Page 1351 Exercises 6.2, 6.3, 6.6, 6.8, 6.14

6.2 You are trying to bake 3 blueberry pound cakes. Cake ingredients are as follows:

- 1 cup butter, softened
- 1 cup sugar
- 4 large eggs
- 1 teaspoon vanilla extract
- 1/2 teaspoon salt
- 1/4 teaspoon nutmeg
- 1 1/2 cups flour
- 1 cup blueberries

The recipe for a single cake is as follows:
Step 1: Preheat oven to 325°F (160°C). Grease and flour your cake pan.
Step 2: In large bowl, beat together with a mixer butter and sugar at medium speed until light and fluffy. Add eggs, vanilla, salt and nutmeg. Beat until thoroughly blended. Reduce mixer speed to low and add flour, 1/2 cup at a time, beating just until blended.
Step 3: Gently fold in blueberries. Spread evenly in prepared baking pan. Bake for 60 minutes.

6.2.1 [5] <§6.2> Your job is to cook 3 cakes as efficiently as possible. Assuming that you only have one oven large enough to hold one cake, one large bowl, one cake pan, and one mixer, come up with a schedule to make three cakes as quickly as possible. Identify the bottlenecks in completing this task.

- Prepare the first cake batter following the recipe until Step 3, where the batter is spread in the pan.
- While the first cake is baking for 60 minutes, prepare the batter for the second cake using the same bowl and mixer.
- Once the first cake is done baking, remove it and place it on a cooling rack.
- Place the second cake in the oven to bake.
- While the second cake is baking, prepare the batter for the third cake using the same bowl and mixer.
- Once the second cake is done baking, remove it, and place it on the cooling rack.
- Put the third cake in the oven to bake.
- The bottleneck is the availability of the oven.

6.2.2 [5] <§6.2> Assume now that you have three bowls, 3 cake pans and 3 mixers. How much faster is the process now that you have additional resources?

- presuming that i'm still using the same small oven that can accommodate one cake at a time, baking three cakes will still take roughly the same amount of time (3 x 60 minutes per cake) which is 3 hours total because the same bottleneck (the oven size) still exists.

6.2.3 [5] <§6.2> Assume now that you have two friends that will help you cook, and that you have a large oven that can accommodate all three cakes. How will this change the schedule you arrived at in Exercise 6.2.1 above?

- This will cut the schedule to about 1/3 of the time. Since all three cakes can be made simultaneously.

6.2.4 [5] <§6.2> Compare the cake-making task to computing 3 iterations of a loop on a parallel computer. Identify data-level parallelism and task-level parallelism in the cake-making loop.

- Data-level parallelism: In the cake-making process, where multiple cake batters are prepared simultaneously using separate bowls, mixers, and ingredients. Each cake batter is self-contained and can be processed independently in parallel.

- Task-level parallelism: It is observed during the baking phase, where one cake can be in the oven baking while another is being prepared. This parallelism resembles the execution of multiple iterations of a loop concurrently, where each iteration represents a distinct task and can run autonomously.

In both scenarios, the primary objective is to leverage parallelism for process optimization, whether it pertains to baking cakes or executing iterations on a parallel computer.

6.3 Many computer applications involve searching through a set of data and sorting the data. A number of efficient searching and sorting algorithms have been devised in order to reduce the runtime of these tedious tasks. In this problem we will consider how best to parallelize these tasks.

6.3.1 [10] <§6.2> Consider the following binary search algorithm (a classic divide and conquer algorithm) that searches for a value X in an sorted N-element array A and returns the index of matched entry: Assume that you have Y cores on a multi-core processor to run BinarySearch. Assuming that Y is much smaller than N, express the speedup factor you might expect to obtain for values of Y and N. Plot these on a graph.

- Since there are y cores, and the array is of size N.
- Therefore, if we run the algorithm in parallel to take full advantage of the processor's capabilities, we will get N/Y elements to be searched in each core.
- The Binary search algorithm has a Big O(Log(n)).
- Therefore, the search will take approximately: Number of cores * the speed of each search = Y * Log (N/Y)
- Thus, according to Amdahl's law, The speedup S= 1/{(1-P)+ Y/P}

6.3.2 [5] <§6.2> Next, assume that Y is equal to N. How would this affect your conclusions in your previous answer? If you were tasked with obtaining the best speedup factor possible (i.e., strong scaling), explain how you might change this code to obtain it.

- if Y equals N, then the algorithm will run in Y*Log(N/Y)= Y*Log(1) = Y.

6.6 Matrix multiplication plays an important role in a number of applications. Two matrices can only be multiplied if the number of columns of the first matrix is equal to the number of rows in the second. Let's assume we have an m × n matrix A and we want to multiply it by an n × p matrix B. We can express their product as an m × p matrix denoted by AB (or A·B). If we assign C = AB, and C(i,j) denotes the entry in C at position (i, j), then for each element i and j with 1 ≤ i ≤ m and (PICTURE). Now we want to see if we can parallelize the computation of C. Assume that matrices are laid out in memory sequentially as follows: a1,1, a2,1, a3,1, a4,1, ..., etc.

6.6.1 [10] <§6.5> Assume that we are going to compute C on both a single core shared memory machine and a 4-core shared-memory machine. Compute the speedup we would expect to obtain on the 4-core machine,

ignoring any memory issues.

• Assuming that P is the parallelizible portion of the entire code, and that the entire code is parallelizable.

• Speedup (S)= 1/[(1-P)+(P/N)]

= 1/[(1-1)+(1/4)]

= 4 times the original speed.

6.6.2 [10] <§6.5> Repeat Exercise 6.6.1, assuming that updates to C incur a cache miss due to false sharing when consecutive elements are in a row (i.e., index i) are updated.

With one cache miss, P = 4/5 its original value.

Therefore, S = 1/ [(1-0.8)+(0.8/4)] = 1/0.4 = 2.5 times the original speed.

6.6.3 [10] <§6.5> How would you fix the false sharing issue that can occur?

By using the cache and a TLB.

6.8 The dining philosopher's problem is a classic problem of synchronization and concurrency. The general problem is stated as philosophers sitting at a round table doing one of two things: eating or thinking. When they are eating, they are not thinking, and when they are thinking, they are not eating. There is a bowl of pasta in the center. A fork is placed in between each philosopher. The result is that each philosopher has one fork to her left and one fork to her right. Given the nature of eating pasta, the philosopher needs two forks to eat, and can only use the forks on her immediate left and right. The philosophers do not speak to one another.

6.8.1 [10] <§6.8> Describe the scenario where none of philosophers ever eats (i.e., starvation). What is the sequence of events that happen that lead up to this problem?

Each philosopher grabs one fork either the ones on the left or the ones on the right, resulting in a deadlock.

6.8.2 [10] <§6.8> Describe how we can solve this problem by introducing the concept of a priority? But can we guarantee that we will treat all the philosophers fairly? Explain. Now assume we hire a waiter who is in charge of assigning forks to philosophers. Nobody can pick up a fork until the waiter says they can. The waiter has global knowledge of all forks. Further, if we impose the policy that philosophers will always request to pick up their left fork before requesting to pick up their right fork, then we can guarantee to avoid deadlock.

By creating a priority system using queues, where each philosopher is assigned a number from P1 to PN, and allowing them access to the forks respectively. We can also introduce a time limit for how long each philosopher can use the forks.

6.8.3 [10] <§6.8> We can implement requests to the waiter as either a queue of requests or as a periodic retry of a request. With a queue, requests are handled in the order they are received. The problem with using the queue is that we may not always be able to service the philosopher whose request is at the head of the queue (due to the unavailability of resources). Describe a scenario with 5 philosophers where

a queue is provided, but service is not granted even though there are forks available for another philosopher (whose request is deeper in the queue) to eat.

Consider the philosophers are labeled anti-clockwise from P0 to P4. If P0 and P3 are eating, four forks are in use.P4 sits between P0 and P3 so has neither fork available, whereas P2 and P1 have one unused fork between them. Suppose P2 wants to eat. Were he to take up the fifth fork, deadlock becomes likely.

6.8.4 [10] <§6.8> If we implement requests to the waiter by periodically repeating our request until the resources become available, will this solve the problem described in Exercise 6.8.3? Explain.

If instead she asks the waiter and is told to wait, we can be sure that next time two forks are released there will certainly be at least one philosopher who could successfully request a pair of forks. Therefore deadlock cannot happen.

6.14 Download the CUDA Toolkit and SDK from hps:// developer.nvidia.com/cuda-downloads. Make sure to use the "emu-release" (Emulation Mode) version of the code (you will not need actual NVIDIA hardware for this assignment). Build the example programs provided in the SDK, and confirm that they run on the emulator. 6.14.1 [90] <§6.6> Using the "template" SDK sample as a starting point, write a CUDA program to perform the following vector operations:
1) a − b (vector-vector subtraction)
2) a · b (vector dot product) The dot product of two vectors a = [a1, a2, … , an] and b = [b1, b2, … , bn] is defined as:
Submit code for each program that demonstrates each operation and verifies the correctness of the results. 6.14.2 [90] <§6.6> If you have GPU hardware available, complete a performance analysis on your program, examining the computation time for the GPU and a CPU version of your program for a range of vector sizes. Explain any results you see. 6.15 AMD has recently announced that they will be integrating a graphics processing unit with their ×·86 cores in a single package, though with different clocks for each of the cores. This is an example of a heterogeneous multiprocessor system which we expect to see produced commercially in the near future. One of the key design points will be to allow for fast data communication between the CPU and the GPU. Presently communications must be performed between discrete CPU and GPU chips. But this is changing in AMDs Fusion architecture. Presently the plan is to use multiple (at least 16) PCI express channels for facilitate intercommunication.

I was unable to install the CUDA toolkit on my system for incompatibility.