CSCE 5160 Parallel Processing

Homework #1: 2.2, 2.3, 2.12, 2.13
(due Jan 28 one week from the day of assignment)

Review
- PRAM model: how is shared memory accessed by multiple processors
- Note: in most cases we use threads, processors, processes or cores synonymously

Communication Networks
- Diameter
- Bisection width
- Cost

Translating PRAM model into networked systems
- Memory accesses require communication

An aside: note PRAM model assumes each memory access takes one cycle
- There are other models that distinguish between local and remote accesses

Abstract Parallel Processing Model – another view

Consider an EREW PRAM with \( p \) processors and \( m \) memory locations. Note PRAM does not account for communication (PRAM is a shared memory model).

We can emulate this model on a \( p \)-processor message-passing parallel computer in which each processor has \( \frac{m}{p} \) memory locations. Let \( t \) be the run time of an algorithm on a \( p \)-processor EREW PRAM model (that is, sequential). Give an upper bound on the run time of the algorithm for the following architectures.

a) a \( p \)-processor ring
b) a \( p \)-processor 2D mesh
c) a \( p \)-processor hypercube

Some assumptions. Since we are starting with a PRAM model (shared memory model), we will assume that the run-time of \( t \) consists of only memory accesses.

Since we started with EREW model, we will simply assume that all memory accesses are read (it does not make any difference what type of access we have).
CSCE 5160 Parallel Processing

Also, since we are interested in an upper bound when the algorithm is implemented on a message-passing architecture, we will assume that each memory access by a processor is to a memory location at a remote node that is at the farthest distance from requesting processor.

Thus, each of the $t$ memory accesses will have to travel the maximum distance possible.

Of course this is very poor distribution of the data. Ideally, we should minimize the communication between two processors by distribution the data needed by a processor into its local memory.

a). When we are dealing with a ring, the maximum distance any message must travel is $\lfloor p/2 \rfloor$ (actually this should use the ceiling operator for $p/2$ but since I do not have the correct font, I will not show that).

So, the upper bound on the algorithm is $t^* (p/2)$

or in terms of complexity, $O(t^*p)$

b). When dealing with a 2-D mesh (with wraparound), the maximum distance a message must travel is $\sqrt{p}$, giving an upper bound for our algorithm as $O(t^* (\sqrt{p}))$.

c). For a hypercube, the maximum distance a message must travel is $\log_2 (p)$, giving us an upper bound on our algorithm of $t^* \log_2 (p)$, or a complexity of $O[t^* \log_2 (p)]$

How to implement concurrent write on shared memory systems
Arbitrary or priority
need a lock or some similar synchronization mechanism
We will comeback to this later

Cache Coherency
You can read this section (page 45-53)
We will spend a lot of time understanding these issues in architecture classes
We need to understand the cost of sending a message from source node to destination node:

- Initialization cost
- Cost per “hop”
- Size of the message (related to bandwidth)

Need to understand how to send one-one message, one-to-many, many-to-one:

- Two types of one to many: broadcast → same message received by everyone
  - Scatter → different message for different nodes
- Many to one:
  - Gather → collect messages from multiple nodes
  - Reduction → combine messages
  - Add values
  - Find minimum or maximum

Depending on the type of network, we can design efficient algorithms.

Will comeback to communication algorithms later.

Chapter 3 – how do we go about designing a parallel algorithm

- How do we decompose a sequential algorithm
- We need to try to minimize communication and sequential portions
- Amdahl’s law
- We need to achieve load balance
- We need to determine if we have data parallelism or task parallelism

**Identifying Functional and Data Parallelism**

Consider an example

- Add 1000 4-digit numbers. Each number is written on an index card
- You have 1000 students at your service to complete this problem
- Students are seated in 25 rows of tables and each table seats 40 students (columns)
- Each student can pass up to 4 cards to his/her neighbors
CSCE 5160 Parallel Processing

a). What is the fastest way of distributing the cards

Give 20 cards to each student at the two ends of each row
They distribute 19 to their neighbor in the same row, and so on
Takes 20 steps
One form of Scatter

Any other ideas?

b). What is the fastest way of accumulating subtotals from students?

Once cards are distributed, you can send your card (and accumulated sum) to your neighbors
We can accumulate sums along rows and then across columns
Or we can accumulate sums from data received from all 4 neighbors
One form of gather and reduction

Identifying Functional and Data Parallelism

Let us look at some examples from Chapter 3– first Matrix*vector product (or inner product)

We can decompose the problem into computing one y per task
(we will deal with granularity and other issues such as data distribution later)
Another example: database query is find all white or green 2001 Honda Civic cars.

Start with a relational database with the data shown:

<table>
<thead>
<tr>
<th>ID</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Dealer</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4523</td>
<td>Civic</td>
<td>2002</td>
<td>Blue</td>
<td>MN</td>
<td>$18,000</td>
</tr>
<tr>
<td>3476</td>
<td>Corolla</td>
<td>1999</td>
<td>White</td>
<td>IL</td>
<td>$15,000</td>
</tr>
<tr>
<td>7623</td>
<td>Camry</td>
<td>2001</td>
<td>Green</td>
<td>NY</td>
<td>$21,000</td>
</tr>
<tr>
<td>9034</td>
<td>Prius</td>
<td>2001</td>
<td>Green</td>
<td>CA</td>
<td>$18,000</td>
</tr>
<tr>
<td>6734</td>
<td>Civic</td>
<td>2001</td>
<td>White</td>
<td>OR</td>
<td>$17,000</td>
</tr>
<tr>
<td>5542</td>
<td>Acura</td>
<td>2001</td>
<td>Green</td>
<td>FL</td>
<td>$19,000</td>
</tr>
<tr>
<td>3645</td>
<td>Maxima</td>
<td>2001</td>
<td>Blue</td>
<td>NY</td>
<td>$22,000</td>
</tr>
<tr>
<td>8354</td>
<td>Accord</td>
<td>2000</td>
<td>Green</td>
<td>VT</td>
<td>$18,000</td>
</tr>
<tr>
<td>4395</td>
<td>Civic</td>
<td>2001</td>
<td>Red</td>
<td>CA</td>
<td>$17,000</td>
</tr>
<tr>
<td>7532</td>
<td>Civic</td>
<td>2002</td>
<td>Red</td>
<td>WA</td>
<td>$18,000</td>
</tr>
</tbody>
</table>

Table 3.1: A database storing information about used vehicles.

We can consider building intermediate results (or tables) as our decomposition.

E.g. Find all Civics
    Find all 2001 vehicles
    Find all white vehicles
    Find all green vehicles

then

Find all 2001 Civics
Find all white or green vehicles

finally

Find white or green 2001 Civics
Once we decide on a decomposition, we can build task graphs. Normally we want to start with as much parallelism as possible. Make tasks as fine grained as possible. For example, instead of all white or green cars in one task, we should have started with separate tasks in our decomposition.

CSCE 5160 Parallel Processing

Task interaction graphs

As seen before, task interactions (or synchronizations) exist even if we have a shared memory. These interactions may cause delays because of producer-consumer dependencies or mutual exclusion.

In the following example, each row of A and one element b are assigned to a task.

*Figure 3.5* Abstractions of the task graphs of Figures 3.2 and 3.3, respectively.

*Figure 3.6* A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task \( i \) computes \( \sum_{0 \leq j \leq 11, \text{all } j \neq i} A(i, j) \cdot b(j) \).
Characteristics of Tasks and Interactions

Task size
- May have a priori (static) knowledge of the size
- Sizes may be fixed
- Sizes may not be fixed and may not be known statically

Task data
- Need to know what data is needed by a task so that you can send the data
- Need to know the size of the data (both input and output)

Task Interactions

Static:
- We can define interaction graph statically
- Known tasks interact (like pipeline parallelism model)
- We can create barriers and wait

Dynamic:
- May not know
- (difficult in MPI to deal with such interactions)
- May overlap communication with computation

Regular interactions:
- Known tasks interact (like pipeline parallelism model)
- We can create barriers and wait

Irregular interactions:
- May need to use "polling" or simulate interrupt
- Difficult with MPI and OpenMP

Task Interactions

Nature of sharing
- Read only
- Read and write

- one way (one task provides all the data or receives all the data)
- like Scatter and Gather in MPI
- two way – producer/consumer type problems
Mapping tasks to processors

Once we have a task graph (and/or interaction graph) we can think of assigning different tasks to different processes (on the same processor or on different processors).

Size of tasks and interactions among the tasks
Costs associated with tasks
creating (and deleting) tasks
computation performed by tasks
do we know this a priori?
memory (data) requirements of tasks

Load balancing

Interactions
communication overhead
delays due to mutual exclusion

An example → two different mapping of tasks to processors.

Figure 3.7 Mappings of the task graphs of Figure 3.5 onto four processes.
More techniques for decomposing a problem for parallel processing

1. Recursive decomposition
e.g., quicksort (see page 96)
   Start with one task to divide the list into two (using a pivot)
   two tasks to divide each of the two lists into 2 (total 4 lists)
   etc.
Works if you started with a recursive sequential algorithm
Later we will see some difficulties in parallelizing recursive algorithms such as depth first search.

2. Data decomposition
   Assign task to deal with different parts of data either
   based on input
   e.g., Dot matrix tasks handle portions of input arrays
   based on output
   e.g., matrix multiplication, different tasks compute different output
   elements
   hybrid
   depending on intermediate data
   we will see examples later

3. Exploratory decomposition
   good for optimization problems that find a solution by searching the solution space.
   Most AI problems, planning problems fall in this category.
   Consider solving the 15 puzzle problem
   We start with a 4x4 grid with one blank square (and 15 filled squares)
   The problem is to move to a target configuration of the grid from a starting
   configuration.
   Note at any given configuration you have at most four possible moves
   leading to (up to) 4 new paths in the search space.
4. Speculative decomposition
   Start with tasks even before knowing if that part should be searched
   
   Parallel discrete event simulation (page 109)
   Time warp simulations
   Speculative threads (in hardware or software)

Critical path in a task graph

This indicates the minimum time to complete all tasks of a graph even if you have unlimited number of resources

We can associate weights with nodes (execution times of tasks) and weights with arcs (communication or "interaction" costs)

Critical path I, A, B, D, O = 95
Note here we are assuming 4 processors for maximum parallelism

What happens if we only have 3 processors? Still 95?
CSCE 5160 Parallel Processing

Consider the problem 3.19 from text on page 145

Bucket sort: we have n integers and each integer is in the range 1…r
Place each integer into one of \( r \) buckets

Decomposition based on input

Divide n inputs into \( p \) tasks and assign each task \( n/p \) numbers
All tasks perform the same operation: place numbers into the buckets

Decomposition based on output

\( r \) tasks, one per output bucket
Each task examine all \( n \) inputs, and discards numbers not belonging to the bucket

CSCE 5160 Parallel Processing

Another example  Problem 3.2 (page 143)
(i) maximum degree of concurrency
(ii) critical path
(iii) maximum speedup with "infinite" number of processors
(iv) number of processors needed in (iii)
(v) speedup if the number of processors is limited to 2, 4 and 8

(i) 8; (ii) 4; (iii) 15/4; (iv) 8
(v) 15/8; 3; 15/4
Another example

Problem 3.2 (page 143)

(i) 8; (ii) 8; (iii) 15/8; (iv) 2
(v) 15/8; 15/8; 15/8

Homework # 2: 3.4, 3.15, 3.16
Due Feb. 4, 2019