Please print in pen:
Waterloo Student ID Number:

WatIAM/Quest Login Userid:

UNIVERSITY OF WATERLOO

Examination
Final
Winter 2024
ECE 459

Open Book

Candidates may bring any reasonable aids.

Times: Friday 2024-04-19 at 12:30 to 15:00 (3PM)
Duration: 2 hours 30 minutes (150 minutes)
Exam ID: 5589727
Sections: ECE 459 LEC 001,002
Instructors: Huanyi Chen, Jeff Zarnett

1. This exam is open book, open notes, calculators with no communication capability permitted.
2. Turn off all communication devices. Communication devices must be stored with your personal items for the duration of the exam. Taking a communication device to a washroom break during this examination is not allowed and will be considered an academic offence.
3. Place all bags at the front or side of the examination room, or beneath your table, so they are inaccessible.
4. There are six (6) questions. Not all are equally difficult.
5. The exam lasts **150** minutes and there are 120 marks.
6. Verify that your name and student ID number is on the cover page and that your examination code appears on the bottom of each page of the examination booklet.
7. If you feel like you need to ask a question, know that the most likely answer is "Read the Question". No questions are permitted. If you find that a question requires clarification, proceed by clearly stating any reasonable assumptions necessary to complete the question. If your assumptions are reasonable, they will be taken into account during grading.
8. After reading and understanding the instructions, sign your name in the space provided below.

Signature

1 Short Answer [30 marks total]

Answer these questions using at most three sentences. Each question is worth 3 points.

(a) (Rate Limits) A colleague suggests that one way to get around the rate limit of a third party service is to create multiple (free) accounts. Is this a good plan? Justify your answer.

(b) (Asynchronous I/O) For creating your Gitlab repos for the course, I wrote a little Rust tool. It has a statement `sleep(Duration::new(10, 0));` in between repo creation and the call to create the branch-protection rules. Without this statement, the branch protection rule call returns an error. Sleeping for a fixed period of time isn't very efficient; what should this code do instead?

(c) (Reduced-resource computation). One way that we could "do less work" is by setting a maximum time limit on a decision and proceeding with a default action if no decision is reached before that time. Give an example of a problem domain where this strategy is reasonable, the default action, and justify your selections.

(d) (DevOps) Imagine that you found that malicious actors had created a lot of fake accounts in your system and their existence was slowing down queries. List three ways that you could speed things up again without spending more money (so you cannot say you would get a better DB server).

(e) (Parallelism) Fifty years ago, before it was practical to send out notifications en masse via text or e-mail, one mechanism for communicating a message was a phone tree. A phone tree is rather like the data structure of a tree, where every node may have up to n children. A message is initiated at the "root" of the tree and needs to be told to everyone in the tree. Each person in the tree is then responsible for telephoning (sequentially) n other people to notify them, who in turn call the people on their list, and so on until the message is distributed to everyone in the tree. What kind of parallelism is this?

(f) (Accuracy for Time) In trading accuracy for time, moving from 64-bit double-precision floating point types to 32-bit types could be an improvement. Would going from 32-bit types to 16-bit types be even better? State any assumptions you rely on for your answer.

(g) (SIMD) Given the following Stream-VByte control stream and data stream, show the expanded data (show all leading zeros, if needed).

Control Stream: `001011111001`
Data Stream: `50bc6386e9451dd1b3e5e7b71536589ab2`

(h) (Compiler Optimization) Many of the compiler optimizations we talked about involved giving the compiler some more information (e.g., lifetime annotations) in the expectation that it results in better decisions. What is a scenario where deleting something would help the compiler make better decisions?

(i) (Profiling) Suppose you a causal profiler to do some what-if analysis and it presented you three options for speedup, each on a different part of the program, each of which could be accomplished in a reasonable time. Give two some reasons why you might not be able to perform all three.

(j) (LLMs) To what extent did you use LLM tools (e.g., ChatGPT, Github Copilot) to help with the course assignments? Whatever answer you give is only for instructors to understand better what (if anything) should change for next year, and no matter what your answer on this subject is, it will not be used against you. Any 2-3 sentence answer that addresses the question will earn full marks. We appreciate your honesty.

2 Queueing Theory [17 marks total]

2.1 Server Analysis [7 marks]

We have two backend servers. Each server completes a request, on average, in 15 ms; the time to complete requests is exponentially distributed. Jobs arrive at a single central queue and are processed FIFO. On average, 135 000 jobs arrive per hour in a Markov process. For this system, write down/calculate the values below:

Property	**Value**	**Property**	**Value**
Kendall Notation Description		Utilization (ρ)	
Intermediate Value (K)		Completion time average (T_q)	
Average queue length (W)		Maximum arrivals/hour the system can handle (full utilization)	
Maximum average service time the system can tolerate		–	–

2.2 Slooooowwwwwwwww Doooooooooowwwwwwnnnnnnn [4 marks]

The *slowdown* of a job is defined as the job's response time divided by its service requirement. Remember that response time is the time from when a job arrives until it completes (so service plus waiting time). So if a job is expected to take $50\ ms$ and it waited $100\ ms$ in total, its response time is $150\ ms$ and the slowdown is $150/50 = 3$.

Some people argue that mean slowdown is more important to performance rather than mean response time. Explain why this may be. Hint: it may help to think about a concrete scenario like processing uploaded files.

2.3 About that Mouse... [6 marks]

Recall the laboratory study involving a certain Mouse, who is known for suing people and whose first cartoon has finally passed into the public domain. There is an argument that the various systems of FastPass and FastPass+ are too complicated and confusing. What if the Mouse just decided to openly admit it's about money and park visitors may simply buy faster access to individual rides with money directly? For example, if you want to go on the *Star Wars* ride, you can take the slow line, or pay a $20 fee (US Dollars, per person, plus tax) to join the fast line. Discuss the expected pros and cons of this approach in terms of its impact on queue lengths and guest experience.

3 Profiling [13 marks total]

3.1 Lies, Deception! [7 marks]

One problem that we can observe in profiling is called *skid*. The perf wiki describes it as follows: "Interrupt-based sampling introduces skids on modern processors. That means that the instruction pointer stored in each sample designates the place where the program was interrupted to process the PMU interrupt, not the place where the counter actually overflows..."; the counter overflowing is the trigger to take the next sample.

Part i. [3 marks] What would be some indications that you are experiencing skid in profiling a program? Show an example to illustrate, using pseudo-assembly annotated with time percentages.

Part ii. [4 marks] Modern Intel and AMD x86_64 CPUs introduce the ability to have more precise (low- or no-skid) sampling take place with some hardware support. Speculate about the reasons why this mode is not always used on CPUs that support it.

3.2 Profiler-Guided Optimization (POGO) [6 marks]

In addition to the common cases of inlining and branch prediction, Profiler-Guided Optimization (POGO) can be used to help the compiler make some other optimization decisions. For the following optimizations, explain how it would help improve the performance of the program.

Register Allocation

Exception Handling Code Separation

Loop Vectorization (Hint: SIMD is vectorization).

4 Rust [20 marks total]

4.1 Lifetimes [5 marks]

Part i. Without changing any of the code but just adding lifetime annotations to the `concat` function to make the code compilable.

```
fn concat(x: &mut String, y: &mut String) -> &String {
    println!("str to be concatenated {}", y);
    x.push_str(y.as_str());
    x
}

fn main() {
    let mut string1 = String::from("abcd");
    let result;
    {
        let mut string2 = String::from("xyz");
        result = concat(&mut string1, &mut string2);
    }
    println!("The string after concat is {}", result);
}
```

Part ii. Given the following code

```
#[allow(dead_code)]
#[derive(...)]
struct Parent { name: String }

#[allow(dead_code)]
#[derive(...)]
struct Child<'a> { parent: &'a Parent, name: String }

#[allow(dead_code)]
#[derive(...)]
struct Family<'a> { parent: Parent, child: Child<'a> }

fn main() {
    let parent = Parent {
        name: String::from("parent"),
    };
    let child = Child {
        parent: &parent,
        name: String::from("child"),
    };

    let family = Family { parent, child };
    println!("{:?}", family);
}
```

- If it is compilable, please explain the lifetime of `parent` and `child`, then explain what traits are needed for `#[derive]`
- If it is not compilable, please explain why it is not, fix the code, then explain what traits are needed for `#[derive]`

4.2 Memory Consistency [15 marks]

I have implemented a minimal spin lock and would like to use it to do some counting work.

```
struct MySpinLock {
    locked: AtomicBool,
}

impl MySpinLock {
    pub const fn new() -> Self {
        Self {
            locked: AtomicBool::new(false),
        }
    }

    pub fn lock(&self) {
        while self.locked.swap(true, Relaxed) {
            std::hint::spin_loop();
        }
    }

    pub fn unlock(&self) {
        self.locked.store(false, Relaxed);
    }
}
```

I have a (concurrent) queue of CPU intensive tasks where each task can be executed by calling its `execute()` function. The queue supports concurrent access. My target is to count how many tasks are successfully executed.

Part i. [10 marks] Please write the code to make use of `mylock` and `counter` to achieve it. A sequential implementation is provided. You program should be more efficient than this.

You can assume the queue is finite and there are in total 8 cores. You can also assume `Send` and `Sync` are correctly implemented, so `Arc` is not needed. For `MyCounter`, it only has `load()` and `store()` implemented, in the same way as `AtomicUsize` implements them.

```
// ...
use std::sync::atomic::Ordering::{Acquire, Relaxed, Release, SeqCst};
// ...

// You can assume they all have Send and Sync traits
// correctly implemented
let queue = setup_queue();
let mylock = MySpinLock::new();
let counter = MyCounter::new(0);
loop {
    match queue.pop() {
        None => break,
        Some(task) => {
            if task.execute() {
                let old = counter.load(Relaxed);
                let new = old + 1;
                counter.store(new, Relaxed);
            }
        }
    }
}
println!("counted: {}", counter);
```

Part ii. [5 marks] Please inspect `MySpinLock` implementation. Assume there is no compiler re-ordering. Please justify under what circumstance it is correct and not correct. Can you modify it to make it always correct?

5 CUDA [20 marks total]

We would like to compute a matrix multiplication `C = A x B`, where each matrix is `1024 * 1024`. The CUDA grid is configured with `64 * 64` thread blocks and each thread block contains `16 * 16` threads.

Part i. [10 marks] In CUDA, the shared memory of a thread block can be accessed far more quickly than the global memory by threads within the thread block. However, a naive implementation of matrix multiplication that does not take advantage of shared memory is: each thread reads one row of `A` and one column of `B`, and computes the corresponding element of `C`. Please fill in the naive implementation in the following code and also answer the question.

```
// Matrices are stored in row-major order:
typedef struct {
    int width;
    int height;
    float* elements; // elements[height][width] but as an 1D-array pointer
} Matrix;

__global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)
{









}
```

1. How many elements does each thread has to read from the global memory?

2. How many elements does each thread has to write to the global memory?

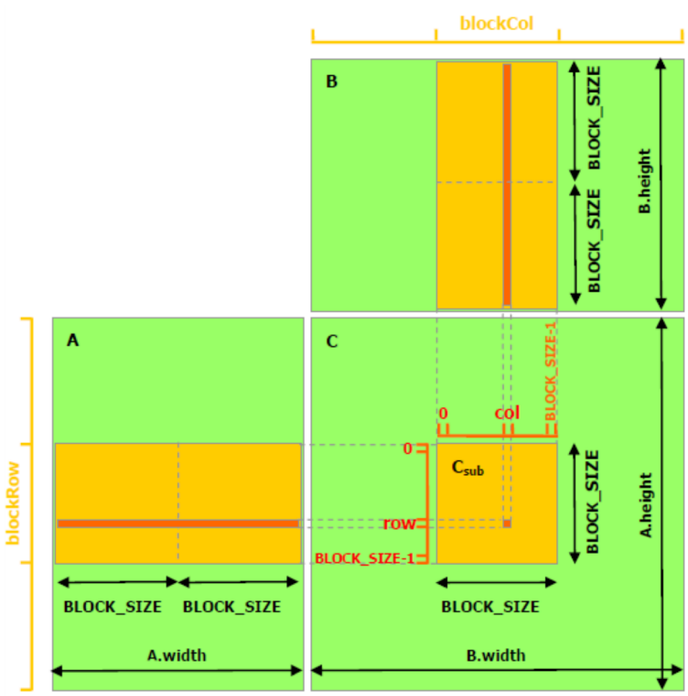
Part ii. [10 marks] To improve the performance of matrix multiplication by taking advantage of shared memory, one can use **Blocked Matrix Multiplication**. The `__shared__` keyword indicats that `As` and `Bs` are put into the shared memory within a thread block. All threads within the same thread block can access the same shared memory but they cannot access the shared memory that belongs to another thread block. The following code shows the implementation. You can assume `GetSubMatrix` returns a submatrix within the original matrix, but the submatrix is always of size `16 * 16`. For example:

`GetSubMatrix(A, 0, 0)` returns `A.elements[0..15][0..15]`, where `&As.elements[0][0] == &A.elements[0][0]`
`GetSubMatrix(A, 0, 1)` returns `A.elements[0..15][16..31]`, where `&As.elements[0][0] == &A.elements[0][16]`
`GetSubMatrix(A, 1, 1)` returns `A.elements[16..31][16..31]`, where `&As.elements[0][0] == &A.elements[16][16]`

And `GetElement` returns a single element within the submatrix.

```
#define BLOCK_SIZE 16

 __global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)
{
    int blockRow = blockIdx.y;
    int blockCol = blockIdx.x;

    Matrix Csub = GetSubMatrix(C, blockRow, blockCol);

    float Cvalue = 0;

    int row = threadIdx.y;
    int col = threadIdx.x;


    for (int m = 0; m < (A.width / BLOCK_SIZE); ++m) {
        Matrix Asub = GetSubMatrix(A, blockRow, m);
        Matrix Bsub = GetSubMatrix(B, m, blockCol);

        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

        As[row][col] = GetElement(Asub, row, col);
        Bs[row][col] = GetElement(Bsub, row, col);

        __syncthreads(); // first sync
        for (int e = 0; e < BLOCK_SIZE; ++e)
            Cvalue += As[row][e] * Bs[e][col];
        __syncthreads(); // second sync
    }

    // Write Csub to global memory
    // which directly changes value(s) in matrix C
    // This function only exists for part ii
    SetElement(Csub, row, col, Cvalue);
}
```

The high-level idea is shown in the following figure.



Given above, please answer the following questions:

1. How many multiplications does each thread has to perform?

2. What is the purpose of doing the first `__syncthreads`?

3. What is the purpose of doing the second `__syncthreads`?

4. How many elements does each thread has to read from the global memory?

5. How many elements does each thread has to write to the global memory?

6 Caching [20 marks]

Though our in-class discussion of caching had focus on a hardware implementation, we know that software products that do something similar (e.g., redis) exist. In this question, you will write a pseudocode description of behaviour for a distributed software cache that uses the MESI states and has write-back behaviour. This cache is for data items retrieved from a database, so if the item is not in any node's cache, write down `retrieve item i from database`, for example. You can assume the cache to be of significant size and that the least-recently-used (LRU) algorithm is used for replacement. As the cache is distributed, we do need to consider what happens if a node comes online and joins the cluster, and what happens if a node is going to shut down and leave the cluster. You may ignore situations like crashes or network outages.

```
Add to cache ( item i ) { // Helper function
  if cache is not full {
    add i to cache
    return
  } else {
    old = least-recently-used item
    if old state is M {
      write old to database
    }
    replace old with i
  }
}


Current Node Startup {
  get known_nodes from coordinator
  Send message to known_nodes to indicate joining
}


New Node Joins ( node n ) {
  Add n to known_nodes
}
```

```
Current Node Shutdown {









}

Node Leaves ( node n ) {









}
```

(Question continues on next page.)

```
Get Item ( item i ) {




}
Other Node Searching ( item i ) {




}
```

```
Update Item ( item i ) {


}

Invalidate ( item i )  {


}
```

Extra Space. Use this space if needed. Be sure to indicate on the original question to refer to this extra space. Also, write the question number (and subpart, if applicable) in the extra space next to your answer.