CS 377P Fall 2023: Mid-term exam (1.5 hours)

October 19th, 2023

Name:

EID:

Problem	Score
1	/25
2	/20
3	/30
4	/25
Total	/100

1. *Architecture (25 points)*

   Modern processors exploit instruction-level parallelism (ILP) through out-of-order execution and in-order commit.

   (a) (3 points) Explain briefly the term *instruction-level parallelism*.

   (b) (5 points) What limits instruction-level parallelism in programs?

   (c) (4 points) Explain briefly the terms *out-of-order execution* and *in-order commit*.

   (d) (3 points) Why is in-order commit important?

   (e) (3 points) Explain the role of the reorder-buffer in exploiting ILP.

   (f) (2 points) What specific purpose does register renaming play in exploiting ILP?

   (g) (3 points) What is a basic block? What is the average size of a basic block for a RISC instruction set?

   (h) (2 points) Based on your answer to the previous question, explain why we need branch predictors to exploit ILP effectively.

This page intensionally left blank.

2. *Short questions (20 points)* Answer the following questions using 3-4 sentences for each one.

   (a) (4 points) Explain the terms *shared-memory parallel programming* and *distributed-memory parallel programming*, focusing on the distinctions between these two styles of programming.

   (b) (3 points) Explain the difference between *true-sharing* and *false-sharing* in the context of shared-memory parallel programming. Which of these patterns of sharing is bad for scalability of parallel programs?

   (c) (4 points) Explain what is meant by an *atomic instruction.* Give two examples of atomic instructions (these do not have to be instructions in an actual ISA). Explain briefly how we use atomic instructions explicitly or implicitly in writing shared-memory programs.

   (d) (2 points) What is the difference between a *direct-mapped cache* and a *set-associative cache*?

   (e) (4 points) Explain the terms *write-invalidate* and *write-broadcast* in the context of cache-coherent architectures.

   (f) (3 points) What considerations determine the choice of step size in using finite-difference methods to solve differential equations approximately?

This page intensionally left blank.

3. *Numerical methods (30 points)*

The 2D Poisson equation is $\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} = f(x,y)$. Consider the following problem:

- The domain is the unit square [0,1]x[0,1], as shown in Figure 1
- $f(x,y) = 2xy$
- The value of u at the boundary is fixed at 40.

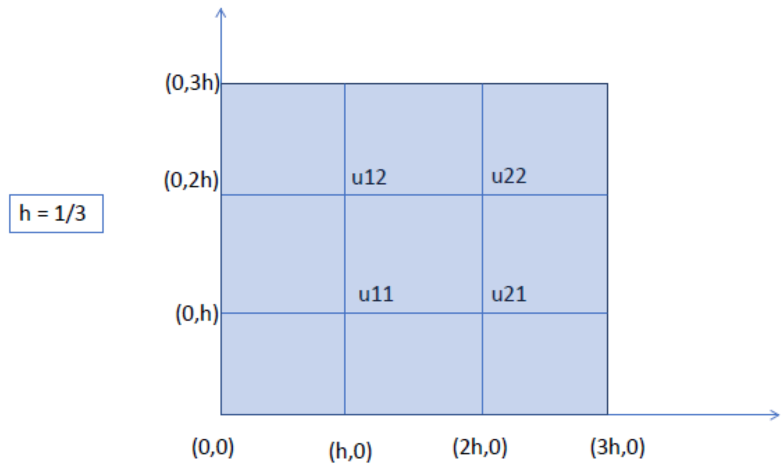

Figure 1: Grid for solving Poisson's equation using centered differences

We want to use the centered-difference method to solve this differential equation approximately.

(a) (5 points) Write down the 1D centered-difference formula for the first and second derivatives of a function $w(x)$. Assume the step size is h and show the formula for a point $x = nh$.

(b) (5 points) Using the 2D centered-difference formula, discretize Poisson's equation at a point (mh, nh), where h is the step size.

(c) (15 points) Use your answer in the previous part to write down a linear system for the four unknowns $u11, u12, u21, u22$ shown in the figure. You do not have to solve this system.

(d) (5 points) Based on your physical intuition, which of $u11, u12, u21, u22$ do you think will have the largest value? Which one will have the smallest value?

This page intensionally left blank.

This page intensionally left blank.

4. *Atomic operations (25 points)*

In this problem, you must implement a construct called a *counting semaphore* using a function called *test-and-set*.

The type of test-and-set is: *int test-and-set (Lock lockVar);*, and it atomically sets *lockVar* to 1 and returns its previous value.

(a) (5 points) Explain how *test-and-set* can be implemented using the swap instruction discussed in class.

(b) (20 points) A counting semaphore contains an integer value. You must write two functions, each of which must execute atomically:

- *sem-post*: increment the value
- *sem-wait*: wait for the value to be positive, then decrement the value

Add C-like pseudocode to the stub below, and explain briefly how your code works. Ignore initialization.

```
typedef struct {
int value;
Lock lockVar;
} sem-t;

sem-post (sem-t *s)
{//your code below
...........
}

sem-wait (sem-t *s)
{//your code below
............
}
```

This page intensionally left blank