Lecture 20: Multicomputers III, IV (Apr 19 and 21, 2005) Yap

Apr 19 and 21, 2005

1 ADMIN

- TODAY's Lecture: We have discussed "multiprocessor systems". This is the first of the 3 categories of parallel computers. Today we discuss the remaining two categories: "multicomputer systems" and "distributed systems".

2 Review

- Q: Draw the Omega network for a 16×16 node system.

  A: Starting from a column of 16 nodes, draw 4 consecutive stages of the shuffle exchange connection.

- Q: Give a basic tradeoff when implementing mutual exclusion for multiprocesses. Discuss.

  A: Should the process wait for the mutex variable or should we switch to a new process?

- Q: We want to keep increasing the processing power of computers (in terms of number of operations per second). But we run into fundamental physical limits (like speed of light). What is one avenue of computer design which avoids such limits?

  A: Parallel Processing.

- Q: Give a proof of correctness of the routing method in Omega-network.

  A: See previous lecture.

3 More General Remarks

Before we delve into multicomputers, let us make various general remarks.

- We have emphasized that classification of parallel computers is a fairly difficult one because of the many parameters involved.

  There is a famous taxonomy of parallel computers from Mike Flynn (1972) (this is not discussed in the text, unfortunately, but you should know it):

  The basis of Flynn's taxonomy is the distinction between **instruction stream** and **data stream**. In uniprocessor machines, you have one stream of each. In parallel computers, you could have many of them. This gives rise to the four basic classes:

  1. SISD (single instruction, single data) Computer
     This is just a uniprocessor computer.
  2. SIMD (single instruction, multiple data) Computer
     Of course, to have parallelism, you need multiple processing units (PU's). But one of these is the brain (denoted K), and it issues the (same) instructions to the remaining PU's.
     The vector machine is usually cited as an SIMD computer, although some people think of it as SISD. The Cray T-90 machine is an example.
     The ILLIAC-IV is the first machine of this class.
  3. MISD (multiple instruction, single data) Computer
     There are few commercial examples of this. The **systolic array** is an example.
  4. MIMD (multiple instruction, multiple data) Computer
     What we call multicomputers is definitely in this class.

- Measuring Parallel Performance

  – The basic parameters are number of processes p, the size of a problem n and parallel time T.

  – We assume p and n are independent parameters.

  – Thus, $T(p, n)$ is the time to solve problem size n with p processors.

  – We say the speedup is $S(n, p) = T(1, n)/T(p, n)$

  – The efficiency is $S(n, p)/p$ (ideal is 1)

- The theoretical ideal for a multiprocessor system is called a PRAM. It is not achievable in practice.

  – assume that the cost to access any memory location from any processor is unit time

  – what happens if several PU's tries to read/write a location?

  –We get four variants: EREW, CRCW, CREW, ERCW

- **Abstract Programming Models** for Parallel computation

  NOTE that we consistently held on to the fiction of what constitute a uniprocessor computer (we already noted that they are all parallel computers in modern architecture).

  This fiction is really a "programming model". Because of this fiction, we do not have to rewrite our programs each time Intel builds a new architecture.

  The same phenomenon will be important in parallel computing. Historically, programming models follow closely the architecture. This has tremendous problems (the life span of parallel architectures is remarkably short lived!)

  In practice, the "programming model" is captured by the "programming language" and its semantics.

4 Multicomputers

- Multicomputers, as the name suggests, involves several independent computers trying to cooperate in solving a common task.

  – Also called: cluster computers, clusters of workstations (COWS)

- Comparing "multicomputers" and "multiprocessor systems"

Feature	Multiprocessors	Multicomputers
Memory	Global	Local
Communication	Shared variables	Message passing
Synchronization	Mutex, semaphores, etc	Protocols
OS	One, uniform	Many (not necessarily same type)
Hardware	CPU-memory unit network	Computer-computer network

- CONTRAST BETWEEN SHARED VARIABLES AND MESSAGE PASSING:

  –In shared variables, we have the usuall LOAD(VAR) and STORE(VAR)

  –This can be refined to LOAD(MemoryUnit,VAR) STORE(MemoryUnit,VAR)

  –What is message passing? The idea is capture by its use of two instructions:

  SEND(destination,data)

  RECEIVE(source,data

  –Note that we address CPU's, not memory, in message passing.

- Computer-Computer Interconnection Network – TOPOLOGY

  Topology Parameters:

  p=number of processes

  n=parameter of the topology

  connectivity,

  diameter,

  bisection width (smallest number of wires to cut to divide network into two equal halves)

  1. array or ring
     –connectivity: constant (2)
     –diameter: p
  2. grid or double torus
     –connectivity: constant (4)
     –diameter: $\sqrt{p}$
     – E.g., Intel Paragon (2D torus), Cray T3D (3D torus)
  3. hypercube
     –connectivity: linear ($\lg p$)
     –diameter: $\lg p$
     –number each node by a string of $k = \lg n$ bits.
     –Gray code routing
     (a) CLAIM: we can number the nodes of a n-cube from 0 to $2^n - 1$ so that two integers have Hamming distance 1 iff they are connected by an edge on the hypercube. [Proof by induction]
     (b) Generalization: The hamming distance gives the distance between two nodes in a hypercube.
     (c) Let the **level** of an edge in the hypercube be i if the ith bit of the two integers differ.
     (d) Hence, out of each vertex, we have n edges, with levels 1 to n.
     (e) ROUTING: Suppose you want to go from source A to destination B. You first compute C=XOR(A,B).
     Starting with C at the source, repeat:
     If C is all 0's we are done (we are already at the destination.
     Else, take ANY non-zero bit of C. If this is the i-th bit, take the level i link to the next node. Also, zero the ith bit of C.
     (f) It follows that if the distance from A to B is k, there are 2^k different paths of length k between them!
     (g) Many other topologies can be embedded in the hypercube.
     (h) Let us consider the problem of embedding the 2^n-ring into the n-cube. This is given by the Gray code.

(i) An n-bit **Gray Code** is a permutation of the integers 0 to $2^n - 1$ such that adjacent integers in the list (and also the first and last in the list) has Hamming distance 1.

(j) Let $G(1) = [0, 1]$ be the Gray Code for $n = 1$.

(k) Inductively, let $G(n) = [g(0), g(1), \ldots, g(2^n - 1)]$.

(l) Then

$$G(n+1) = [0g(0), 0g(1), \ldots, 0g(2^n-1, 1g(2^n-1), 1g(2^n-2), \ldots, 1g(0)].$$

[Correctness is clear]

(m) We can symbolically write this as

$$G(n+1) = 0G(n); 1G'(n)$$

where $G'(n)$ is the reverse of $G(n)$.

(n) E.g., $G(2) = [00, 01, 11, 10]$. $G(3) = [000, 001, 011, 010, 110, 111, 101, 100]$.

(o) Consider 2 Gray codes to be equivalent if one is a cyclic shift of the other. Let $g(n)$ be the number of non-equivalent Gray codes. Thus $g(n) = 1$.

(p) For each $G(n)$, we get two inequivalent Gray codes, $G(n+1) = [0G(n), 1G'(n)]$ and $G(n+1) = [1G(n), 0G'(n)]$. These are inequivalent because ? There are no others because? Hence $g(n+1) = 2g(n)$. Thus $g(2) = 2$ and $g(3) = 4$.

4. Trees

–assume complete binary trees

–assume PE's at the leaves only

–connectivity: constant (3)

–diameter: p

–E.g., CM-5 machine

– FAT TREES: To avoid bottleneck at root, allow child-parent bandwidth at depth k to be half of the bandwidth at depth $k - 1$.

– Define the capacity of a link to be the number of pairs of leaves that has to communicate through the link.

– Links nearer the root has greater capacity.

– FAT TREES: the capacity at the level i is half the capacity at the next level below.

5. n-to-n connection

–crossbar

–omega network (E.g., NYU Ultracomputer, BBN Butterfly)

6. arbitrary topology (internet)

– Typically, planar graphs

– Connectivity: linear p but expected constant $O(1)$

– Diameter: p worst case

We can further generalize this to allow dynamic topology.

- ROUTING:

  Given a certain topology, how do we route messages?

  Message passing in arbitrary topology:

  – packet switching:

  A message is broken into **packets**, and injected into the network. Each packet finds a path to the destination, and these are reassembled at the destination.

  E.g., internet messages.

  – circuit switching

  First set up a path from source to destination, then the entire message (or session) is routed along this path.

  E.g., Telephony.

  – packet switching requires local memory at switches (e.g.,

- REMOTE PROCEDURE CALLS

  – Tanenbaum [p.537] points out that the message passing was the DOMINANT paradigm for communication in multicomputers.

  – But Birrell and Nelson (1984) introduced something rather different into the landscape with the idea of RPC

  – Client (Callee): the process calling the RPC; Server: the process receiving the RPC

  – Mechanics:

    1. Client program calls the CLIENT STUB in the local kernel (local process call)
    2. Client process is suspended
    3. CLIENT STUB packages the parameters, etc and send message to server machine (kernel).
    4. The SERVER STUB on server machine calls the server program.
    5. The return path is the reverse. The Client process is reactivated.

  – ADVANTAGES:

  procedure calls is well-understood high-level language mechanism

  makes distributed computing easy to use

  – Issues:

    1. pointer parameters,
    2. impossible with weak type information (e.g., vectors with unknown length, printf statements)

3. global variables

-