CS 351 Exam 4 — Tue. 12/10/2013

Rules and Hints

- You may use one handwritten 8.5 × 11" cheat sheet (front and back). This is the only additional resource you may consult during this exam. *No calculators.*
- Include step-by-step explanations and comments in your answers, and show as much of your work as possible, in order to maximize your partial credit.
- You may use the backs of these pages if you need more space, but make it clear where to find your answer to each question.
- You may write your answers in the form *[mathematical expression][units]*. There is no need to actually do the arithmetic.

Grade

	Your Score	**Max Score**
Problem 1: Reliability metrics etc.		10
Problem 2: Disk performance		10
Problem 3: Flash		10
Problem 4: RAID		20
Problem 5: Parallelizing code		10
Problem 6: Parallel performance metrics		15
Problem 7: Cache coherence		15
Problem 8: GPUs		10
Total		100

Problem 1: Reliability metrics etc. (10 points)

Part A (1 point)

+1 extra credit point if entire class gets it right.

Seriously, though: how many bits are in a byte?

Part B (9 points)

A particular web service is down for repair 36.5 days out of every year. Each outage lasts for approximately one day.

- What is its availability?

- What is its MTTF?

- What is its MTTR?

- What is its MTBF?

Problem 2: Disk performance (10 points)

Part A: Performance (7 points)

A certain hard disk has the following specifications:

- Average seek time: 8.9 ms
- Rotation speed: 7200 rpm
- Data transfer rate: 166 MB/sec

How long does this disk take to do a 1.5 MB transfer?

Part B: Energy (3 points)

The same disk has the following power requirements:

- Power during seek: 5.3 W
- Power while just spinning: 4.6 W
- Power during read/write: 5.8 W

How much *energy* will this disk use during the course of the 1.5 MB transfer?

Problem 3: Flash (10 points)

Part A: Organizing data (5 points)

In a log-structured filesystem, all data is written to a sequential log – think of this as a gigantic circular buffer. New data, no matter what file it's from, goes in the next spot in this log. When data is changed, it's not overwritten – the new version of the data is just written to the log and the old version is left in place.

To read a file, in theory, you would have to read through the entire storage device in sequence to find the most recent copy of each chunk of data. In practice, the locations of each file's data will be stored in a fast cache on the device, so you can go straight to the data itself.

Would a log-structured filesystem be a good fit or a bad fit for a flash (SSD) device? Explain.

Part B: Access types (5 points)

Which of the following access types is fastest on flash, and which is slowest?

- Random (small) reads
- Random (small) writes
- Large sequential reads
- Large sequential writes

Problem 4: RAID (20 points)

You have 7 TB of unique, non-redundant data. You are planning to store it on an array of disks that are 1 TB each.

Your workload has a mix of small requests, which take 12 ms on this disk, and large requests, which take 40 ms on this disk (12 ms of latency and 28 ms of transfer time).

Part A: RAID 1 (10 points)

Answer the following questions about constructing a RAID 1 array for your data from these disks:

- How many *total* disks will you need?
- If small accesses are evenly distributed across your disks, how many small reads can you complete per second on this array?
- If small accesses are evenly distributed across your disks, how many small writes can you complete per second on this array?
- How long will it take to do a single large read on this array?
- How long will it take to do a single large write on this array?

Part B: RAID 5 (10 points)

Answer the following questions about constructing a RAID 5 array for your data from these same disks:

- How many *total* disks will you need?

- If small accesses are evenly distributed across your disks, how many small reads can you complete per second on this array?

- If small accesses are evenly distributed across your disks, how many small writes can you complete per second on this array?

- How long will it take to do a single large read on this array?

- How long will it take to do a single large write on this array?

Problem 5: Parallelizing code (10 points)

Consider the following code. Each function in this code takes a long time to run.

```
A = f1(B, C, D)  // Line 1
E = f2(A, B)       // Line 2
F = f3(C, D)      // Line 3
G = f4(E, A)      // Line 4
```

Part A: 4 points

Draw a dependency graph with one node for each line.

Part B: 3 points

How many threads can this code take advantage of?

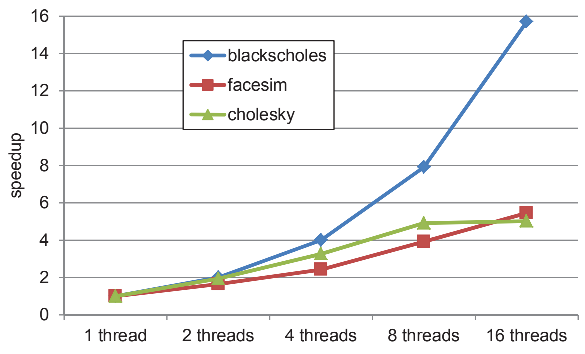
Part C: 3 points

If Line 1 takes t_1 units of time to run, Line 2 takes t_2 units, etc., then what is the speedup of a fully parallelized version of this code over a sequential version?

Problem 6: Parallel performance metrics (15 points)

The graph below shows the speedup vs. number of threads for three parallel applications: blackscholes (the line that goes highest), facesim (the squares), and cholesky (the triangles).

Please ask for help if anything on the graph is too small to read.



From Eyerman et al., "Speedup Stacks: Identifying Scaling Bottlenecks in Multi-Threaded Applications," in *Int'l. Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2012.

Part A (5 points)

Which, if any, of the three applications exhibit strong scalability? Explain your answer.

Part B (5 points)

What is the approximate efficiency of facesim and cholesky at 16 threads? Please show enough work to make clear how you arrived at your answer.

Part C (5 points)

This problem continues to refer to the graph on the previous page.

To be weakly scalable, a program must consist of two parts:

- A part that is inherently sequential and can never be parallelized
- A part that is parallelizable and will exhibit linear speedup in the number of threads

Assume that facesim is weakly scalable, and show how you would determine what fraction of the original program is inherently sequential.

You should set up the math, but you do not need to actually do any arithmetic.

Problem 7: Cache coherence (15 points)

You have a dual-core processor with private, write-back L1 caches, a shared L2, and a shared bus to the L2 cache.

The following references to some location X happen in order. There are no intervening references to location X.

1. P1 reads the value 2 from location X in L2 (L1 miss, L2 hit)
2. P2 writes the value 3 to location X (L1 miss, L2 hit)
3. P1 writes the value 5 to location X (L1 hit)
4. A significant of amount of time later, P2 reads location X (L1 hit)
5. P1 reads location X (L1 hit)

Part A (5 points)

If there is no hardware cache coherence mechanism, what values will P1, P2, and the L2 cache have for location X after this sequence of references is complete?

Part B (10 points)

Assuming an MSI (modified-shared-invalid) cache coherence protocol, track the state of location X's block in each processor's L1 cache using the table below. You do not need to write down what happens on the shared bus to L2.

Ref.	**P1 cache's state**	**P2 cache's state**
Initial state	Invalid	Invalid
After ref. 1		
After ref. 2		
After ref. 3		
After ref. 4		
After ref. 5		

Problem 8: GPUs (10 points)

Part A (5 points)

Write a snippet of C-like code that is well suited to being rewritten in CUDA and run on a GPU, and *explain why.*

Part B (3 points)

If you were to port this code to CUDA, how would you divide up the work among threads?

Part C (2 points)

If CUDA allows 1024 threads per block, how many threads and how many blocks would you instantiate?