Parallel Processing Technology (Parallel computing and its models)

Lecture 02: Jan. 10, 2022

Prof. K.R. Chowdhary *: Professor of CS*

2.1 Parallel Computing and Parallel Algorithms

Let Π be an arbitrary computational problem which is to be solved by a computer. Usually our first objective is to design an algorithm for solving Π . Clearly, the class of all algorithms is infinite, but we can partition it into two subclasses, the class of all *sequential algorithms* and the class of *all parallel algorithms*. While a sequential algorithm performs one operation in each step, a parallel algorithm may perform multiple operations in a single step. In this lecture, we will be mainly interested in parallel algorithms. So, our objective is to design a parallel algorithm for Π.

Let P be an arbitrary parallel algorithm. We say that there is parallelism in P. The parallelism in P can be exploited by various kinds of parallel computers. For instance, multiple operations of P may be executed simultaneously by multiple processing units of a parallel computer C_1; or, perhaps, they may be executed by *multiple pipelined* functional units of a single-processor computer C_2. After all, P can always be sequentially executed on a single-processor computer C_3 simply by executing P's potentially parallel operations one by one in succession.

Let $C(p)$ be a parallel computer of the kind C (C_1 or C_2 or C_3) which contains p processing units. Naturally, we expect the performance of program P on $C(p)$ to depend both on C and p. We must, therefore, clearly distinguish between the potential parallelism in P on the one side, and the actual capability of $C(p)$ to execute in parallel, the multiple operations of P, on the other side. So the performance of the algorithm P on the parallel computer $C(p)$ depends on $C(p)$'s capability to exploit P's potential parallelism.

Before we continue, we must unambiguously define what we really mean by the term "performance" of a parallel algorithm P. Intuitively, the "performance" might mean the time required to execute P on $C(p)$; this is called the parallel execution time (or, parallel runtime) of P on $C(p)$, which we will denote by, T_p.

Alternatively, we might choose the "performance" to mean how many times is the parallel execution of P on $C(p)$ faster than the sequential execution of P (T_s); this is called the speedup S of parallel algorithm P on computer $C(p)$,

$$S = \frac{T_s}{T_p}. \tag{2.1}$$

So parallel execution of algorithm P on computer $C(p)$ is S-times faster than sequential execution of P.

Next, we might be interested in how much of the speedup S is on average, due to each of the processing units. Put differently, the term "performance" might be understood as the average contribution of each of the p processing units of $C(p)$ to the speedup; this is called the "efficiency" E of P on $C(p)$,

$$E = \frac{S}{P}. \tag{2.2}$$

Since $T_p \leq T_s \leq pT_p$, it follows that speedup is bounded above by p and efficiency is bounded above by

$$E \leq 1. \tag{2.3}$$

Note that speedup increases due to increse in number of processors p, i.e., more you pay, more you gain. but, how much is gain due to each processor on the average, is efficiency. So is also very important.

This means that, for any C and p, the parallel execution of P on $C(p)$ can be at most p times faster than the execution of P on a single processor. And the efficiency of the parallel execution of P on $C(p)$ can be at most 1. (This is when each processing unit is continually engaged in the execution of P, thus contributing $\frac{1}{p}$-th to its speedup.)

From the above definitions we see that both speedup and efficiency depend on T_p, the parallel execution time of program P on computer $C(p)$. This raises new questions:

- How do we determine T_p?
- How does T_{par} depend on C (the type of the parallel computer) ?
- Which properties of C must we take into account in order to determine T_p ?

These are important general questions about parallel computation, and must be answered prior to embarking on a practical design and analysis of parallel algorithms. The answer to answer these questions help in appropriately modeling the *parallel computation*, discussed in the next section.

2.2 Modeling Parallel Computation

Parallel computers vary in big way in their organization. We will see that their processing units may or may not be directly connected one to another; some of the processing units may share a common memory while the others may only own local (private) memories; the operation of the processing units may be synchronized by a common clock (synchronous), or they may run each at its own pace (asynchronous). Furthermore, usually there are architectural details and hardware specifics of the components, all of which show up during the actual design and use of a computer. And finally, there are technological differences, which prevail in different clock rates, memory access times etc. Hence, the following important question arises:

"Which properties of parallel computers must be considered and which may be ignored in the design and analysis of parallel algorithms?"

To answer this question, we apply ideas of sequential computation. We discussed various models of computation. In short, the intention of each of these models was to abstract away the relevant properties of the (sequential) computation from the irrelevant ones.

In our case, a model called the Random Access Machine (RAM) is particularly attractive. The reason is that RAM distills the important properties of the general-purpose sequential computers, which are still extensively used today, and which have actually been taken as the conceptual basis for modeling of parallel computing and parallel computers. Figure 2.1 shows the simple structure of the RAM.

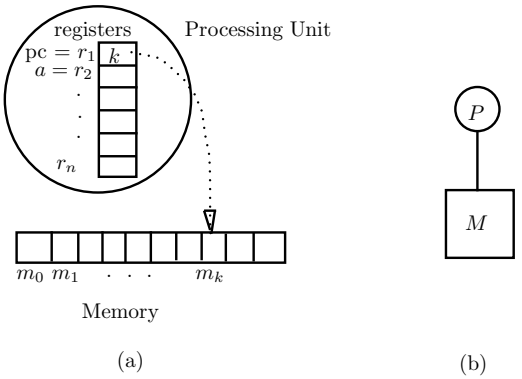

Figure 2.1: (a) RAM model of computation with memory M and processing unit P, (b) Symbol of RAM model

The RAM consists of a processing unit and a memory. The memory is a potentially infinite sequence of equally sized locations $m_0, m_1, \ldots$. The index i is called the address of m_i. Each location is directly accessible by the processing unit: given an arbitrary i, reading from m_i or writing to m_i is accomplished in constant time.

Registers are a sequence $r_1 \ldots r_n$ of locations in the processing unit. Registers are directly accessible. Two of them have special roles. Program counter pc ($=r_1$) contains the address of the location in the memory which contains the instruction to be executed next. Accumulator a ($=r_2$) is involved in the execution of each instruction. Other registers are given roles as needed. The program is a finite sequence of instructions (similar to those in real computers).

Before the RAM is started, the following is done: (a) a program is loaded into successive locations of the memory starting with, say, m_0; (b) input data are written into empty memory locations, say after the last instruction in the program. From now on, the RAM operates independently in a mechanical step-wise fashion as instructed by the program. Let $pc = k$ at the beginning of a step. (Initially, $k = 0$.) From the location m_k, the instruction I is read and executed. At the same time, pc is incremented. So, when I is completed, the next instruction to be executed is at m_{k+1}, unless I was one of the instructions that change pc (e.g. *jump* instructions).

So the above question can be modified as : What is the appropriate model of parallel computation? It turned out that finding an answer to this question is substantially more challenging than it was in the case of sequential computation, because, as there are many ways to organize parallel computers, there are also many ways to model them; and what is difficult is to select a single model that will be appropriate for all parallel computers. As a result, in the past several models were proposed for parallel computation. However, so far no common agreement has been reached about which is the right one. In the following, we

describe the models based on RAM.

Self Review Questions

1. How do you differentiate between sequential and parallel algorithms?
2. What are the possible hardware based computer classes on which you can run parallel algorithms?
3. What is speedup in parallel computers?
4. What is meaning of *performance* of parallel computer?
5. What questions we must answer before we model a parallel computer?
6. "For any computer C and processors' count p, the execution of a program P on $C(p)$ can be at most p times faster than P on a sequential processor." Justify this.
7. How a RAM is different from conventional computer?
8. How a PRAM is different from conventional computer?