CPSC 5600/4600
Exam #1

Practice Exam

Solution

So that you may receive partial credit, remember to show all your work. **PRINT CLEARLY** – if it is unreadable, it is incorrect.

This test is closed-book and closed-notes except for a 8½ x 11" hand-written one-sided "cheat sheet". You are allowed the full class period to complete the exam, 2 hour and 5 minutes.

DO NOT comment extensively **OR** give elaborate I/O messages due to the time constraint of this test.

Question	Points	Score
I	20	
II	20	
III	20	
IV	20	
V	20	
Total	100	

I. Pair-Wise Product

1. Imagine a problem where we have a circuit with a series of "gates" where the probability of missing each gate is known. We want to calculate the probability, q, of a flow going through all the gates using a parallel pair-wise product reduction algorithm. The calculation is given by the formula:

$$q = \prod_{i=0}^{n-1}(1 - p_i)$$

[The capital pi notation, Π, is like the sigma summation notation, Σ, but using multiplication instead of addition.]

Assume you have the **Heaper** base class similar to that in HW2. Write a C++ class, **GateFlow**, that extends **Heaper**, takes a probability array as an input, and calculates the reduction. The **GateFlow** class starts as shown below. You must write the rest of the class. Assume that all necessary include files have already been included. The work should be parallelized fairly evenly over 16 threads, using the **std::async** function to create the threads.

```
typedef std::vector<double> Data;

class GateFlow : public Heaper<Data> {
public:
    GateFlow(Data *probabilities) : Heaper(probabilities) {
        calcProb(0, 0);  // node=0, level=0
    }
    double getProbability() {
        return value(0);
    }
private:
    void calcProb(int i, int level) {
        if (!isLeaf(i)) {
            if (++level < 4) {
                auto handle = async(launch::async, &GateFlow::calcProb,
                                    this, left(i), level);
                calcProb(right(i), level);
                handle.wait();
            } else {
                calcProb(left(i), level);
                calcProb(right(i), level);
            }
            interior->at(i) = value(left(i)) * value(right(i));
        }
    }
protected:
    virtual double value(int i) {
        if (i < n-1)
            return interior->at(i);
        else
            return 1.0 - data->at(i - (n-1));
    }
```

II. Prefix Product

1. Extend the above solution, **GateFlow**, to also calculate a scan version of the gate flow probabilities. It should also parallelize to 16 threads. Its public method in **GateFlow** is as follows:

```
void getProbabilities(Data *output) {
    calcScan(0, 0, 1.0, output); // node=0, level=0, priorProb=1.0, out
}
```

Write the private **calcScan** method:

```
void calcPrefix(int i, int level, double prodPrior, Data *output) {
    if (isLeaf(i)) {
        output->at(i - (n-1)) = prodPrior * value(i);
    } else {
        if (++level < 4) {
            auto handle = async(launch::async, &GateFlow::calcPrefix,
                                this, left(i), level, prodPrior,
                                output);
            calcPrefix(right(i), level, prodPrior * value(left(i)),
                       output);
            handle.wait();
        } else {
            calcPrefix(left(i), level, prodPrior, output);
            calcPrefix(right(i), level, prodPrior * value(left(i)),
                       output);
        }
    }
}
```

III. Bitonic Sequences

1. Which of the following arrays are bitonic sequences with a central vertex (what we get out of the bitonic_sequence method?
   a. **{1, 2, 3, 4, 5, 6, 7, 8, 100, 10, 11, 12, 13, 14, 15, 16}**
   b. ✔**{DOWN 10, 10, 9, 4, –6, –6, –7, –8 |UP 9, 10, 11, 12, 13, 14, 15, 16}**
   c. ✔**{UP 1, 2, 3, 3, 3, 3, 3, 3 |DOWN 3, 3, 3, 3, 3, 3, 3, –30}**
   d. ✔**{DOWN –1, –2, –3, –4, –5, –6, –7, –8 |UP 9, 10, 11, 12, 13, 14, 15, 16}**
   e. ✔**{DOWN 1, 1, 1, 1, 1, 1, 1, 0 |UP 7, 7, 7, 7, 7, 7, 7, 7}**
   f. **{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}**

2. Take the first one of the sequences above that you included in your answer and simulate the steps of a bitonic sort. Perform the steps in the same order as would be done in the bitonic network diagram we've used in the classroom, showing the intermediate results after each set of 8 comparators have run (i.e., after each "column" in the network diagram).

   {10, 10, 9, 4, –6, –6, –7, –8 | 9, 10, 11, 12, 13, 14, 15, 16}

   after first bitonic_merge (up), i.e., first column (compares eight apart):
   {9, 10, 9, 4 | –6, –6, –7, –8 | 10, 10, 11, 12 | 13, 14, 15, 16}

   after the two second bitonic_merges (up), i.e., second column (compares four apart):
   {–6, –6 | –7, –8 | 9, 10 | 9, 4 | 10, 10 | 11, 12 | 13, 14 | 15, 16}

   after the four third bitonic_merges (up), i.e., third column (compares two apart):
   {–7 | –8 | –6 | –6 | 9 | 4 | 9 | 10 | 10 | 10 | 11 | 12 | 13 | 14 | 15 | 16}

   after the eight fourth bitonic_merges (up), i.e., fourth column (compares one apart):
   {–8, –7, –6, –6, 4, 9, 9, 10, 10, 10, 11, 12, 13, 14, 15, 16}

3. Show pseudocode for the bitonic merge step (up) for an input of size n.

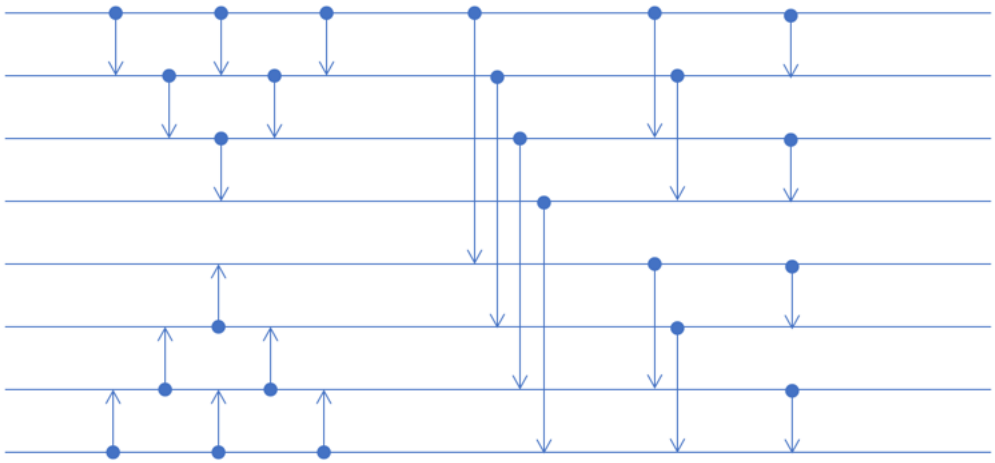
```
for (i = 0; i < n/2; i++)
        if (a[i] > a[i + n/2])
                swap(a[i], a[i + n/2]);
```

4. What properties does the array have after the merge step above?

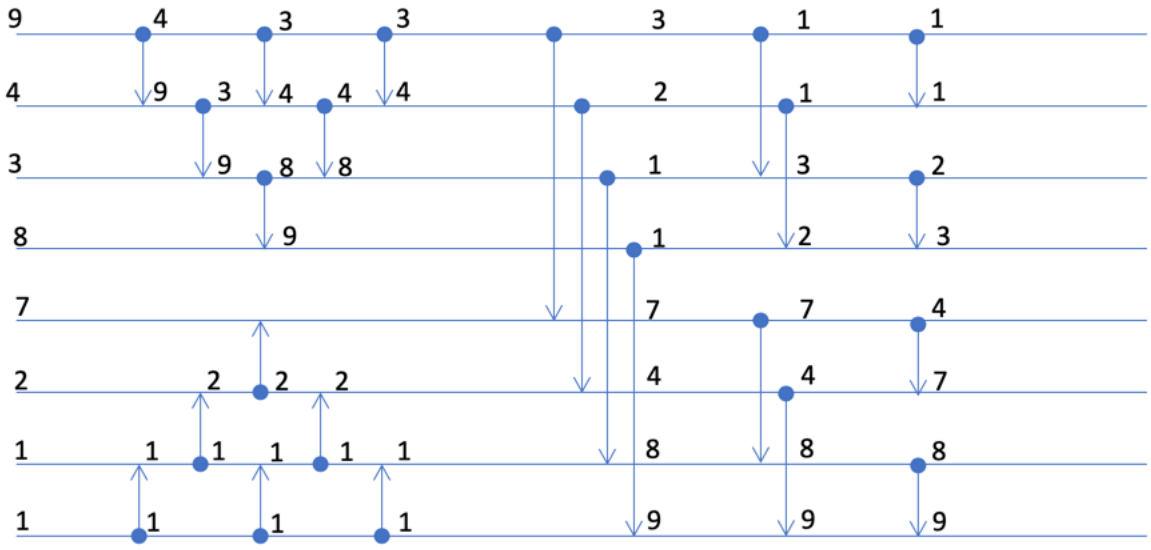
the first half is a bitonic sequence and the second half is also a bitonic sequence; the second half elements are all greater or equal to all the first half elements (i.e., partitioned in the middle)

IV. Bitonic Sorter Network

1. Assume we have the following 4-sorter component and a bitonic sorter component. Use only those components to build an 8-sorter network. Extend the diagram below to show the whole 8-input network. Show all the internal comparators of each component. You may use a mirror-image of a component if you like.



2. Simulate the following array through your network above:
{9, 4, 3, 8, 7, 2, 1, 1}



V. Analysis

1. What is the time-efficiency (big oh) for the amount of work done (basic operation is multiplication) by Problem I? Explain.

   There are n-1 multiplications done (complete binary tree interior nodes each have one multiplication and a complete binary tree has n-1 interior nodes if there are n leaves). So, $O(n)$.

2. What is the time-efficiency for the ideal latency of Problem I? Explain.

   The data dependencies are on the left and right child below, so we end up with a mandatory time step for each level of the tree. All the products for the same level in the tree can be calculated simultaneously, so no delay there in ideal parallelism. So, we end up with the number of time steps being the number of interior levels of the tree, i.e., $\log_2(n)$ time steps or $O(\log n)$ latency.

3. What is the time-efficiency for the amount of work done (basic operation is a comparator) by Problem IV? Explain.

   For this assume that $n = 2^k$ and a 2^{k-1}-input sorting network component is formed in the same manner as that shown for the 4-sorter component and that it is extended to a 2^k-sorter in the same manner that you did.

   The 4-sorter is doing bubble sort, which we know is $O(n^2)$ work and the bitonic sort stage does $O(n \log n)$. Since we are composing our network of two bubble sorts of size n/2 and one bitonic sort stage of n, we get:

   $2*O((n/2)^2)+O(n \log n) = O(n^2) + O(n \log n) = O(n^2)$

   Alternately, we could observe that the number of comparators in our preliminary sorter of size n going diagonal by diagonal would be $n + (n-1) + (n-2) + \ldots + 1$ which is $(n)(n+1)/2$ which is $O(n^2)$. Then on the bitonic sort component note that it has $\log(n)$ columns each with n/2 comparators, so $O(n \log n)$. Then proceed as above to get $\mathbf{O(n^2)}$.

4. What is the time-efficiency for the ideal latency of Problem IV? Explain. Use the same assumptions as above.

   One can see from the diagram that the comparisons that form the interior of the pyramid shape in the 4-sorter can be done concurrently with the comparators at the surface. But the pyramid surface has a whole line of dependencies all the way from one side of the base to the other. If we count those steps up and down the pyramid surface it will be n-1 going up and another n-2 going down for a total of 2n-3 timesteps, so $O(n)$ for those components. Our two mirror circuits there can also be done simultaneously, so for our whole network, those two initial parts require $2(n/2) - 3$ timesteps, i.e., $O(n)$. Then the bitonic sort component we know takes minimum one timestep for each column, for a total of $\log_2(n)$ timesteps, or $O(\log n)$. The total circuit then has $O(n)+O(\log n) = \mathbf{O(n)}$.