Gavin McRoy
Homework 5

Due 11:59PM Dec. 6

This assignment is extra credit and optional. Students who have less than desired grades so far due to missed submission may benefit from completing it.

Exercise question 6.6 from Zybook:

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.

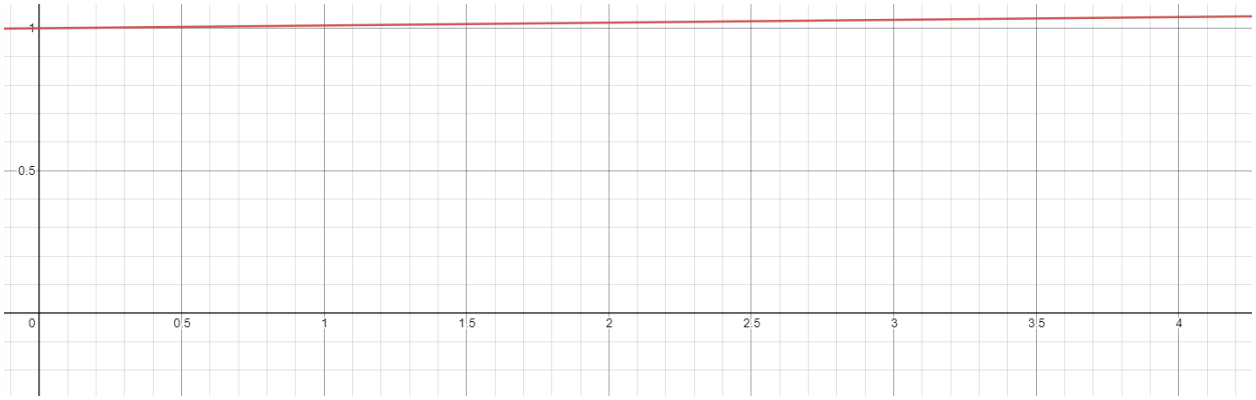
6.3.1 [15] <COD §6.2> Consider the following binary search algorithm (a classic divide and conquer algorithm) that searches for a value *X* in a sorted N-element array A and returns the index of matched entry:

```
BinarySearch(A[0..N-1], X) {
   low = 0
   high = N - 1

   while (low <= high) {
      mid = (low + high) / 2

      if (A[mid] > X)
         high = mid - 1
      else if (A[mid] < X)
         low = mid + 1
      else
         return mid         // found
   }

   return -1                // not found
}
```

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.

Binary search by itself is pretty fast, and it isn't easy to speed up and parallelizable without doing a lot of surgery on the code. I doubt increasing the cores above two will make any difference. Maybe you could get away with core 1 making the comparison between low and high, core two performing the computation for mid, and core 3 performing the comparison, but this may damage performance since core communication also has overhead.

Below is a graph of what the program speed will possibly look like. The X-axis is the number of cores, and Y is the run time. We assume 1 second is the run time on some arbitrary data set. I would believe the run time would slowly increase the more cores you add because of overhead. The runtime may roughly stay the same also. Any performance benefits would be profoundly negligible without architectural changes.



6.3.2 [15] <COD §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.

I believe that the number of cores is made equal to the number of array elements. I do not see how this will give any speed up. It seems more efficient to create threads to compare the N elements with, let's say, x and perform these in parallel. The comparison can be completed in the amount of time it takes to perform a single computation. However, binary search already takes Log2N operations, so the performance gain is again negligible.

Exercise question 6.6 from Zybook:

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 $m \times n$ matrix A and we want to multiply it by an $n \times p$ matrix B. We can express their product as an $m \times p$ matrix denoted by AB (or $A \cdot B$). 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 $1 \le i \le m$ and $1 \le j \le p$ $c_{i,j} = \sum_{k=1}^{n} a_{i,k} \text{ X } b_{k,j}$. Now we want to see if we can parallelize the computation of C. Assume that matrices are laid out in memory sequentially as follows: $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, $a_{4,1}$, …, etc.

6.6.1 [25] <COD §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.

This problem is already extremely parallel by nature and is often solved on GPUs with thousands of CUDA cores. The speedup expected by this notably parallel problem is close to 4 but not precisely four due to Amdahl's law. The summation may be performed on a single core, so the speed up may only be about 3.8x, but still, the value will be close to 4 but not exactly 4.

6.6.2 [15] <COD §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.

So if 4 cores are all working but map to the same cache line, then when the cache miss happens it will reduce the speed up factor by a factor 3 times the cost of repairing the cache miss

Exercise question 6.7 from Zybook:

6.7 Consider the following portions of two different programs running at the same time on four processors in a *symmetric multicore processor* (SMP). Assume that before this code is run, both x and y are 0.

Core 1: `x = 2;`
Core 2: `y = 2;`
Core 3: `w = x + y + 1;`
Core 4: `z = x + y;`

6.7.1 [15] <COD §6.5> What are all the possible resulting values of w, x, y, and z? For each possible outcome, explain how we might arrive at those values. You will need to examine all possible interleavings of instructions.

1. 3rd Core x = 0, y = 0 and compute + write w = 1
   4th Core x = 0, y = 0 and compute + write z= 0
2. 3rd Core x = 2, y = 0 and compute + write w = 3
   4th Core x = 0, y = 0 and compute + write z= 0
3. 3rd Core x = 0, y = 2 and compute + write w = 3
   4th Core x = 0, y = 0 and compute + write z= 0
4. 3rd Core x = 2, y = 2 and compute + write w = 5
   4th Core x = 0, y = 0 and compute + write z= 0
5. 3rd Core x = 0, y = 0 and compute + write w = 1
   4th Core x = 2, y = 0 and compute + write z= 2
6. 3rd Core x = 2, y = 0 and compute + write w = 3
   4th Core x = 2, y = 0 and compute + write z= 2
7. 3rd Core x = 0, y = 2 and compute + write w = 3
   4th Core x = 2, y = 0 and compute + write z= 2
8. 3rd Core x = 2, y = 2 and compute + write w = 5
   4th Core x = 2, y = 0 and compute + write z= 2
9. 3rd Core x = 0, y = 0 and compute + write w = 1
   4th Core x = 0, y = 2 and compute + write z= 2
10. 3rd Core x = 2, y = 0 and compute + write w = 3
    4th Core x = 0, y = 2 and compute + write z= 2
11. 3rd Core x = 0, y = 2 and compute + write w = 3
    4th Core x = 0, y = 2 and compute + write z= 2
12. 3rd Core x = 2, y = 2 and compute + write w = 5
    4th Core x = 0, y = 2 and compute + write z= 2
13. 3rd Core x = 0, y = 0 and compute + write w = 1
    4th Core x = 2, y = 2 and compute + write z= 4

14. 3rd Core x = 2, y = 0 and compute + write w = 3

4th Core x = 2, y = 2 and compute + write z= 4

15. 3rd Core x = 0, y = 2 and compute + write w = 3

    4th Core x = 2, y = 2 and compute + write z= 4

16. 3rd Core x = 2, y = 2 and compute + write w = 5

    4th Core x = 2, y = 2 and compute + write z= 4

6.7.2 [15] <COD §6.5> How could you make the execution more deterministic so that only one set of values is possible?

The execution is proper since the correct values are inside the variable. Technically there is no need for any synchronization. In this instance dependent variables should refer to the values of the variable only when they are properly updated. For example Core 1 and 2 should update first and then only core 3 and 4 refer to them. Achievable by synchronization or removing data dependencies