Homework 2 Theory & Writeup

Michael Tang

due May 2, 2019

1 Theory

1.1 Exercise 25

No. L1 requires that every method call has its response immediately after. Essentially, changes done by method must be nonoverlapping and sequential by time. Sequential consistency only requires that method calls appear to execute in a sequential order. This is a looser requirement that doesn't regard time intervals.

1.2 Exercise 29

No; the property does not guarantee that each method call only takes a finite amount of steps. With infinite calls having infinite steps, some calls may still have infinite steps.

1.3 Exercise 30

No; nowhere in the property is there a guarantee that any method calls take finite steps. They could all take infinite steps.

1.4 Exercise 31

Neither; the bound stated is dependent on the number of calls. As the number of calls reaches infinity, so does the number of steps per call. So, it cannot be ensured that every method call takes finite steps and the implementation is not wait-free.

1.5 Exercise 32

For Line 15:
If Thread A and B call *enq*() concurrently and Thread A executes Line 15 before Thread B, Thread B can still exit *enq*() first if it finishes Line 16 first. So, the order on execution of Line 15 does not bind the order of finishing the call to *enq*() and Line 15 is not a valid point of linerarization.
For Line 16:
If Thread A and B call *enq*() concurrently and Thread A executes Line 16 before Thread B, Thread B still could have executed Line 15 first, which puts Thread A after Thread B in the queue. When both finish their call of *enq*() it would be the same result as if Thread B executed *enq*() before Thread A with

no concurrency. So Line 16 is not a valid point of linearization either.
If there is no single point in *enq*() which can be linearized, *enq*() cannot be linearized barring a lock on the entire call.

2 Writeup

2.1 Usage

Use make or make all to compile and link pacver.c and the required modules. Modules must be, relative to the working directory, in the path "../utils", which they are in my svn repository. make clean is also available to remove .o files.
Execute the program as follows:
./pacver [-n *nsources*] [-T *npackets*] [-D *qdepth*] [-W *work*] [-s *distribution*] [-m *opmode*] [-o *outdir*]
-n specifies number of sources, which in parallel mode translates to number of threads **besides the dispatcher thread**. This means for the performance experiments specifications of n were actually n-1.
-T specifies number of packets per source.
-D specifies queue capacity in packets, default 32.
-W specifies mean work per packet.
-s specifies work distribution mode between constant (0), uniform (1), and parallel (2). Default 0.
-m specifies operation mode between serial (0), serial-parallel (1), and parallel (2). Default 0.
-o specifies the path to write output to. Default stdout.

2.2 Design Changes

There were no major design changes from the proposed design, except a solidification of where to use volatiles and the gcc atomic memory barrier, as well as a removal of the proposed *output()* function. The removal of that helper function was because output really only required one line of code in this assignment.

2.3 Testing Results

2.3.1 Correctness

The main invariants tested were:

- FIFO ordering
- Unique packets only being passed over one time
- All sources processing the specified amount of packets

The outputs parallel_constant, uniform, exponential_correctness.txt, serial_queue_correctness.txt, and serial_correctness.txt were generated from execution ./pacver -n 10 -T 10 -W 10000 with according distribution and operation mode arguments. These outputs were then fed into a Python 3 script, correctness_check.py to verify that packets came in order, were processed only once, and all packets enqueued were dequeued. All operation modes had to be checked, and for robustness also all work distribution modes (at least for parallel, where I judged that timing changes would likely have the most effect). All passed verification. All of these files can be found in hw2/test_outputs.
This testing was also based on the assumption that the checksum and packet generation operations were implemented correctly.

2.3.2 Performance

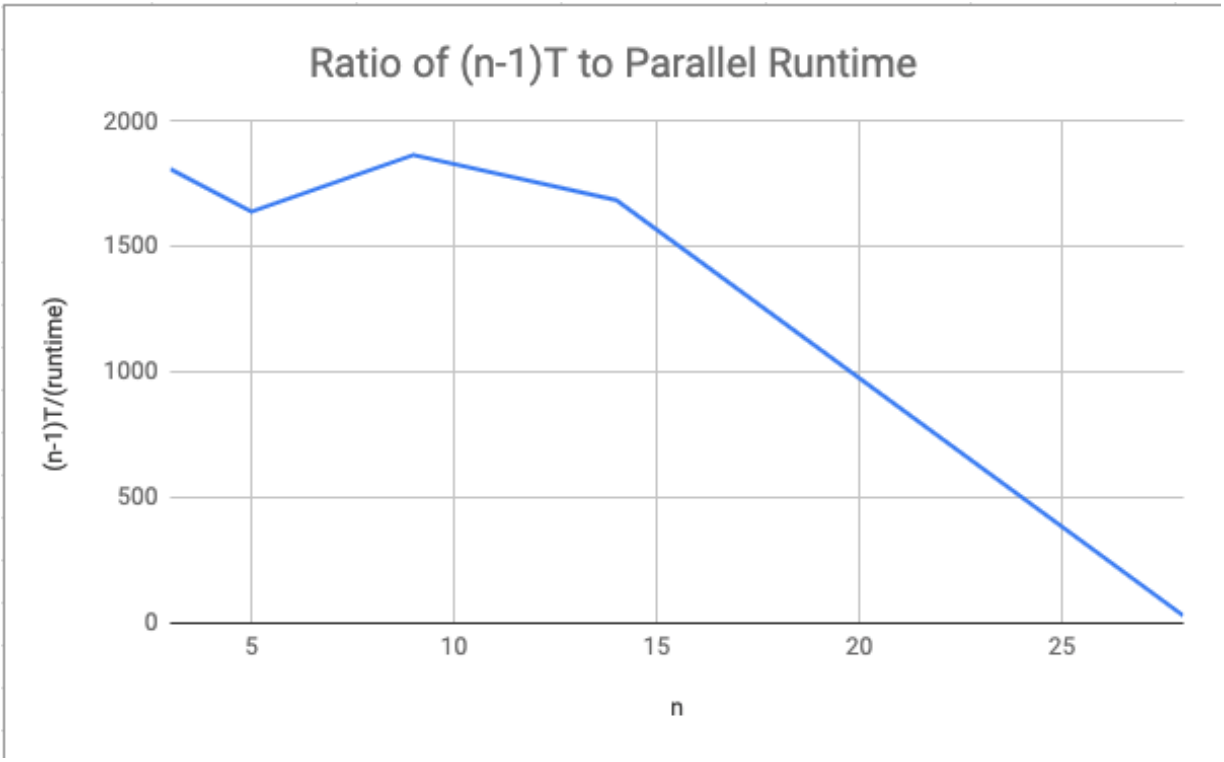
All tests were performed with SLURM as specified in the assignment directions.

Experiment 1

Experiment 2

The results are on the table and chart below.

N	Serial	Parallel (T=1)	Ratio of Parallel to Serial
16	0.075	0.234	3.12
32	0.572	0.889	1.554195804
64	4.454	3.663	0.8224068253
128	17.537	19.944	1.137252666
256	101.602997	108.249001	1.065411496
512	783.745972	803.745972	1.025518472
1024	6252.091797	6217.073242	0.9943989058



Experiment 3

Experiment 4
Experiment 5