CS416 Project 3: Scalable Parallel Counters

Due: 11/16/2021

Points: 100

Description

Counting is perhaps the simplest and most natural thing a computer can do. However, counting efficiently and scalably on a large shared-memory multiprocessor can be quite challenging. Furthermore, the simplicity of the underlying concept of counting allows us to explore the fundamental issues of concurrency without the distractions of elaborate data structures or complex synchronization primitives. Hence, counting is a non-trivial problem widely used in operating system kernels and also user-space application.

In the last project, you implemented a user-level thread library; but it does not fully explore the multi-core CPU resources in modern systems. In this project, you have the opportunity to explore parallel programming in a multi-core system. You are required to (1) implement 4 versions of parallel counters specified in Part 1; (2) write an report that explains the running behaviors of those 4 version of counters, and answer the questions specified in Part 2.

Part 1 Implementation

In Part 1, you are required to implement the following 4 parallel counters. In each implementation, the counter allows multiple threads (POSIX pthread library) to increment the counter by 1 with a large number of iterations. And after all of those threads finishes. You need to check whether the counter gives the correct or approximate correct value. You are given a skeleton code. Each implemetation of the following 4 counters has its own C file.

Here are the requirements:
1) There's a shared counter (initialized to be 0) amoung all threads.
2) Each threads are incrementing the counter by 1 and iterate with (1UL << 24) times (16777216 in decimal).
3) After all threads are done, you need to call printf() to show the run time of your counter, the value of the counter, and what is is supposed to be. 4) Your program need to take one argument which is the number of threads

Here's one example of the desired format when calling your program:

```
./naive_counter 4
```

Here's the desired output of running 4 threads:

```
Counter finish in 189.025146 ms
The value of counter should be 67108864
The value of counter is 17316058
```

1.1 Naive Counter

The simplest way is to just increment the shared counter by 1 in the loop for each running threads.

1.2 Naive Counter Plus

Is the naive counter gives the correct output? If not, why is it not giving the correct output? And what could you do to make it correct?

(**Hint:** think about the synchronization mechanisms you implemented in last project.)

1.3 Atomic Counter

What is the performance of Naive Counter Plus compared with Naive Counter? What could we do to reduce the synchronization overhead in software-level? operations like “counter++” is not a single machine instruction though it is just 1 statement in C language. It is a *read-modify-write* operation; that means you need to read the value first, increment it by 1 in register, and then write the new value back.

(**Hint:** As the name suggest, the atomic counter is using atomic operations.)

1.4 Scalable Counter

What would be the performance of an Atomic Counter? If it is still not good, what would be the reason for the performance issue? Modern systems usually have over 100 CPU cores, so we need a really quick and scalable counter in most of the scenarios. Please note that you CANNOT simply make a per-thread counter and sync them together when finish. That is because, when some threads are counting, other threads will also be reading the counter at the same time. Hence you cannot simply implement a per-thread counter, because then every read to the counter, you need to sum everything up, which will make the performance terrible. Now you need to think how to make the counter scalable.

(**Hints:** In most of the real-word senarios, the counter need NOT to be exact accurate, it allows a minor error bound. For example, if the counter will be always over 1000000, then a +-50 error will usually not a big problem.)

Part 2 Report and Questions

The Part 1 in this project seems to be not difficult. But this time, we want you to really think through the fundamentals behind the above 4 implementations of the parallel counters. So in this part, you need to finish the following contents in a report:
1) For each counter implementation, you need to plot the result running with different threads on iLab machines. (y-axis should be the running time (ms), x-axis should be the number of threads (1, 2, 4, 8, 16, 32))
2) You need to think in deep and answer the questions specified below:

Question 1: Why is Naive Counter has huge difference from the correct value?
Question 2: Why would Atomic Counter be superior to Naive Counter Plus?
Question 3: Why is Atomic Counter still not able to achive better performance than Naive Counter?

Please put your report in the pdf file, and specify which iLab machine you are using for getting the experiment results.

Useful Links

Cache Coherent protocol

`http://meseec.ce.rit.edu/551-projects/fall2010/1-3.pdf`

GCC built-in functions for atomic memory access

`https://gcc.gnu.org/onlinedocs/gcc-4.6.3/gcc/Atomic-Builtins.html`

Submission

To submit your assignment, simply submit the following files as is, directly to Canvas as is. (Do NOT compress the files! Otherwise I will reduce your credits):

1. naive_counter.c
2. naive_counter_plus.c
3. atomic_counter.c
4. scalable_counter.c
5. Makefile (even if you didn’t modify it)
6. Report.pdf