CS260 Homework 1

January 22, 2020

Deadline. The homework is due **Monday Jan. 27**. You can submit your write-up to me after class or before that.

Late days. You have five late days to use throughout the quarter. This includes homework, course project proposal and project final report.

Collaboration policy. Discussion on the homework problems is allowed, after you have thought about the problems on your own. It is OK to get inspiration (but not solutions) from books or online resources, again after you have thought about the problems on your own. You must cite your collaborators and resources fully and completely (e.g., *Alice explained to me the definition of work in Problem 1* or *I used Wikipedia site* `https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)` *about Master theorem for Problem 3*). You must write up your solution independently: close the book and all of your notes, and with no one else helping you when you are writing your final answers.

Write-up. Please do not use hand-writing submissions. Using LaTeX is highly-recommended. For all problems, please explain how you get the answer instead of directly giving the final answer.

Programming Problems. For all programming assignments, you need to submit your source code through iLearn by 23:59 Jan. 27. In the write-up, you need to provide detailed report of the experimental results and analysis. The programming part gives you up to a 5% bonus when you get very good performance or write good "report" to analyze the experiments.

1 Solve Recurrences (15 pts)

(a) $T(n) = 2T(n/2) + \Theta(n \log n)$

(b) $T(n) = T(n/2) + \Theta(1)$

(c) $T(n) = T(\sqrt{n}) + \Theta(1)$

(d) $T(n) = 10T(n/5) + \Theta(n)$

(e) $T(n) = 3T(n/2) + \Theta(n^2)$

(f) (3pt bonus) $T(n) = T(n - \sqrt{n}) + \Theta(1)$

For (d) and (e), please select one of them and solve it without using Master Theorem.

2 Medians (10pts)

There is a well known linear-work algorithm for finding the median of a set of values (the median value is the value v such that if the values are sorted, v would be in the middle). In this question you don't need to understand why the algorithm works, but you need to answer some questions about it and analyze its costs based on a description of its steps:

(a) If the input has 5 or fewer values, find the median by brute force, otherwise:

(b) Group the input into $n/5$ groups of 5 and find the median of each group in parallel.

(c) Find the median of the medians recursively. Call this p.

(d) Use p to filter out $3/10^{th}$(s) of the values in $\Theta(n)$ work and $\Theta(\log n)$ span.

(e) Recurse on the remaining $7/10^{th}$(s) of the values.

Questions:

(a) (6pts) Write down recurrences for work and span.

(b) (4pts) Based on your recurrence, is the work linear (i.e. $\Theta(n)$)? Please explain.

(c) (3pts bonus) Based on your recurrence, calculate the span in terms of Θ.

3 N-ary Forking vs. Binary-forking (10 pts)

In the class we saw two implementations of parallel scan algorithm. The first one uses divide-and-conquer, and makes use of the results of a reduce algorithm.

```
Function PrefixSum(A, B, s, t, ps) {
  If  s=t  then B[s] = ps + A[s]
    In Parallel:
       PrefixSum(A, B, s, mid, ps)
       PrefixSum(A, B, mid+1, t, ps+leftSum)
}
```

Here the `leftSum` is the sum of the left half of the array computed as side-effects of the reduce algorithm. The divide-and-conquer reduce algorithm follows the same computational DAG to divide the array.

The other one uses two parallel for-loops. It is a recursive algorithm.

```
Function PrefixSum(In, n, Out) {
  if (n==1) Out[0] = In[0];
  para_for (i=0 to n/2)
    B[i] = In[2i]+In[2i+1]
  PrefixSum(B, n/2, C);
  Out[0] = In[0];
  para_for (i=1 to n) {
     if (i%2) Out[i] = C[i/2];
     else Out[i] = C[i/2-1] + In[i];
  }
}
```

Please analyze the work and depth of the two algorithms under n-ary fork-join and binary fork-join models, respectively (four recurrences in total).

(a) (4pts) Write down the recurrence for each of them, respectively.

(b) (3pts) Do they have the same work bound?

- If not, explain the reason.
- If so, do they also have the same recurrence in work? If not, explain the reason.

(c) (3pts) Do they have the same depth bound?

- If not, explain the reason.
- If so, do they also have the same recurrence in depth? If not, explain the reason.

4 Fibonacci Numbers (15 pts)

In the class we saw how to compute Fibonacci numbers ($\{F_i\}$) in parallel. Unfortunately the algorithm uses exponential work. Now let's use another algorithm to do it. Recall that $F_0 = 0, F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$.

(a) (4pts) First, define a vector $v_i = \begin{bmatrix} F_{i-1} \\ F_i \end{bmatrix}$ to denote the $(i-1)$-th and i-th Fibonacci number. How can we represent v_{i+1} using v_i (hint: write down the transition matrix)?

(b) (3pts) Write v_n using v_1. Then you'll be able to write v_n using any v_i where $i < n$.

(c) (8pts) Give a parallel algorithm to compute $F_1, F_2, \ldots, F_n$. Your algorithm must take $O(n)$ work and $O(\log n)$ depth. Please prove the cost of your algorithm.

5 Programming (30 pts)

You can use the server provided by the department, or any other multicore machine that you have access to (better with more than 10 cores).

5.1 Log in to Machine ti-05

If you want to use the server provided by the department, here is the instruction.

(a) Log in to `bolt.cs.ucr.edu` using your **cs account**.

(b) From there, log in to ti-05:

```
ssh ti-05
```

(c) First, enable the compiler, binaries, library paths, etc. This needs to be done **every time** you log in to the system.

```
scl enable devtoolset-7 bash
```

If you want to use the Intel TBB libraries, use:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/src/tbb/build/
   linux_intel64_gcc_cc7.3.1_libc2.17_kernel3.10.0_release/
```

(d) Then you can test your own code using cilk or TBB.

Note: To get the best performance and collect reasonable final data, you may want to have all cores available. Do not wait until the last minute, since debugging and testing needs time, and also the machine might be very busy before homework deadlines.

Remember to back up all files you have in this machine. It will be repurposed for some other for Spring quarter.

5.2 Downloading Testing Code

In `https://github.com/syhlalala/paralgocode`, you can find sample code using PBBS scheduler and cilk.

To download the code, use:

```
git clone --recurse-submodules https://github.com/syhlalala/paralgocode.
   git
```

You will find that the PBBS library under pbbssample directory is a sub-repository from the public PBBS library.

Under each directory there is a separate makefile to compile all the code.

Note: These samples **do not** give you satisfactory performance. Finally in this problem you will need to write your own version of these algorithms to get good performance.

5.3 Answer the Following Questions

1. (5pts) How many cores do you have in your tested machine? How large is the L1, L2, L3 cache? How many hyperthreads can you use?

2. (5pts) Test the reduce and scan code in the repository. You can use either cilk or the PBBS scheduler (or both).

   Usually you want to collect the following data:

   (a) The sequential running time of the algorithm (i.e., adding them one by one sequentially). Compare it with the running time of your parallel algorithm running on one core.

   (b) Change the number of threads (usually 1, 2, 4, 8, ...) and see the scalability curve of running time.

   (c) Test the performance of different input sizes.

   Here are some other thing that you can also try. These may not make much difference for testing the reduce or scan algorithms, but in general they are useful experimental settings designed to test the performance of an algorithm.

   (a) Test the performance using different settings/languages, e.g., using cilk, PBBS scheduler, OpenMP, etc.

   (b) Test different data types (int, float, ...).

   (c) Test different input distributions (uniform, Zipfian, exponential, ...).

   (d) Add optimizations, one at a time, and show performance improvement using each optimization.

   To change the used threads in cilk, change the variable CILK_NWORKERS. For PBBS, change NUM_THREADS. For example, in cilk, using

   ```
   export CILK_NWORKERS=1
   ```

   to only use one thread in the computation.

3. (10pts) **Granularity Control.** The reduce algorithm may have unsatisfied performance, especially when you have many cores and small input sizes. This is because scheduling (forking and joining threads) causes overhead, which is significant when the input size is small. A simple trick to tackle this is to control the parallelism granularity (also called *coarsening*). When the size is small enough, we stop doing recursive calls, but directly add them up and return.

```
int reduce(int* A, int n) {
  if (n < threshold) {
    int ret = 0;
    for (int i = 0; i < n; i++) ret += A[i];
    return ret;
  }
  int L, R;
  L = cilk_spawn reduce(A, n/2);
  R = reduce(A+n/2, n-n/2);
  cilk_sync;
  return L+R;
}
```

The appropriate threshold depend on the platform and the tested environment. Next you need to test the new code with granularity control, and find a good threshold. Write down your approach for finding the best parameter.

4. (10 pts) Finally, try to implement an efficient scan (prefix sum) algorithm. Explain your code and your optimizations. Again, you need to design experiments to test the performance of your code. You can use the sample code in the repository as a reference, but again, that wouldn't directly give you good performance.