Name – Atharva Nandanwar
Lab 2 Write up

Code Organization

- Code consists of two programs – mysort & counter
- Code is separated as follows:
  - **Main**
    This chunk of code handles argument parsing, reading the numbers from the file, dumping sorted list into an output file, and integrating the lock and barrier functions with bucketsort.
  - **Bucketsort**
    This functionality is similar to Lab 2. Bucketsort with two step parallelization for splitting the array into buckets and sorting the individual buckets. It uses function pointers lock, unlock & barrier_wait for splitting and sorting threads. These functions are assigned appropriate pointer by the main function.
  - **Locks**
    - **TAS Lock**
    - **TTAS Lock**
    - **Ticket Lock**
    - **MCS Lock**
    - **Pthread Lock** – uses global pthread_mutex_t
  - **Barrier**
    - **Sense Reversal Barrier –** barrier_custom_wait function emulates that functionality
    - **Pthread Lock –** uses global pthread_barrier variable
  - **Counter**
    This chunk of code takes care of argument parsing for counter program. Similar functionality of function pointers is used assign appropriate lock functions for provided arguments.

Files

- main.c
- bucketsort.c
- bucketsort.h
- locks.c
- locks.h
- barriers.c
- barriers.h
- counter.c
- Makefile

Compilation Instructions

make → mysort
make counter → counter
make clean → cleans the the build

Execution Instructions

counter [--name] [-t NUM THREADS] [-i=NUM ITERATIONS] [--bar=<sense,pthread>] [--lock=<tas,ttas,ticket,mcs,pthread>] [-o out.txt]
mysort [--name] [source.txt] [-t NUM THREADS] [--bar=<sense,pthread>] [--lock=<tas,ttas,ticket,mcs,pthread>] [-o out.txt]

Performance Report

Performance Analysis was done by `perf` tool on local machine with 6 cores and 12 threads.
For sorting program – mysort
Number of threads – 10; Input file – 0-2000000 shuffled

Lock	Barrier	Run Time (sec)	Page fault	L1 Cache Hit Rate	Branch Prediction Hit Rate
TAS	Sense	1.299219814	6063	96.85%	97.18%
TTAS	Sense	0.670983529	6076	97.84%	96.99%
Ticket	Sense	0.608127613	6076	98.99%	98.55%
MCS	Sense	0.602277158	6076	99.51%	99.02%
Pthread	Sense	0.558191197	6074	97.04%	97.43%
TAS	Pthread	1.180993499	6068	96.22%	96.86%
TTAS	Pthread	0.661806725	6064	97.63%	96.56%
Ticket	Pthread	0.626145505	6063	97.86%	98.40%
MCS	Pthread	0.556556490	6062	99.47%	98.94%
Pthread	Pthread	0.564335232	6062	96.13%	97.19%

For counter program – counter
Number of threads – 10; Number of iterations – 100000

Lock	Run Time (sec)	Page fault	L1 Cache Hit Rate	Branch Prediction Hit Rate
TAS lock	3.381765644	88	86.44%	95.87%
TTAS lock	1.915642662	89	96.84%	96.24%
Ticket lock	1.200277490	91	98.91%	99.58%
MCS lock	0.999836476	92	99.57%	99.81%
Pthread lock	0.791771780	88	91.50%	97.33%
Sense reversal barrier	0.540498727	99	98.01%	98.39%
Pthread barrier	15.068676847	107	88.76%	97.85%

Lock Algorithms & Results of Performance Report

- **Test and Set Lock**
  Test and Set lock checks for the value of lock, and then decides on the action to take. For convention, let's assume that acquiring the lock is setting boolean value to 1, and releasing the lock is resetting the boolean value to 0. Test and Set lock sees if the boolean is 0, and sets it to 1 for acquiring the lock. If the boolean is 1 (meaning someone has the lock) it spins on it until it is 0. It uses atomic operation exchange and return to read and change the value of an atomic variable. TAS lock has the worst cache hit performance in all the locks (excluding pthread). There is a lot of contention in the cache line due to each thread wanting a modifiable value of the lock boolean. This lock is also starvation prone, as the thread which released the lock is most likely to get the modifiable copy for the boolean. Branch prediction rate seems to be worse than other locks as well. Page fault is lowest for TAS lock, however run time is very high. A reason for run-time high would be that each thread is going to spin wait for other thread while also have cache misses for their lock booleans.
- **Test and Test and Set Lock**
  Test and Test and Set lock is a variation of TAS Lock. Instead of spinning and waiting for modifiable copies, we spin on local read-only copies of cache line and that makes cache hits very less. This can be seen from the

performance report. TTAS Lock reads the value for lock boolean atomically and further tries to execute the atomic exchange operation to set the value. However, this doesn't request for modifiable copy of cache line when the lock is acquired by other thread. Here, the cache hit performance has been improved. We have also decreased the run time of the program. Interesting thing to note is that the branch prediction hit rates are worse between TAS lock and TTAS lock in mysort program, but branch hit rates are improved in counter program. My understanding is that this is caused by the primary operation we are conducting in both the programs.

- **Ticket Lock**
  This is a FIFO lock, that assigns ticket numbers to each thread that tries to acquire the lock. The locks are held by the threads waiting for their turn to arrive. This lock is more fair lock. However, this lock causes a lot of cache traffic as we are trying to change the lock and every thread would have cache miss. We can see this when we compare the cache hit rates for MCS lock and Ticket lock. The resultant time with FIFO locks have reduced, this is probably because of the nature of workload. Important thing to note is that with FIFO locks we have significant delay in resultant operation when we have thread count more than the number of hardware threads. This happens due to less hardware trying to communicate between these threads as they are scheduled in and out of the hardware threads.
- **MCS Lock**
  MCS Lock has better cache performance among all the locks. This lock is FIFO lock and also poses delay in resultant operation due to less hardware threads trying to synchronize accesses between threads. MCS Lock offers better cache performance because of the design, as we are waiting on thread-local copy and global copy which is read only. This helps us improve the cache performance of the lock. Being a FIFO lock, this lock is not prone to starvation.
- **Sense Reversal Barrier**
  Sense Reversal Barrier exhibits the fastest operation. This was a surprise as opposed to pthread barrier which takes a lot of time for execution. Compared to pthread barrier, sense reversal barrier has significant performance improvements.
- **Pthread Locks and Barriers**
  Out of all the locks and barriers, pthread locks and barriers have worst performance in terms of cache hits and branch prediction hits. However, when other perf data was observed, it was evident that pthread locks and barriers consumed a lot less processing cycles.

Extant Bugs

- Running `./mysort <sourcefile>`, it would give a SIGSEGV. `./mysort` without input file gives segmentation fault.
- There is an unknown error when providing numbers more than 2000000 causes the program to shut down with a segmentation fault.
- Running the program with valgrind gives errors. I have not been able to take a look at them.
- Having inputs larger than INTMAX would cause segmentation fault.