Ruprecht-Karls University of Heidelberg
Institute of Computer Engineering

Holger Fröning
Bernhard Klein

Advanced Parallel Computing
Summer term 2019

Exercise 3

- **Return electronically until Tuesday, May 14, 9:00 am**
- **Include name on the top sheet.**
- **A maximum of two students is allowed to work jointly on the exercises.**

3.1 Reading

Read the following two papers and provide reviews as explained in the first lecture (see slides):

- David A. Wood and Mark D. Hill. 1995. Cost-Effective Parallel Computing. Computer 28, 2 (February 1995), 69-72. DOI=http://dx.doi.org/10.1109/2.348002
- P. Stenstrom. A survey of cache coherence schemes for multiprocessors. Computer, vol. 23, no. 6, pp. 12-24, June 1990. doi: 10.1109/2.55497

(25 points)

3.2 Shared counter

Implement a Pthread program which performs the following calculation: a counter (initialized to zero) has to be incremented **C** times, so that at the end of the program it has the value **C**. A configurable number of threads (**N** threads in total) shall cooperatively increment the counter. Use a mutex to ensure mutual exclusion when incrementing the counter. Before the threads start incrementing the counter, all threads have to synchronize using a barrier. This avoids the visibility of start-up effects and maximizes contention. **N** and **C** should be command line parameters of your program.

In pseudo-code, each thread should do the following:

```
Barrier();
for (i = 0 to (C/N)-1) {
  Mutex_Lock();
  inc(counter);
  Mutex_Unlock();
}
```

Develop a suitable program, first without the mutex. Depending on a sufficient large **C**, data races should result in a mismatch between counter and **C** at the end of the program. Reproduce this behavior. Provide an interpretation why this is dependent on the value of **C**.

Now, implement the program using a mutex to protect the critical section. For a varying C, ensure that after execution the counter matches **C**, i.e. there are no race conditions anymore.

Conduct your experiments on `moore`.

(10 points)

3.3 Shared counter revisited

Start with the program from the previous exercise. Implement two alternative counter update methods and answer the questions:

1. Use an atomic operation to increment the counter. Obviously, now no locking is required anymore (why?). For gcc, the following intrinsic can be used:

   ```
   uint32_t var = 0;
   __sync_add_and_fetch(&var, 1);
   ```

   In this example, variable var is atomically incremented. Feel free to use any other atomic intrinsic.

2. Now, use the atomic operation (or another suitable one) to implement your own locking mechanism. I.e., implement a `lock_rmw(*lock)` and an `unlock_rmw(*lock)` function, by relying on atomic operations. Why are atomic operations required in this case?

Develop your programs and perform initial testing on moore. Validate the correctness of these two new programs with the same methodology as in exercise 3.2 (for a varying **C** and **N**, ensure that after execution the counter matches **C**, i.e. there are no race conditions anymore).

(25 points)

3.4 Shared counter performance analysis

Now, measure the overall execution time using suitable functions (e.g. `clock_gettime` or `gettimeofday`). Report the overall execution time and the derived number of updates per second for a varying number of threads (1-48) and sufficiently large number of updates (providing stable results). For this experiment, use the computer `moore` (48 cores in total). As `moore` is often heavily used, please ensure that you only use it for performance experiments.

	PTHREAD MUTEX		ATOMIC INCREMENT		LOCK_RMW	
Thread count	Execution time	Updates per second	Execution time	Updates per second	Execution time	Updates per second
1						
2						
4						
8						
12						
16						
24						
32						
40						
48						

Run this experiment with all four implementations. Report results in the table above, include appropriate units (ns, us, ms, s). Include a graphical representation here (varying number of threads on x-axis with updates per second on y-axis). Interpret your results.

(15 points)

Total: 75 points