Parallel Processing Technology (Multiprocessor Models)

Lecture 03: Jan. 13, 2022

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

3.1 Multiprocessor Models

A multiprocessor model of parallel computation is based on the RAM model of serial computation, i.e., it generalizes the RAM. In fact, the generalization can be done in three essentially different ways resulting in three different multiprocessor models. Each of the three models has some number p ($p \geq 2$) of processing units, but the models differ in the organization of their memories and in the way the processing units access the memories.

These models are called:

1. Parallel Random Access Machine (PRAM),
2. Local Memory Machine (LMM), and
3. Modular Memory Machine (MMM).

3.1.1 Parallel Random Access Machine

The Parallel Random Access Machine, in short PRAM model, has p processing units that are all connected to a common unbounded shared memory (Fig. 3.1). Each processing unit can, in one step, access any location (word) in the shared memory by issuing a memory request directly to the shared memory.

The PRAM model of parallel computation is idealized in several respects. First, there is no limit on the number p of processing units, but p is finite. Next, also idealistic is the assumption that a processing unit can access any location in the shared memory in one single step. Finally, for words in the shared memory it is only assumed that they are of the same size, but can be of arbitrary finite size.

Note that in this model there is no interconnection network for transferring memory requests and data back and forth between processing units and shared memory.

However, the assumption that any processing unit can access any memory location in one step is unrealistic. To see why, suppose that processing units P_i and P_j simultaneously issue instructions I_i and I_j where both instructions intend to access (for reading from or writing to) the same memory location L (Fig. 3.1(b)).

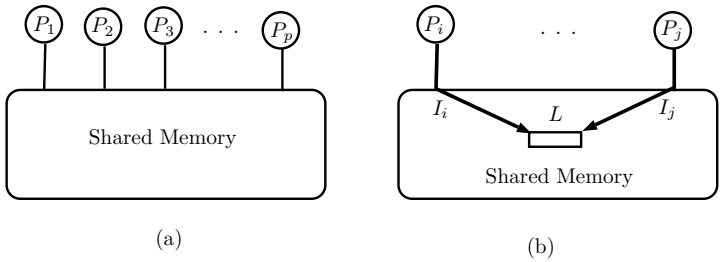

Figure 3.1: (a) The PRAM model of parallel computation, (b) Hazards of simultaneous access to a location. Two processing units simultaneously issue instructions each of which needs to access the same location L

The Fig. 3.1(a) shows The PRAM model of parallel computation, where p processing units share an unbounded memory. Each processing unit can in one step access any memory location. The Fig. 3.1(b) shows hazards of simultaneous access to a location. Two processing units simultaneously issue instructions each of which needs to access the same location L.

Even if a truly simultaneous physical access to L had been possible, such an access could have resulted in unpredictable contents of L. Imagine what would be the contents of L after simultaneously writing the values 3 and 5 into it. Thus, it is reasonable to assume that, eventually, actual accesses by instructions I_i and I_j to L are somehow, on the fly serialized by hardware so that I_i and I_j physically access L one after the other[1].

Does such an implicit serialization neutralize all hazards of simultaneous access to the same location? Unfortunately not so. The reason is that the order of physical accesses of I_i and I_j to L is unpredictable: after the serialization, we cannot know whether I_i will physically access L before or after I_j. Consequently, also the effects of instructions I_i and I_j are unpredictable. This is as follows: If both P_i and P_j want to read simultaneously from L, the instructions I_i and I_j will both read the same contents of L, regardless of their serialization, so both processing units will receive the same contents of L–as expected. However, if one of the processing units wants to read from L and the other simultaneously wants to write to L, then the data received by the reading processing unit will depend on whether the reading instruction has been serialized before or after the writing instruction. Moreover, if both P_i and P_j simultaneously attempt to write to L, the resulting contents of L will depend on how I_i and I_j have been serialized, i.e., which of I_i and I_j was the last to physically write to L.

In sum, simultaneous access to the same location may end in unpredictable data in the accessing processing units as well as in the accessed location.

In view of the conclusion drawn above, the question arises: Does this unpredictability make the PRAM model useless? The answer is no, as we will see shortly.

There can be several variants of PRAMs. For example, what types of simultaneous accesses to the same location are allowed, and the way in which unpreidtability is avoided when simultaneously accessing the same location. The variants of PRAMs are as follows:

- Exclusive Read Exclusive Write PRAM (EREW-PRAM): This is the most realistic

[1]Note that memory modification is through fetch-modify-rewrite cycle. If fetch of location L is done by I_i followed with thsi I_j fetches L, and modification is done in the same order, the I_j will overwrite L that has been modified by I_i, and not the one which it has fetched.

of the three variations. This model does not support simultaneous accessing to the same memory location; if such an attempt is made, the model stops executing its program. Accordingly, the implicit assumption is that programs running on EREW-PRAM never issue instructions that would simultaneously access the same location. Construction of such programs is the responsibility of algorithm designers.

- Concurrent Read Exclusive Write PRAM (CREW-PRAM): This model supports simultaneous reads from the same memory location but requires exclusive writes to it. Again, the burden of constructing such programs is on the algorithm designer.
- Concurrent Read Concurrent Write PRAM (CRCW-PRAM): This is the least realistic of the three versions of the PRAM model. The CRCW-PRAM model allows simultaneous reads from the same memory location, simultaneous writes to the same memory location, and simultaneous reads from and writes to the same memory location. However, to avoid unpredictable effects, different additional restrictions are imposed on simultaneous writes. This yields the following versions of the model CRCW-PRAM: 1. *Consistent* (processing unit may simultaneously attempt to write to L, but assumed that all write the same value), 2. *Arbitrary* (simultaneously write, but may not the same value), however, only one will succeed, and 3. *Priority* (priority order imposed for writing).

Relevance of the PRAM model to parallel computing

The answer of relevance of PRAM model depends on what we expect from the PRAM model or, more generally, how we understand the role of theory. When we strive to design an algorithm for solving a problem Π on PRAM, our efforts may not end up with a practical algorithm, ready for solving Π. However, the design may reveal something inherent to Π, namely, that Π is parallelizable. In other words, the design may detect in Π subproblems some of which could, at least in principle, be solved in parallel. In this case it usually proves that such subproblems are indeed solvable in parallel on the most liberal (and unrealistic) PRAM, the CRCW-PRAM.

We can replace CRCW-PRAM by the realistic EREW-PRAM and solve Π on the latter, all of that at the cost of a limited degradation in the speed of solving Π. In sum, the relevance of PRAM is reflected in the following method:

1. Design a program P for solving Π on the model CRCW-PRAM(p), where p (number of processor units) may depend on the problem Π . Note that the design of P for CRCW-PRAM is expected to be easier than the design for EREW-PRAM, simply because CRCW-PRAM has no simultaneous-access restrictions to be taken into account.
2. Run P on EREW-PRAM(p), which is assumed to be able to simulate simultane-ous accesses to the same location.
3. Guarantee that the parallel execution time of P on EREW-PRAM(p) is at most $O(\log\ p)$-times higher than it would be on the less realistic CRCW-PRAM(p).

3.1.2 The Local-Memory Machine

The LMM model has p processing units, each with its own local memory (Fig. 3.2). The processing units are connected to a common interconnection network. Each processing unit

can access its own local memory directly. In contrast, it can access a non-local memory (i.e., local memory of another processing unit) only by sending a memory request through the interconnection network.

The assumption is that all local operations, including accessing the local memory, take unit time. In contrast, the time required to access a non-local memory depends on:-

1. the capability of the interconnection network, and
2. the pattern of coincident non-local memory accesses of other processing units as the accesses may congest the interconnection network.

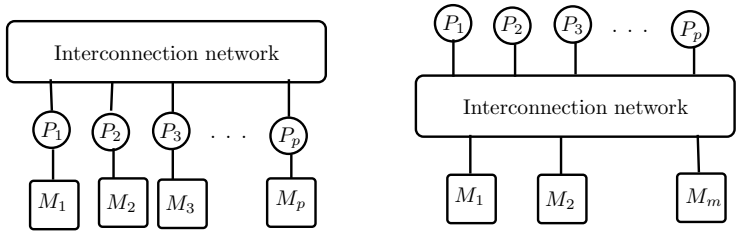

Figure 3.2: (a) LMM model, each machine with its own memory, (b) MMM model with sharable memories

Fig. 3.2(a) shows the LMM model of parallel computation has p processing units each with its local memory. Each processing unit directly accesses its local memory and can access other processing unit's local memory via the interconnection network.

3.1.3 Modular Memory machine

The Fig. 3.2(b) shows the MMM model of parallel computation with p processing units and m memory modules. Each processing unit can access any memory module via the interconnection network.

There are no local memories to the processing units.

3.2 Communication in Parallel Processing

We have seen that both LMM model and MMM model explicitly use interconnection networks to convey memory requests to the non-local memories (see Figs. 3.2(a) and 3.2(b)). In this section we focus on the role of an interconnection network in a multiprocessor model and its impact on the the parallel time complexity of parallel algorithms.

Interconnection Networks

Since the start of parallel computing, the major components of a parallel system have been the type of the central processing unit (CPU) and the interconnection network. This is

now changing. Recent experiments have shown that execution times of most real world parallel applications are becoming more and more dependent on the communication time rather than on the calculation time. So, as the number of cooperating processing units or computers increases, the performance of interconnection networks is becoming more important than the performance of the processing unit. Specifically, the interconnection network has great impact on the efficiency and scalability of a parallel computer on most real world parallel applications. In other words, high performance of an interconnection network may ultimately reflect in higher speedups, because such an interconnection network can shorten the overall parallel execution time as well as increase the number of processing units that can be efficiently exploited.

The performance of an interconnection network depends on several factors. Three of the most important are the *routing*, the *flow-control algorithms*, and the *network topology*.

Definition 3.1 *Routing. Here routing is the process of selecting a path for traffic in an interconnection network.* □

Definition 3.2 *Flow-control. Flow control is the process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver.*□

Definition 3.3 *Network-topology. Network topology is the arrangement of the various elements, such as communication nodes and channels, of an interconnection network.* □

For the routing and flow-control algorithms efficient techniques are already known and used. In contrast, network topologies have not been adjusting to changes in technological trends as promptly as the routing and flow-control algorithms. This is one reason that many network topologies which were discovered soon after the parallel computing came into existence are still being widely used. Another reason is the freedom that end users have when they are choosing the appropriate network topology for the anticipated usage. However, the end users have no option available now in choosing or altering routing or flow-control algorithms. As a consequence, a further step in performance increase can be expected to come from the improvements in the topology of interconnection networks. For example, such improvements should enable interconnection networks to dynamically adapt to the current application in some optimal way.

Self Review Questions

1. How a RAM model is different from conventional von Newmann machine model?
2. How a PRAM model is different from a von Newmann machine model?
3. The shared memory model comprises interconnection network between processor and memory (True/False)?
4. In shared memory system, the memory words can be of different sizes in different memory locations (True/False)?
5. How the simultaneous access to same location is protected in PRAM machine?
6. Who will be responsible for the read-write control program in EREW-PRAM and in CREW-PRAM machines?

(a) Application programmar
(b) System programmar
(c) Operating system
(d) Built-in the hardware

7. What are the three variants of PRAM models with respect to rad/write operations in memory?
8. How the consistent, arbitray, and priority models of CRCW-PRAM differ from each other?
9. Performance of the Interconnection-Networks depend on what major factors?
10. Explain the symmetric and regular interconnection networks.
11. What are the criterias that make an interconnection network scalable?
12. How many crossbar switches are there in a 100×100 connected crossbar switch that connects 100 processors to 100 memory modules?

Exercises

1. Design of which of the following is most difficult and costly out of the following? Justify your answer.

   (a) EREW-PRAM
   (b) CREW-PRAM
   (c) CRCW-PRAM

2. Explain with diagram a general communication network using graph. Explain, how you will compute the *diameter* of this graph? Also, give the logical steps (algorithm) to computer the diameter of such graph (network).