Chapter 12

Introduction to Parallel Algorithms

Preview

- Overview
- Lecture Notes
  - Parallel Architectures
    - Control Mechanism
    - Address-Space Organization
    - Shared-Address-Space Architecture
    - Message-Passing Architecture
    - Interconnection Networks
    - Static Interconnection Networks
    - Dynamic Interconnection Networks
  - The PRAM Model
    - Designing Algorithms for the CREW PRAM Model
    - Finding the Largest Key in an Array
    - Applications of Dynamic Programming
    - Parallel Sorting
    - Designing Algorithms for the CRCW PRAM Model
- Quick Check
- Classroom Discussion
- Homework
- Keywords
- Important Links

Overview

A computer can often complete a job faster if it has many processors executing instructions in parallel. A **processor** is a hardware component that processes instructions and date. All of the algorithms presented previously have been designed to be implemented on a traditional sequential computer. These computers are designed to take a single set of instructions and operate on a single sequence of data. Such a computer is called a **single instruction, single data stream (SISD)** computer, or *serial* computer.

Computers that operate in **parallel** have multiple processors that can each operate simultaneously with all the other processors. My connecting microprocessors, it is possible to obtain computing power faster than the fastest serial computer for substantially less money. Applications that can benefit from parallel processing include database query processing, weather prediction, and analysis of protein structures.

Lecture Notes

Parallel Architectures

The construction of parallel computers can vary in each of the following three ways:

1. Control mechanism
2. Address-space organization
3. Interconnection network

Control Mechanism

Each processor in a parallel computer can operate either under the control of a centralized control unit [**single instruction stream, multiple data stream (SIMD)**] or independently under the control of its own control unit. Figure 12.3 (a) illustrates an SIMD architecture. The interconnection network represents the hardware that enables the processors to communicate with each other. In an SIMD architecture, the same instruction is executed synchronously by all processing units under the control of the central control unit. Not all the processors must execute an instruction in each cycle. Any given processor can be switched off in any given cycle.

Parallel computers are called **multiple instruction stream, multiple data stream (MIMD)** computers. Figure 12.3 (b) is an illustration of an MIMD. MIMD computers store both the operating system and the program at each processor.

SMID computers are suited for programs in which the same set of instructions is executed on different elements of the data set. Such programs are called **data parallel programs**. One problem with SIMD computers is that they cannot execute different instructions in the same cycle.

Any processor that finds that $x \neq y$ must do nothing while the processors find that $x = y$ are executing instructions *A*. Those that find that $x = y$ must be idle while the others are executing instructions *B*.

In general, SIMS computers are best suited to parallel algorithms that require synchronization. Many MIMD computers have extra software that provides synchronization, which means that they can emulate SIMD computers.

Address-Space Organization

Processors can communicate with each other either by modifying data in a common address space or by passing messages. The address space is organized differently according to the communication method used.

Shared-Address-Space Architecture

In a **shared-address-space architecture**, the hardware provides for read and write access by all processors to a shared address space. Processors communicate by modifying data in the shared address space. Figure 12.4 (a) depicts a shared-address-space architecture in which the time it takes each processor to access any word in memory is the same. Such a computer is called a **uniform memory access (UMA)** computer. In a UMA computer, each processor may have its own private memory. The private memory is used to hold local variables necessary for the processor to carry out its computations. None of the actual input to the algorithm is in the private area. One problem with the UMA computer is that the interconnection network must simultaneously provide access for every processor to the shared memory.

One way to address this problem is to provide each processor with a portion of the shared memory. Each processor has access to the memory stored at another processor. However, it has faster access to its own memory than to memory stored at another processor. Such a computer is called a **nonuniform memory access (NUMA)** computer

Message-Passing Architecture

In **message-passing architecture**, each processor has its own private memory that is accessible only to that processor. Processors communicate by passing messages to other processors rather than modifying data elements. In the case of the NUMA computer, the interconnection network is wired to allow each processor access to the memory stored at the other processors. In the message-passing computer, the interconnection network is wired to allow each processor to send a message directly to each of the other processors.

Interconnection Networks

There are two general categories of interconnection networks: *static* and *dynamic*. Static networks are used to construct message-passing archtectures, whereas dynamic networks are used to construct shared-address-space architectures.

Static Interconnection Networks

A **static interconnection network** contains direct links between processors and are sometimes called **direct networks**. The most efficient, and the most costly, of these is the **completely connected network**. In such a network, every processor is linked to every other processor. The number of links is quadratic in terms of the number of processors, so this type of network is very costly.

In a **star-connected network**, one processor acts as the central processor. Every other processor has a link to that processor.

In a **bounded-degree network** of degree *d*, each processor is linked to *d* other processors. A message can be passed along by sending it first on one direction and then in another direction until it reaches its destination.

A slightly more complex, but popular static network is the hypercube. A zero-dimensional hypercube consists of a single processor. A one-dimensional hypercube is formed by linking the processors in two

zero-dimensional hypercubes. A two-dimensional hypercube is formed by linking each processor in a one-dimensional hypercube to one processor in another one-dimensional hypercube.

Recursively, a **(*d* +1) – dimensional hypercube** is formed by linking each processor in a *d* – dimensional hypercube to one processor in another *d* – dimensional hypercube.

Dynamic Interconnection Networks

In a **dynamic interconnection network**, processors are connected to memory through a set of switching elements. One of the most straightforward ways to do this is to use a **crossbar switching network**. In such a network, *p* processors are connected to *m* memory banks using a grid of switching elements. This connection is called *dynamic* because the connection between a processor and a memory bank is made dynamically when a switch is closed. A crossbar switching network is nonblocking. The connection of one processor to a given memory bank does not block the connection of another processor to another memory bank.

Other dynamic networks include bus-based networks and multistage interconnection networks.

The PRAM Model

It would be useful to find a universal model for parallel computation. Any such model would have to be sufficiently general to capture the key features of a large class of parallel architectures. Algorithms designed according to such a model must execute efficiently on actual parallel computers. No such model is currently known, and it seems unlikely that one will be found.

The **parallel random access machine (PRAM)** computer has become widely used as a *theoretical* model for parallel machines. A PRAM computer consists of *p* processors, all of which have uniform access to a large shared memory. Processors share a common clock, but may execute different instructions in each cycle.

The PRSM model is a natural extension of the serial model of computation. This makes the PRAM model conceptually easy to work with when developing algorithms. Algorithms developed for the PRAM model can be translated into algorithms for many of the more practical computers.

In the shared- memory computer such as a PRAM, more than one processor can try to read from or write two the same memory location simultaneously. There are four versions of the PRAM model:

- **Exclusive-read, exclusive-write (EREW) –** no concurrent reads or writes are allowed. That is, only one processor can access a given memory location at a given time. This is the weakest version of the PRAM computer, because it allows minimal concurrency.
- **Exclusive-read, concurrent-write (ERCW)** – simultaneous write operations are allowed, but not simultaneous read operations.
- **Concurrent-read, exclusive-write (CREW) –** simultaneous read operations are allowed, but not simultaneous write operations.
- **Concurrent-read, concurrent-write (CRCW) –** both simultaneous read and write operations are allowed.

Although programming languages for parallel algorithms exist, we will use our standard pseudocode with some additional features. Just one version of the algorithm is written, and after it is written, it is executed on all processors simultaneously. Each processor needs to know its own index while executing the algorithm. The processors are indexed P_1, P_2, P_3, etc. A variable declared in the algorithm could be a variable in shared memory, or it could be in private memory. In the latter case, each processor has its own copy of the variable.

All of our algorithms will be data-parallel algorithms. The processors will execute the same set of instructions on different elements of the data set. The data set will be stored in shared memory. If an

instruction shares a value of an element of this data set to a local variable, we call this a *read from* shared memory. If it assigns the value of a local variable to an element of this data set, we call this a *write to* shared memory. We will allow direct comparisons to variables like *n*, the size of the data set. All processors that read during a given step read at the same time, and all processors that write during a given step write at the same time. We assume that as many processors as we want are always available to us. This is an unrealistic assumption in the real world.

Designing Algorithms for the CREW PRAM Model

We illustrate CREW PRAM algorithms with the problem of Finding the Largest Key in an array. It takes at least n - 1 comparisons to find the largest key only by comparisons of keys, which means that any algorithm for the problem, designed to run on a serial computer, must be $\Upsilon(n)$. Using parallel computation we can improve on this running time. The parallel algorithm still must do at least $n - 1$ comparisons. By doing many of the comparisons in parallel, the algorithm will finish sooner.

This algorithm cannot benefit from using more processors because the result of each iteration of the loop is needed for the next iteration. Figure 12.10 illustrates how a parallel algorithm for this method proceeds. We need only half as many processors as array elements. Each processor reads two array elements into local variables *first* and *second*. It then writes the larger of *first* and *second* into the first of the array slots from which it has read. After three such rounds, the largest keys end up in S [1].

Alternatively, we can simply allow all the processors to execute in each step. The ones that need not execute simply do useless comparisons. The important things are that all the processors that should be executing are executing, and that the other processors are not changing the values of memory locations needed by the ones that should be executing.

When analyzing a parallel algorithm, we do not analyze the total amount of the work done by the algorithm. Rather, we analyze the total amount of work done by any one processor.

Applications of Dynamic Programming

Many dynamic programming algorithms are amenable to parallel design because often the entries in a given row or diagonal can all be computed simultaneously. We illustrate this approach by rewriting the algorithm for the binomial coefficient (Algorithm 3.2) as a parallel algorithm. Refer to the text for a more detailed discussion.

Parallel Sorting

Recall the dynamic programming version of Mergesort 3 (Algorithm 7.3). That algorithm simply starts with the keys as singletons, merges pairs of keys into sorted lists containing two keys, merges pairs of those lists into sorted lists containing four keys, and so on.

We can do the merging at each step in parallel. Algorithm 12.3 illustrates this method. Refer to the text for a more detailed discussion.

Designing Algorithms for the CRCW PRAM Model

CRCW stands for concurrent-read, concurrent-write. Unlike concurrent reads, concurrent writes must somehow be resolved when two processors try to write to the same memory location in the same step. The most frequently used protocols for resolving such conflicts are as follows:

- **Common –** allows concurrent writes only if all the processors are attempting to write the same values.
- **Arbitrary –** picks an arbitrary processor as the one allowed to write to the memory location.

- **Priority –** all the processors are organized in a predefined priority list, and only the one with the highest priority is allowed to write.
- **Sum –** writes the sum of the quantities being written by the processors.

We write an algorithm for finding the largest key in an array that works with *common-write*, *arbitrary-write*, and *priority-write* protocols and that is faster than the one given previously from the CREW model (Algorithm 12.1). Let the n keys be in the array S in the shared memory. We maintain a second array T of n integers in shared memory and initialize all elements in T to 1. Next we assume that we have n $(n - 1)/2$ processors indexed as follows:

$$Pij \quad \text{where} \quad 1 \le i < j \le n.$$

In parallel, we have all the processors compare $S[i]$ with $S[j]$. In this way, every element in S is compared with every other element in S. Each processor writes a 0 into $T[i]$ if $S[i]$ loses the comparison and a 0 into $T[j]$ if $S[ij]$ loses. Only the largest key never loses a comparison. Only the element of T that remains equal to 1 is the one that is indexed by k such that $S[k]$ contains the largest key. Figure 12.11 illustrates the steps. Refer to the text for a more detailed discussion of the algorithm. The algorithm works with common-write, arbitrary-write, and priority-write protocols.

Quick Check

1. In addition to Address space organization and Interconnection network, ____ is one of the ways by which the construction of parallel computers can vary
   Answer: Control mechanism
2. In a message-passing architecture, each processor has its own private memory that is accessible only to that processor. (True or False)
   Answer: True
3. In a(n) ____ network of degree d, each processor is linked to at most d other processors.
   Answer: bounded-degree
4. The weakest version of the PRAM computer, which allows minimal concurrency is called, ____.
   Answer: Exclusive-read, exclusive-write (EREW)
5. There are four versions of the PRAM model depending on how concurrent memory accesses are handled. (True or false)
   Answer: True

Classroom Discussion

- Discuss a PRAM algorithm for the Mergesort.

Homework

Assign Exercises 1, 4 and 16

Keywords

- **Bounded-degree network –** a network of degree d, each processor is linked to at most d other processors.
- **Completely connected network** – a network in which every processor is directly linked to every other processor.
- **Concurrent-read, concurrent-write (CRCW)** – both simultaneous read and write operations are allowed.

- **Concurrent-read, exclusive-write (CREW)** – simultaneous read operations are allowed, but not simultaneous write operations.
- **Crossbar switching network** – there is a switch at every position on the grid.
- **(d + 1) – dimensional hypercube –** formed by linking each processor in a d-dimensional hypercube to one processor in another d dimensional hypercube.
- **Data parallel programs** – programs in which the same set of instructions is executed on different elements of a data set.
- **Direct networks** – also called static interconnection networks. They contain direct links between processors.
- **Dynamic interconnection network** – processors are connected to memory through a set of switching elements.
- **Exclusive-read, concurrent-write (ERCW)** – simultaneous write operations are allowed, but not simultaneous read operations.
- **Exclusive-read, exclusive-write (EREW)** – no concurrent reads or writes are allowed. Only one processor can access a given memory location at a given time.
- **Message-passing architecture –** an architecture in which each processor has its own private memory that is accessible only to that processor.
- **Multiple Instruction Stream, Multiple Data Stream (MIMD) –** parallel computers, in which each processor has its own control unit
- **Nonuniform memory access (NUMA) –** a computer that has access to the memory stored at another processor.
- **Parallel computer –** computers are called parallel because each processor can execute instructions simultaneously (in parallel) with all the other processors.
- **Parallel Random Access Machine (PRAM)** – a computer that consists of p processors, all of which have uniform access to a large shared memory.
- **Processor –** in a computer is a hardware component that processes instructions and data.
- **Shared-address-space architecture –** provides for read and write access by all processors to a shared address space.
- **Single Instruction Data Stream (SISD) –** a computer that takes a single sequence of instructions and operates on a single sequence of data.
- **Single Instruction Stream, Multiple Data Stream (SIMD)) –** each processor in a parallel computer can operate either under the control of a centralized control unit or independently under the control of its own control unit.
- **Star-connected network –** a network in which one processor acts as the central processor.
- **Static interconnection network –** a network that contains direct links between processors also **direct networks**.
- **Uniform memory access (UMA) computer** – a computer with a shared-address-space architecture in which the time it takes each processor to access any word in memory is the same.

Important Links

- http://www.nist.gov/dads/ (Dictionary of Algorithms and Data Structures)
- http://www.sciencedirect.com (Journal of Algorithms)
- http://www.cs.cmu.edu/~scandal/nesl/algorithms.html (A Library of Parallel Algorithms)