Name - Pramod Kulkarni
Email - prku8035@colorado.edu

Performance of different locks and barriers on mysort for 10 threads and 5,00,000 input

Lock_type or barrier_type/Perf stats	Runtime (in seconds)	L1 cache hit rate	branch-prediction hit rate	page-fault count
TAS	4.033767s	98.87%	99.22%	2171
TTAS	1.115350s	99.80%	99.2%	2321
Ticket	1.732045s	99.73%	99.72%	2,178
MCS	1.317485	99.74%	99.72%	5,738
Pthread lock	0.374831	99.2%	99.1%	5,764

Sense	2.452937s	99.17%	99.14%	2,170
Pthread barrier	21.749916	89.57%	98.2%	2,171

Explanation : As per the expectation among TAS, TTAS, Ticket and MCS locks TTAS has very less runtime because same thread acquires the lock. Mcs is second fastest followed by ticket. Tas is supposedly the slowest among all these because it is inefficient lock. Pthread lock outperforms all of them. MCS has more cache miss rate compared to ticket and tas and pthread. MCS is faster as all threads get fair chance to make progress on their chunk.

Pthread_barrier is very slow compared to sense and that is expected output. Sense has better cache hit rate and branch prediction.

Performance of different locks and barriers on counter for 10 threads and 50,000 iterations

Lock_type or barrier_type/Perf stats	Runtime (in seconds)	L1 cache hit rate	branch-prediction hit rate	page-fault count
TAS	7.433232s	96.5%	99.6%	157
TTAS	0.133147s	98.23%	98.96%	156
Ticket	0.507068	99.38%	99.88%	157
MCS	0.208290	99.53%	99.9%	4064
Pthread_lock	0.238166	92.22%	98.6%	4066
Sense	1.456761	98.4%	98.95%	158
Pthread_barrier	25.393498	88.99%	98.2%	159

Pthread barrier performs the worst in counter case while sense is almost 25x faster. Among the locks pthreads beats all locks which is weird. TTAS has best runtime. MCS performs faster than Ticket. TAS being basic lock is bit inefficient and does very poorly.

The readings are observed just after one run. I expect things to perform as per the expectations if we take perf stat results and average them across 2000 iteration runs.

1.**Algorithms**

Irrespective of the algorithm the code first divides input into block sizes based on the number of threads supplied by command line. If num_threads = 1 whole input is processed by main thread and no chunks are created.

bucketsort_parallel- Number of buckets are decided based on size of input.

num_buckets = ln(nums.size())

1)For bucket sort each thread processes its sub-array/sub-vector. For each number it gets the prioroity_queue it belongs to and tries to acquire lock for it. If lock is acquired number is populated else thread waits for the lock.

2)Finally in the main thread all the priority_queues are concatenated, output of

which is a sorted array.

Counter_parallel –

If current iteration % num_threads is equal to the tid of the thread currently executing, then it increments the counter else waits. For synchronization have used all lock and barrier types mentioned in the lab document.

2. Code Organization

Abstractlock.h houses abstractLock class which has two virtual functions lock and unlock. Classes tas(in tas.h), ttas(in ttas.h) and ticket(in ticket.h) extend anstractLock class and define functions lock and unlock in their respective cpp files.

Mcs.h class has Node class's blueprint in it. Mcs class has two methods acquire and release. Pointer to Node class is member variable of mcs class.

Sense class has code for sense barrier and has one function wait.

Mysort.cpp

Calls process_command_line_arguments which processes command line inputs.

ii)calls parse_input_file and populates input numbers into nums vector.

iii)Calls create_nums_partitions which creates division indices and saves into

partition_indices structure.

iv)Times the calls to bucketsort and bucketsort_barrier. Selects appropriate synchronization primitive to sort the numbers based on command line arguments.

Outputs the final run time.

v)calls write_output_file and populates output file with sorted numbers.

Counter.cpp

i) Calls process_command_line_arguments which processes command line inputs.

ii) calls create_threads_and_call_counter or barrier_main based on the synchronization primitive specified in command line arguments.

iii)Both functions create thread and call counter function. Based on value in lock_type and barrier_type a synchronization primitive is selected.

3.A description of every file submitted

Mysort.cpp – main file for bucket sort function.

Counter.cpp – main file for counter program.

abstractLock.h – houses base class from which tas,ttas and ticket classes inherit.

Tas.cpp/tas.h – Houses the code tas lock.

Ttas.cpp/ttas.h – Houses the code ttas lock.

Ticket.cpp/ticket.h – Houses the code for ticket lock.

Mcs.cpp/mcs.h – Houses the code for mcs lock.

Sense.cpp/sense.h – Houses the code for sense barrier.

Commons.h – Has all the includes

Makefile – Has instructions to compile code for both mysort and counter.

4. Compilation instructions

cd to main directory and just type "make" that will compile and output a "mysort" and “counter” application.

5. Execution instructions

./counter -t 10 -i 50000 --bar=sense --lock=tas -o o.txt

./mysort input.txt -t 10 -o o.txt --alg=bucket --lock=mcs

./mysort input.txt -t 10 -o o.txt --alg=bucket –bar=sense

6. Any extant bugs

For mysort and barrier code have padded input with dummy numbers to evenly distribute across all threads. Finally after sorting padded numbers are removed.