problem called *cpu to memory bottleneck*, which mainly results from disparity of high cpu speeds and much lower memory speeds. Several techniques were introduced to improve these early designs, such as *cache memory*, *outlining* which led to the conception of early super computers. However this trend is coming to end due to several limitations of sequential computations.

A **Parallel or Distributed Architecture** is the one that consists of collection of processing units that cooperates to solve a given problem by working simultaneously on different part of that problem. There is no limit, at least in principle, to the number of actions that can be executed in parallel; hence the concept offers an arbitrary degree of improvement in the speed of computation.

1.1.2 Parallel Processing Paradigms

Computers operate simply by executing instructions on data. A stream of instructions inform the computer what to do at each step. Flynn classified the architecture of the computer on the basis of how the machine relates its instructions to the data being processed. Four classifications are:

1. Single Instruction Stream, Single Data Stream (**SISD**)
2. Single Instruction Stream, Multiple Data Stream (**SIMD**)

3. Multiple Instruction Stream, Single Data Stream (**MISD**)
4. Multiple Instruction Stream, Multiple Data Stream (**MIMD**)

All Von neuman machines belongs to SISD. An algorithm running in SISD are also called *sequential algorithm*.

A SIMD consists of N processors, N memories and an interconnection network and a control unit.

In a MISD computer, each processor has its own control unit and share a common memory unit where data reside.

MIMD machines are most general and powerful in the paradigm of parallel computing. In this case, there are N processors, N memories and N streams of data.

1.1.3 Modeling and Characterizing Parallel Algorithms

A von neumann type of architecture can be studied using an abstract model which is called the random access machine model or **RAM model**.The RAM model can be used to theoretically study sequential algorithm and evaluate their complexities.

The parallel counterpart to the RAM model is parallel random access machine or **PRAM model**. A PRAM is basically a set of synchronous processors connected to a shared memory. A main feature of PRAM is capability for diffrent processors to simultaneously make references to distinct memory cells.

PRAMS are useful for studying parallel algorithms and evaluating their behaviour and properties. This is because, if an algorithm doesnot perform well on PRAM, then it will be pointless to try and implement it on real parallel architecture.

1.1.4 Cost vs Performance Evaluation

There are several issues related to parallelization that do not arise in sequential programming.

The first issue is **task allocation**, which is the breakdown of workload into smaller tasks assigned to diffrent processors, and the proper sequencing of task when some of them are independent and cannot be executed simultaneously. To achieve the higher level of performance, it is important to ensure that each processor is properly utilized. And this process is called *load balancing or scheduling*, and it consider to be extremly 'formidable' to solve and also belongs to NP-Complete problems.

The second issue is **communication time** between the processor. The problem of communication becomes more serious when the number of processors increase to hundred or thousands.

The choice of better *inter connection network* also plays an important role in achieving high performance.

Once a new algorithm for given problem has been designed, it is usually evaluated. The evaluation helps in quantifying how well an architecture matches a particular algorithm. Some of the measurements used for performance evaluation are given below:

1.1.4.1 Execution Time

Speeding up computations appears to be one of the main reasons behind the interest in building PDC systems.So, the only complete and reliable measure of performance is time, which cane be defined as *the time taken by the algorithm to solve a problem on a prallel computer, or the time elapsed from the moment the algorithm starts to the moment it terminates.*

If the various processors do not all begin and end their operation sumultaneously, then the execution time is equal to the time elapsed between the moment the first processor begins computing and the moment the last processor computing.

1.1.4.2 Speedup

It is quite natural to evaluate a parallel algorithm in terms of the best availale sequential algorithm for that problem. Thus, a good indication of the quality of a parallel algorithm is the speedup it produces. This is defined as:

$$S = \frac{"Runningtimeofthebestavailablesequentialalgorithm"}{"Runingtimeoftheparallelalgorithm"}$$

Obvously, the larger the speedup, the better the parallel algorithm.Ideally, one hopes to achieve the maximum speedup of N when solving such a problem using N processors operaing in parallel. In practice, such spedup cannot be achieved for every problem because:

1. It is not always possible to decompose a proble into N tasks

Amadahl has pointed out that the speedup is limited by the amount of parallelism inherent in the algorithm which can be characterized by a parameter d, the fraction of the computation that must be done sequentially. He reasons that the maximum speedup of an N-processor system in executing an algorithm as a function of f is given by:

$$S_N <_/ \frac{1}{f + (1 - f)/N} <= 1/fforeveryN$$

Note that ($S_{\max} = 1$) (no speedup) where (f=1) (all computations must be sequential), and ($S_{\max} = N$) where (f=0) (all computations must be in parallel).

Moreover in many algorithms, the interprocessor communication time significantly contributes to the overall execution time. Hence, the effect of communication (or communication penalty) can be characterized by the following ration:

$$c_{pi} = \frac{E_i}{C_i}$$

Where E_i is the total execution time spent by processor i to run the algorith, and C_i is the corresponding time attributed only to communication.

1.1.5 Software and General-Purpose PDC

One of the most important of the problems that face PDC today is the construction of standard parallel languages and software portability. In reality, these are the same factors that made sequential computing very popular. Some researchers even claim that it is worth sacrificing some performance if one is to solve the software problem.

The process of writing parallel programs must be made more systematic. In other words, programmers should be able to write a parallel program, translate it into a workable version and port it to variety of PDC platforms. To write a parallel program in a machine independent fashion requires the development of abstract models for parallel computations without sacrificing implementability of these programs. Of these paradigms, one can list *communicating sequential processes(CSP)*, *calculus of communicating systems(CCS)*, and *Perti nets*.

When a program is executed on a computer. two of the most important issues are how long it will take to execute and how much memory it will require. Turing machines are the simplest and most widely used theoretical model to study sequential computation. Although turing machines are very slow, they tend to capture the essence of the computing process. The importance of a theoretical model, such as the Turing machine, stems from the fact that such a model enables one to highlight the strengths and limitations of a given computational mpde. Moreover Turing machines are also capable of compuing any function that is computatb;e by any other theoretical model. (eg: finite automata, push down automata, linear bound automata).

1.2 Semantics of concurrent programming

In sequential programming, a program in execution has a single thread of control, so that when it is executing, a single instruction is active at any one time. *Concurrent Programming* refers to programs whose execution may have more than one thread of control, i.e, more than one active instruction at a time.

The study of programming language semantics deals with assigning meanings to programs or program parts written in a specific programming language. Semantic definitions of programming languages can be divided into three broad categories: **axiomatic semantic definitions**, which define the semantics of a language by making statement(under the form of axioms) about each construct of the language; **operational semantic definitions** , which define the semantics of a language by defining the effect of each construct of the language on the underlying machine (or, equivalently, how the underlying machine proceeds to execute each statement of the language); and **denotational semantic definitions**, which define the semantics of a language by mapping each one of its constructs into a mathematical object that represts its meaning.

1.2.1 Models of Concurrent Programming

We have identified a number of attributes of concurrency model; each model can be defined by the values that it takes for these attributes.

1. **Level of granularity**: We consider two agents who are accessig a centralized database simultaneously (eg. two bank tellers or two travel agents). IF we observe their activities at the transaction level, the we find naturally that their activities are concurrent, given that they are performing their transactions at the same time. If, on the other hand, we observe their activities at the level of CPU cycle, the we find that their activities are sequential, because the CPU takes turns serving their queries. The activities of these agents can be considered as concurrent or sequential depending on the level of granularity at which we observe them. One of the key features of any concurrent programming model is the level of granularity(also called the *level of atomicity*) at which actions are considered.
2. **Sharing the clock**: When several processes operate concurrently and must evetually interact (eg: to exchange information), it is important to determine whether they share the same clock. The options they have for the exchange of information vary considerably, depending on whether they share the same clock.
3. **Sharing Memory**:When several process operate concurrently and must eventually interact, it is important to determine whether they share some memory space. If they do, this has several implications: first, this space may be used for sharing information; second access tot his space may have to be mutually exclusive, hence affecting the operation of the processes; third, the level of granularity of the processes's activities is then typically larger than or equal to memory access.
4. **Pattern of interaction**: There exists two forms of interaction between processes: synchornization and communication. Synchornization defines a chronological order between events taking place within diffrent processes. Communication defines a transfer of information from one processs to another.

5. **Pattern of synchronization**: There exist two patterns of synchronization between processes: mutual exclusion and mutual admission. Mutual exclusion is defined by an encapsulated sequence of actions whose execution is indivisible; once a process starts executing this sequence, it may not be interrupted until the sequence is completed. Mutual admission is defined by an encapsulated sequence of actions which must be executed by two process simultaneously; if one process is ready and the other is not, it must wait until both are ready to give the sequence their undivided attention.
6. **Pattern of communication**: There exist two modes of communication between processes: synchronous communication and asynchornous communication. In synchronous communication, the transfer of information is achieved while both communicating process are giving their undivided attention to the transfer. In asynchronous communication, the transfer is achieved in two steps: first, the sender deposits the message in a mailbox area; second the receiver picks it up from the mailbox. The implementation of the transfer depends to a large extent on the characterstics of the mailbox: whether the mailbox has finite capacity or infinite capacity; whether messages are ordered by their time of arrival or placed in an arbitary order; and whether the mailbox is public or private.

1.2.2 Semantic Definitions

A programming language is defined by two features: its syntax, which defines the set of legal sentences in the language; and its semantics which assigns meaning to legal sentences of the language. Three broad techniques are known nowadays for the definitions of programming language semantics: `axiomatic definition`; and *denotational definition.*

1.2.2.1 Axiomatic semantics

The axiomatic semantic definition of a programming language defines the meaning of language constructs by mkaing statements (in the form of axioms or inference rules) about the effect of such constructs on the state of the program. The effect of programming construct on the state of the program is defined by *precondition* and a *postcondition.* The preconfition describes the state of the program before execution and the postcondition describes the state of the program after the execution. This method of semantic definition, which comes about as a by product of Hoare's program verification method, is based on the following notation:

$$\{p\}S\{q\}$$

where p and q are state predicates and S is a progarm or a statement. The formula is interpreted as follows: IF statement S is executed in a state where p holds, then it terminates, and q holds upon termination of S.