ECE 459: Programming for Performance
Midterm

March 2, 2012

Student ID: _____

Name (optional): _____

- This is a **closed-book** exam
- You are allowed **non-programmable** calculators
- No other aides permitted
- There are **4 questions**
- There are **9 pages**
- **Write all your answers on the exam paper**
- Do not separate the pages (except for page 9)
- If, for some reason, your pages are separated, write your student number **on each page**
- **Hint:** leave Question 4, Part 2 for last

Question	**Mark**
1	/20
2	/20
3	/35
4	/25
Total	/100

Question 1: Definitions (20 marks, 10 marks each)

Pick only **two** of the three defintions and answer them.

Process vs. Thread
Explain the main difference between threads and processes. Write one advantage and disavantage to using threads over processes.

Mutex vs. Spinlock
Explain the main difference between a simple mutex (assume the mutex will never act like a spinlock) and a spinlock (in terms of the `lock` operation). Describe a reasonable condition where a spinlock would outperform a simple mutex.

Data vs. Task Parallelism
Explain the main difference between data and task parallelism. Give one example of a data parallelism operation and one example of using task parallelism.

Question 2: Parallelization Limitations (20 marks)

Consider a program which has a serial and parallelizable component. The serial component executes in 2 seconds and the parallel component executes in 8 seconds, for a total runtime of 10 seconds **when run serially**.

Part 1 - Amdahl's Law (10 marks)

Assume the problem size is **fixed**. Calculate how many processors would you need to reach a desired speedup of 3.75.

N =

Question 2: Parallelization Limitations (20 marks)

Part 2 - Gustafson's Law (10 marks)

For the same problem, now assume the problem size is not fixed. We have **4 processors** and the serial runtime is **constant** for any problem size (larger problems only require that the parallel component runs for longer), which is 2 seconds.

Calculate how long the parallel component would have to run in a **parallel execution** (in seconds) to reach the speedup of 3.75.

Next, calculate how long the parallel component would run in a **serial execution** under the same conditions.

Finally, assuming the problem size scales linearly with the execution time, how many times larger did we have to make the problem until we saw our desired scaling?

Parallel component, time of execution in parallel (s) =

Parallel component, time of execution in serial (s) =

How many times larger is the problem = Above answer / 8 (s) =

Question 3: Race Conditions/Thread-Safety (35 marks)

Consider the following code:

```
1 void swap(int *x, int *y) {
2     int t = *x;
3     *x = *y;
4     *y = t;
5 }
```

For this question, assume the following global variables: `int a = 1, b = 2, c = 3`.

Part 1 (5 marks)

Let's say we have two threads, one calls `swap(&b, &a)` and the other `swap(&b, &c)`. Write down all possible expected outputs of the values of the variables after the calls complete, assuming the code is **thread-safe**.

Expected outputs:

Part 2 (10 marks)

Assume each line of code is atomic. Write down an interleaving of the code in two seperate threads that shows this function is not thread-safe along with the final output. That is, your output should not match either expected result you found in Part 1. You should only have code in one thread for each row of the table. (Use a scrap piece of paper and write your final answer in the table)

Thread 1 - `swap(&b, &a)`	**Thread 2 -** `swap(&b, &c)`	**a**	**b**	**c**

Actual output:

Part 3 (5 marks)

Explain briefly (one sentence) what the `restrict` keyword does. Assume we declare both pointers with a `restrict` keyword. Would this make the function thread-safe? Explain why or why not. If it does make the function thread-safe, ignore the locking portion when completing Part 4.

Question 3: Race Conditions/Thread-Safety (35 marks)

Part 4 (15 marks)

Assume in your code you create a mutex, `m`, and properly destroy it when the program ends(!). Write below where you would lock and unlock the mutex to make the code thread-safe (pseudocode like `lock(m)` is okay). Use the finest-grain locking possible with a single lock.

```
1  void swap(int *x, int *y) {
2
3      int t = *x;
4
5      *x = *y;
6
7      *y = t;
8
9  }
```

Show what happens with the same interleaving (that you used in **Part 2**) on your modified code, now with locks. Verify you get one of the expected outputs. Explain why the code is now thread-safe in all cases.

Thread 1 - `swap(&b, &a)`	**Thread 2 -** `swap(&b, &c)`	**a**	**b**	**c**

Actual output:

Explanation:

Question 4: Dependencies (25 marks)

Memory-carried Dependencies (15 marks)

Consider the following code:

```
1 z = long_calc_a(x)
2 x = long_calc_b(y)
3 i = long_calc_a(x) + z
4 j = long_calc_c(x) + y
```

List all RAW (read after write), WAR (write and read) and WAW (write after write) dependencies of the variables, along with their type in the provided table below. For example, if we included RAR (read after read) "dependencies", we would notice a read of `y` on line 2, followed by a read of `y` on line 4 and record it in the table as shown.

Variable	**First Line**	**Second Line**	**Type of Dependency**
y	2	4	RAR (read after read)

Now, assume the following runtimes for each function (which do not executive speculatively):

`long_calc_a` = 2 seconds
`long_calc_b` = 1 second
`long_calc_c` = 3 seconds

Show how you minimize the runtime of the code using any number of processors. Assume the cost of reading/writing/copying variables is 0. Clearly identify what lines can run in parallel.

What is the minimum amount of time required to execute the code in parallel?

Question 4: Dependencies (25 marks)

Loop-carried Dependencies (10 marks)

Consider the following code:

```
1 for (int i = 0; i < 5; ++i) {
2     a[i + 1] = a[i % 3];
3 }
```

Show how you would minimize the runtime of the loop using any number of processors. Assume that each iteration of the loop takes 1 unit of time. Clearly identify which iterations are run sequentially/in parallel. For parallel execution, show which processor executes each iteration.

What is the minimum amount of time required to execute the code in parallel?

What is the minimum amount of time required to execute the code in parallel if there are 100 iterations of the loop (assuming again, there's an unlimited number of processors)?

Scrap Page (feel free to detech, you do not need to hand-in)