ECE459 Midterm W2012

Question 1

1. Each process has its own virtual address space, while threads run in a shared memory space. Thread advantages include faster and simpler interthread communication (compared to IPC), since processes must use operating system utilities to communicate; threads also have lower startup/shutdown and synchronization costs. Thread disadvantages include less safety against bugs or security holes: since there is shared memory, buffer overruns or other memory security issues can impact multiple threads.

2. A mutex implements blocking: locking a mutex blocks until the lock is acquired. Spinlocks are nonblocking: if the lock is not acquired, the thread is not blocked, but simply retries again until the lock is acquired. Spinlocks may be better in a multi-core system where locks are held for very short periods of time, since this saves the overhead of putting the threads to sleep and reawakening them.

3. In data parallelism, multiple threads perform the same operation on separate data items. An example would be adding 1 to every element in an array: we assign some portion of the array to every thread, and each thread adds one to each element of their portion. In task parallelism, multiple threads perform different operations on separate data items. An example would be having one thread filtering out negative numbers from an array and a different thread summing the filtered elements.

Question 2

1. Using Amdahl's Law, we have that $3.75 = 1/(0.2 + 0.8/x)$. Solving for x, the number of processors, we have that we would need 12 processors.

2. Not covered. Using Gustafson's Law, we have that speedup = $N+(1-N)S$. Plugging in the numbers, we get $3.75 = 4 + (1-4)*S$. Solving for S, the serial fraction of the program, we get $S=1/12$. Hence in the parallel execution, the parallel component runs for 22 seconds. Since there are 4 processors, in the

serial execution, the parallel component runs for 88 seconds. The problem is 2.75x larger.

Question 3

1. a=3, b=1, c=2; a=2,b=3,c=1

2.

Thread 1	Thread 2	a	b	c
int t = *x		1	2	3
	int t = *x	1	2	3
*x = *y		1	1	3
*y = *t		2	1	3
	*x = *y	2	3	3
	*y = *t	2	3	2

3. Not covered. Adding restrict tells the compiler that for the lifetime of the pointer, no other pointer will be used to access the object to which it points. This does not make the function thread-safe.

4. lock(m) at beginning of function and unlock(m) at end. Ensures that other threads cannot read or write to x and y in the middle of a swap.

Question 4

1. Not really covered?

Variable	First line	Second line	Type of dependency
y	2	4	RAR
z	1	3	RAW
x	1	2	WAR
x	2	3	RAW

Line 1 must run before line 2, unless we first save the value of x. Lines 1 and 2 must run before line 3. Line 2 must run before line 4. Lines 3 and 4 can run in parallel. Hence we can run the code as follows.

1. Save value of x to temp variable. Run lines 1 and 2 in parallel, using the saved temp value of x for line 1.
2. When line 2 finishes, begin line 4.
3. When both lines 1 and 2 finish, begin line 3.

The runtime is 4 seconds.

2. We run the code as follows.
    a. a[1] = a[0]; a[4] = a[0]
    b. a[2] = a[1]; a[5] = a[1]
    c. a[3] = a[2]

    The code takes 3 units of time. The minimum amount of time to execute the code in parallel if there are 100 iterations of the loop is 3 units of time.