CSCE 435 Spring 2020

HW 2: Parallel Programming with Threads II

Due: 11:59pm Friday, February 14, 2020

Minimum of a List

You are provided with a program `list_minimum.c` to compute the minimum element of a list using threads. The main program initializes data structures and calls `pthread_create` to create threads that execute the function `find_minimum`. Each thread is assigned a sublist ranging from `my_start` to `my_end`, and is responsible for computing the minimum value in this range. The thread is also responsible for updating the global variable `minimum` to ensure that the global minimum is no larger than the minimum value in its sublist. **The program is missing code to update** `minimum`**.** You need to add code to the `find_minimum` routine so that the program computes the minimum correctly. Access to `minimum` is controlled by the mutex `lock_minimum`. The variable `count` has been initialized to zero in the main program, and can be used to determine how many threads have accessed `minimum`.

Once you complete the code, it can be compiled with the command:

```
icc -o list_minimum.exe list_minimum.c –lpthread –lc -lrt
```

Note the use of `–lc` to link the library that provides the random number generator. Also note that `–lrt` links to the real time library that provides a higher resolution timing function. To execute the program, use

```
./list_minimum.exe <n> <p>
```

where <n> represents the number of elements in the list and <p> represents the number of threads. The output of a sample run is shown below.

```
./list_minimum.exe 100000 8
Threads = 8, minimum = 18074, time (sec) =   0.0006
```

1. (20 points) Complete the function `find_minimum` so that the program computes the minimum of the list correctly. *You will get full points if you add code only to the* `find_minimum` *routine*.
2. (10 points) Execute the code for n= 2×10^8 with p chosen to be 2^k, for k = 0, …, 13. Plot execution time versus p to demonstrate how time varies with the number of threads. Use logarithmic scale for the x-axis. Plot speedup versus p to demonstrate the change in speedup with p.
3. (10 points) Give reasons for the observed variation in execution time as p is varied from 1 to 2^{13}.

Barrier

You are provided with a program `barrier.c` that has code to implement a barrier for a multithreaded program. The main program initializes data structures and calls `pthread_create` to create threads that execute the function `work` before calling the barrier function. The file `csce435.h` includes `work` that forces a thread to sleep for a specified time before returning. **The program is missing code for the barrier function called** `barrier_simple`**.** You need to add code to `barrier_simple` to implement a barrier among the threads. Threads should wait for all

threads to enter the barrier routine before they are allowed to exit. The variable `count` can be used to determine how many threads have entered the barrier. You may use the mutex `lock_barrier` to allow a thread to obtain exclusive access to `count`. The condition variable `cond_barrier` can be used to have threads wait to receive a signal when `count` reaches a specific value.

4. (20 points) Complete the function `barrier_simple` to implement a barrier among the threads. *You will get full points if you add code only to the* `barrier_simple` *routine*.
5. (10 points) Execute the code for $p=2^k$, for k = 1,…,14. Plot execution time versus p to demonstrate how time varies with the number of threads. Use logarithmic scale for the x-axis.
6. (10 points) Set `sleeptime.tv_sec=0` and `sleeptime.tv_nsec=0` in `csce435.h` and run the experiments from the previous step again. Plot execution time versus p. How would you characterize the growth in time with p? What is the reason for such a growth?

List Statistics

7. (20 points) Modify the program in `list_minimum.c` so that it computes the mean and standard deviation of the list elements instead of the minimum. Name the new program file `list_statistics.c`. You may define global variables `mean` and `standard_deviation` that will store the values. *You will get full credit only if the mean and standard deviation of the list are computed by threads before exiting the thread routine*.

Submission: You need to upload the following to eCampus as a **single zip file**:

1. Problem 1: submit the file `list_minimum.c`.
2. Problem 4: submit the file `barrier.c`; you do not need to submit `csce435.h`.
3. Problem 7: submit the file `list_statistics.c`.
4. Problems 2, 3, 5, and 6: submit a single PDF or MSWord document with your responses.

Helpful Information:

1. Source files are available on the shared Google Drive for the class.
2. You may use either ADA or TERRA for this assignment. Indicate which machine was used in your experiments.
3. Load the compiler module prior to compiling your program. Use:
   ```
   module load intel/2017A
   ```
4. Compile C programs using `icc`. Link the `pthreads` library when using pthreads. For example, to compile `code.c` to create the executable `code.exe`, use
   ```
   icc -o code.exe code.c -lpthread
   ```
5. The run time of a code should be measured when it is executed in dedicated mode. Create a batch file and submit your code for execution via the batch system on the system.