CS260 Assignment 0

March 27, 2020

You need to submit through iLearn by 23:59 April 10. 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).

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.

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.

3 Answer the Following Questions

1. 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. 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. (Optional) **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. (Optional) 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.