

CSCE 5610 Solutions To Midterm (Krishna Kavi)

Thursday, October 8, 2009; 3:30-5:00pm

1 (25%). Consider routing a message using store-and-forward. We have seen that the cost of sending a message of m words from a source to destination on a path of length d is given by $t_s + (m \cdot t_w) \cdot d$

Consider a variation. We will divide the message into k smaller messages, each of size m/k words. Derive the cost of communication on a path of length d , if

- a new message can be sent from a node as soon as the previous message is received by the next node in the path.
- a new message can only be sent after the original message is received by the final destination.

Key. For both parts, we have k messages and thus we have $k \cdot t_s$ as the startup costs. But for (a), we are sending smaller messages in a pipelined fashion, and thus we only have to wait for $(m/k) \cdot t_w$ at each node. For (b), we have to wait for each of the k messages to receive independently.

- $k \cdot t_s + (m/k) \cdot t_w \cdot d$
- $k(t_s + (m/k) \cdot t_w \cdot d) = k \cdot t_s + m \cdot t_w \cdot d$

In other words, dividing a large message into smaller one is beneficial if we can pipeline the message transmission.

2 (20%). By now you know how dot-product works.

```
for (i=0; i<n; i++) dot_product += a[i]*b[i];
```

Assume that you have p processors connected in a ring. Derive an expression for the execution time on p processors (including both communication costs and computation costs). Ignore t_s (set up time) and assume that t_w , multiplication and addition take the same amount of time t .

Key. Consider what we need for our algorithms

1. Scatter initial data to p processors. We will assume that a and b vectors are scattered to the p processors. The discussion in section 4.4. can be used for this part. If we ignore start up costs the lower bound for scatter is $m \cdot t_w \cdot (p-1)$ REGARDLESS OF THE INTERCONNECTION NETWORK.

In our case, we are dealing with $m = 2 \cdot n/p$ since we have to scatter both a and b vectors

2. Computation. Each processor completes n/p multiplications and n/p additions for a local sum. That is $2 \cdot n/p$ operations.

3. We need to perform a reduction of all the p partial results into a single dot product value. Following the discussion in section 4.2.4, the cost is given for a all to all broadcast (ignoring startup costs) is also given by $2 \cdot m \cdot t_w \cdot (p-1)$ and $m = 1$.
Total cost = $(2 \cdot n/p) \cdot t \cdot (p-1) + 2 \cdot (n/p) + 2 \cdot t \cdot (p-1) = 2 \cdot t \cdot (2p+n-2)$

Comments. Some of you did not include the initial data distribution (scatter). Some of you assuming that reduction of partial dot_product results can be achieved in $\log(p)$ operations. However, you still need to

send/received data among the “logical neighbors” as viewed in a tree like reduction. In a ring, these neighbors will be at a distance of 1, 2, 4, 8,... (doubling the distance as you move up the tree of reduction).

3 (15%). The distance between two nodes u and v is the length of the shortest path from u to v . Given a d -dimensional hypercube and a designated source node s , how many nodes are at distance i from s ; $0 \leq i \leq d$.

Key

Note u and v are at a distance i if the binary values of v and u differ in i bit positions. Since we have d bits for each u and v (d -dimensional hypercube), we are looking at all possible combinations where v and u differ in i out of d positions.

$$C(d,i) = (d!)/(i!)*(d-i)!$$

Comments. I know this is a “curve ball”, but it is actually one of the problems from the textbook

4 (15%). Consider the bucket sort problem we discussed in class. We take n numbers, and place them into r buckets (each bucket holding numbers in different value ranges; for example, 0..9; 10..19; 20-29;...).

Let us assume that the numbers are uniformly distributed (ie., you will place equal number of numbers in each bucket). What is the speed up that is possible with p processors, if we use data composition on input data? And what is the speedup you can achieve if you use data decomposition on output?

Key.

Distribution based on input

Each processor sorts n/p numbers into r buckets, taking $O(n/p)$

If we can sort each bucket in parallel (assuming $p > r$) sorting each bucket using quick sort we have $O((n/r)*\log(n/r))$

assuming that each bucket will contain n/r elements

Total parallel execution time = $O[(n/p) + (n/r)*\log(n/r)]$

Since we assumed p is greater than r , we can drop n/p

Speed up = $O(n \log n) / [(n/r)*\log(n/r)] = O[r * \log(n)/\log(n/r)]$

Distribution based on input

Each processor is assigned a bucket and each processor will examine all n input numbers and place them in the bucket assigned to it. $O(n)$.

Then we sort each bucket similar to above: $O((n/r)*\log(n/r))$

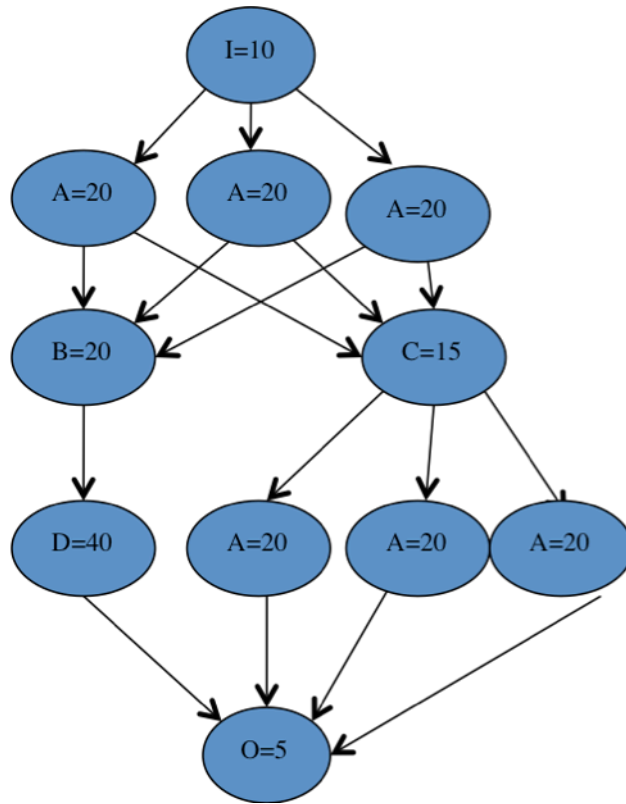
Total parallel execution time = $O[n + (n/r)*\log(n/r)]$

Speedup = $O[(n*\log n)/(n + (n/r)* \log(n/r))]$

Thus decomposition based on input is more effective under these assumptions

5 (25%). Consider the following task graph. The numbers inside the nodes represent execution times. Assuming that we ignore the cost of communication among tasks (represented by the arcs in the graph), find the minimum computation time possible if we use 2 identical processors. Show the tasks executed by

each processor. Now consider that it cost 10 units to communicate between processors (and all arcs have the same weight). What is the minimum amount of time needed to complete the task with 2 processors? Again show tasks assigned to each processor.



Key.

I suggested heuristics by dividing the nodes into levels and schedule tasks in a given level giving priority to tasks in the critical path.

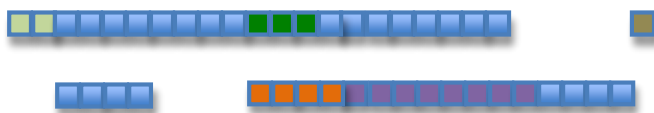
If we want load balance you may assign tasks as following. However, note that this will not lead to minimum execution time.

Processor 1 1: I, A, A, C, A, A, O ($10+20+20+15+20+20+5 = 110$)

Processor 2: A, B, D, A ($20+20+40+20 = 100$)

Processor 2 must wait for I to complete on processor 1, and Processor 1 must wait for Processor 2 to complete D and A

The following timeline shows the execution of tasks (I used different colors for different tasks). The top row is the execution time line for P1 and the second row is for P2; each square is 5 units.



If I counted right, we will see that we need 135 units of time to complete all tasks.

Also, this causes additional communications (part b of the problem).

Consider the communications we need)

From I to A (processor 1 to task 2)

From A's on Processor 1 to Processor 2 (before B can start)

From D/A on processor 2 to Processor 1 before O can be completed

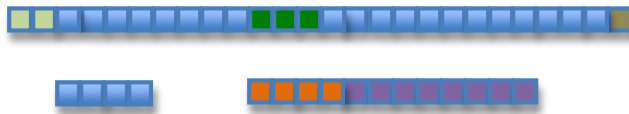
So, in addition to the execution time of 135 units, we have 30 additional units of communication costs.

Consider the following assignment

Processor 1: I, A, A, C, A, A, A, O ($10+20+20+15+20+20+20+5= 130$)

Processor 2: A, B, D ($20+20+40 = 80$)

Processor 1 is more heavily loaded. But see the execution time line (without communications). Again the top row is the time line for P1 and bottom row is for P2; each square is 5 units. Each task is assigned a different color.

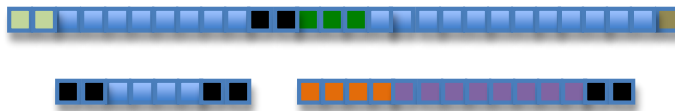


In this case the total execution completes in 130 units.

Also note we eliminated the need the last communication (from D to O) can be overlapped with computation of one of the 3 A tasks assigned to processor 1.

With communication. If we assume that communication and computation can proceed in parallel

Here is the timeline.



Note, the black squares represent communication. I am assuming that the sender cannot compute and communicate concurrently; but a receiver can receive while performing a computation. Under these assumptions, we can complete the task graph in 140 units of time as shown above.

If communication and computations cannot be overlapped, we need an additional 30 units of time (the black squares in P2 time line).