Lab 2
Concurrent Programming

Name : Suraj B Thite

This Lab implements TAS, TTAS, Ticket , MCS , pthread locking mechanisms and pthread , sense barriers for synchronization between threads in execution for bucket sort developed as a part of Lab2. The performance of the locks is jotted down as below :

Code Organization :

The below tree describes the brief organization of the code for the Lab2 in SRC directory along with their descriptions.

SRC

|--**BucketSort**

|----main.cpp - Main Application File consisting sequential flow of operations.

- main()
- file_to_array()
- array_to_file()
- *fj_merge()
- get_bucket_range()
- *bucketSort()

|----main.h - header File consisting of function declarations and data structures implemented.

|----sorts.cpp - Source File consisting of functions for Merge sort and Quick sort on input data.

- mergesort()
- quicksort()
- partition()
- swap()
- merge ()

|----custom_locks.cpp – This is a library developed to implement custom locks namely TAS, TTAS, Ticket, MCS, Pthreads and Barriers namely Sense and Pthread.

- TAS_lock()
- TAS_unlock()
- TTAS_lock()
- TTAS_unlock()
- Ticket_lock()
- Ticket_unlock()
- Pthread_lock()
- Pthread_unlock()
- Pthread_barrier()

- sense_bar()
- class MCS
    - acquire()
    - release()

|----custom_locks.h – This is a header file which consists of declarations of custom locks and barriers required as a part of this assignment.

|----Makefile - Make file to build the project and application binary mysort.

|--**Counter**

|----counter.cpp – Counter Main Application File consisting sequential flow of operations.

- main()
- thread_main()

|----counter.h - header File consisting of function declarations and data structures implemented for counter.cpp.

|----sorts.cpp - Source File consisting of functions for Merge sort and Quick sort on input data.

- mergesort()
- quicksort()
- partition()
- swap()
- merge ()

|----custom_locks.cpp – This is a library developed to implement custom locks namely TAS, TTAS, Ticket, MCS, Pthreads and Barriers namely Sense and Pthread.

- TAS_lock()
- TAS_unlock()
- TTAS_lock()
- TTAS_unlock()
- Ticket_lock()
- Ticket_unlock()
- Pthread_lock()
- Pthread_unlock()
- Pthread_barrier()
- sense_bar()
- class MCS
    - acquire()
    - release()

|----custom_locks.h – This is a header file which consists of declarations of custom locks and barriers required as a part of this assignment.

|----Makefile - Make file to build the project and application binary counter.

|----**Makefile** - This make file internally builds the binaries for mysort and counter and copies them to the SRC parent folder.

Code Organization :

The code is organized in the SRC directory with Bucket sort and Counter implementing the sorting and micro benchmarking functionality, respectively. The make file in the SRC Directory compiles the binaries from both the folder and places it in the SRC directory. The functions implementing the TAS, TTAS, MCS , Ticket, Pthread locking and unlocking functionality along with sense-reverse are implemented in the custom_locks.cpp and custom_locks.h files. An array of function pointers is implemented to use the custom locking and unlocking depending the argument passed in the command line.

The same structure is used for the counter binary which implements a thread which increments a counter value protected by the locks and barriers depending upon the arguments passed by the user. As per the requirements either of one i.e. –lock or –barrier is processed by application. The final counter value is updated on the output file specified by the user.

Compilation Instructions :

Navigate to the SRC directory of the project and open a terminal.

Run make to compile the project and generate mysort and counter binary file in the SRC directory.

Run make clean to clean the application binaries from the project folder.

Execution Instructions :

Usage :

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

mysort [--name] [source.txt] [-o out.txt] [-t NUM THREADS] [--alg=<fj,bucket>]

[--bar=<sense,pthread>] [--lock=<tas,ttas,ticket,mcs,pthread>]

If -- name is passed as argument, the program is terminated.

All the arguments to be passed for mysort application i.e. both barrier and lock while either of them need to be passed for the counter application.

Failure to open the input file or write sorted data to the output file or incorrect no of arguments passed will stop the further execution printing appropriate messages on terminal.

A description of two parallelization strategies

Strategy to parallelize Bucket sort :

1. Set the bucket count equal to number of threads passed as arguments and determine the maximum value in the dataset to calculate the range of each bucket.
2. Depending upon the number of buckets , divide the input elements in equal halves and store it in an argument structure to be handled by each thread in execution.
3. A thread with bucket function handler and pass the structure argument to it for execution.
4. The handler assigns the elements in their sub-array to an appropriate bucket and then its joined in the main thread thus merging all the sub-array to the main array.

Locks and Barrier Performance stats on Counter Application

Locks	Run Time (s)	L1 Cache Hit Rate	Branch Prediction Hit Rate	Page Fault Count
TAS	0.0047	96.5 %	96.92 %	156
TTAS	0.0072	94.5 %	97.14 %	157
Ticket	0.0101	94.2 %	97.03 %	158
MCS	0.0086	93.9 %	97.02 %	159
Pthread	0.0184	94.2 %	98.53 %	158

Barriers	Run Time (s)	L1 Cache Hit Rate	Branch Prediction Hit Rate	Page Fault Count
Pthread	0.58	86.87 %	96.95 %	160
Sense	175.25	99.98 %	99.99 %	157

The above observations are made for 10 threads and 1000 iterations.

Locks and Barrier Performance stats for Bucket Sort Application

Locks	Run Time (s)	L1 Cache Hit Rate	Branch Prediction Hit Rate	Page Fault Count
TAS	0.8844	98.64 %	99.38 %	9698
TTAS	0.8132	99.05 %	99.78 %	9695
Ticket	0.8597	97.4 %	99.85 %	9893
MCS	0.9786	96.62 %	99.87 %	14794
Pthread	1.2337	94.53%	98.75 %	9697

Barriers	Run Time (s)	L1 Cache Hit Rate	Branch Prediction Hit Rate	Page Fault Count
Pthread	0.0028	92.59%	97.28%	161
Sense	0.0490	99.98 %	99.99%	164

The above observations are based on the 600 samples from input file passed as argument.

Observations :

1. As per the above analysis the pthread mutex takes most run time for same amount of load. The TAS and TTAS locks has lowest latency due to its simpler design and LIFO structure which most probably gives the lock to a thread which releases it leading to best L1 cache hit rate. The TAS and TTAS may lead to better cache hit rate but provide poor fairness and more starving amongst threads.

2. The MCS lock has the worst cache hit rate since it is a FIFO lock thus attributing to the longer execution time in case of more iterations / numbers to be sorted. The Ticket and MCS have more overhead of processing and switching thus taking up more space but providing all the threads fair share to acquire the lock on the basis of their arrival.

3. Since the data is fetched from the local cache, the similar performance is observed for bucket sort binary.

4. The sense barrier has higher cache hit rate compared to pthread barrier but it more time to complete the sorting process. Similar outcome is observed for the counter application.

5. The TAS and TTS has lower page faults than compared to MCS and Ticket Locks.

The best primitive in situation where fairness supersedes than timing and space requirements is the MCS or ticket lock since MCS lock has less timing overheard but more spatial overhead and vice versa for ticket lock.

In case of situations where there is lower timing requirement and fairness of a process is not required then TTAs is preferred over TAS since it reduces the cache invalidation and contention misses.

The pthread mutex lock can be chosen in an ideal scenario which provides linear timings and cache hit rates with linear increase in number of threads and functionalities.

Extant Bugs :

No bugs in the code as such, the project compiles successfully and passes the test case as mentioned in the assignment requirements. The code throws an error if wrong barrier other than pthread/sense is selected and takes TAS as default lock if no lock is passed.