OPERATING SYSTEMS

SHUFFLING IN NUMA ARCHITECTURE
TEAM 3

Objective

- To understand and implement the shuffling algorithm for the scheduling of threads with lock contentions in a Multi-core Multiprocessor System.
- To evaluate the shuffling framework mentioned above on a number of multi-threaded programs provided by the PARSEC benchmark.

Experimental setup

- The Perf Tool
    - **perf** is a performance analyzing tool in Linux.
    - **perf-lock** tool is used to find the lock times of the threads.
    - More specifically, we will be using
        - **Perf lock record** to start recording lock times from the given instant of time.
        - **Perf lock report** to read the data collected by perf-lock record
- The ***ps*** (i.e., *process status*) command is used to provide information about the currently running *processes*, including their PIDs,TIDs etc.
- **Set-affinity** command is used schedule a thread to a specific processor.

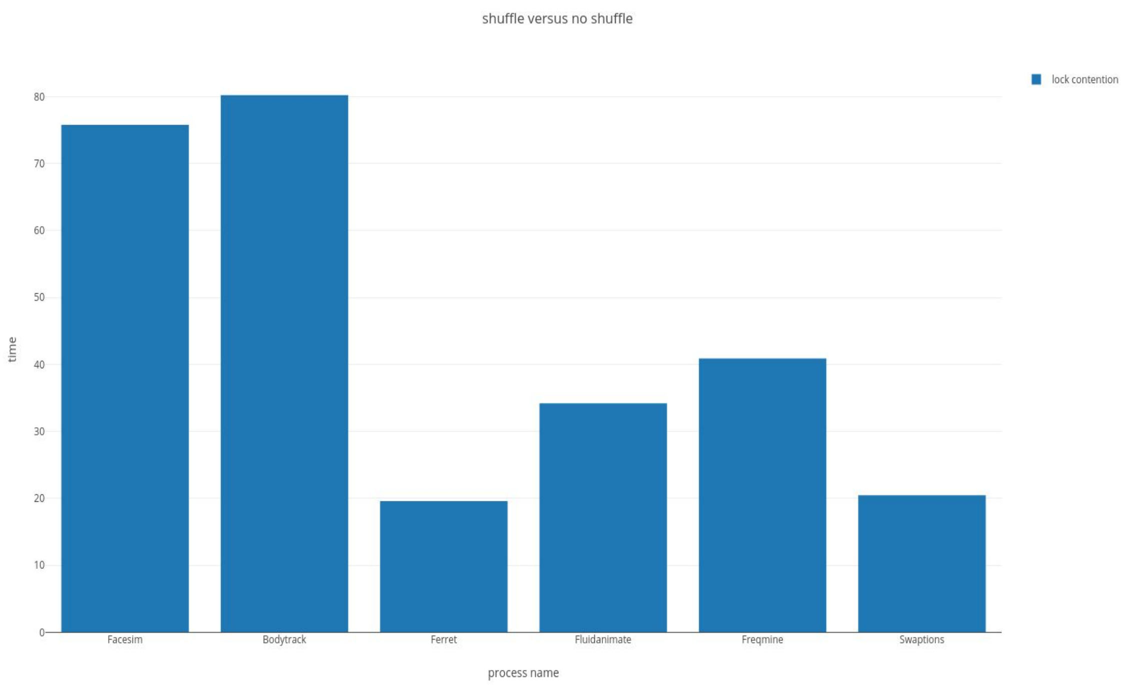
The Shuffling Algorithm

- Shuffling aims to reduce the variance in the arrival times of the threads scheduled on the same Socket by scheduling threads whose arrival times are clustered in a small time interval so that they can all get the lock without losing the lock to a thread on another Socket.
- But here our shuffling algorithm is based on thread lock-times as the calculation of the arrival times of the threads can be quite complicated.
- Shuffling is performed here by executing the following three steps repeatedly throughout the application's lifetime.
    - Monitoring of Threads

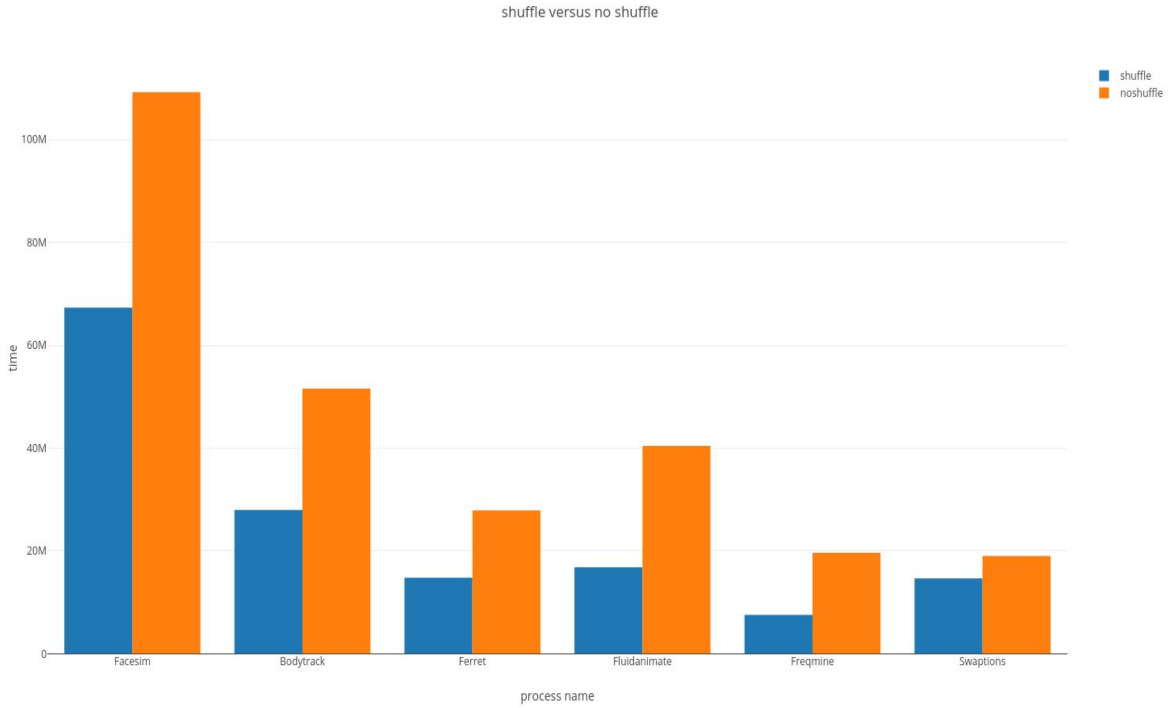
        - The fraction of execution time that each thread spends waiting for locks (Lock-Time) is monitored .
        - If this time exceeds a certain threshold(here, it is 10% of the elapsed time), then the next two steps of the algorithm are performed , else we jump to the next iteration which starts with monitoring of threads again.
        - Here the Lock-Times are recorded for an interval of 200ms.
    - Forming the Thread Groups
        - Threads that experience similar lock times will be placed in the same group.
        - Therefore we sort the threads according to their lock times and then divide them into many thread-groups.

    - Implement Shuffling
        - Threads are scheduled in cpu's cores consecutively based on the sorted thread-groups formed in the previous step.
        - Set-affinity command is used to realise the task specified in this step.

Plot of lock contention percentage with process



Plot of time taken with and without shuffling per process



Observations

- Shuffling (rescheduling of threads) is executed on an average of 15 - 20 times in the tested applications.
- Monitoring interval of 200ms gave the least value of lock time most of the times.
- Few applications have very high lock contention, one of them is facesim.

Results

- Shuffling step is executed larger number of times in applications which have higher lock contention.
- Greater improvement in lock times is seen in applications with high lock contention.
- Shuffling has no effect on Applications with lesser lock contention.
- An improvement upto 50% is seen in some applications.
- Very little improvement is seen in applications with lesser lock times.
- Smaller monitoring time is resulting in smaller lock time until 200ms.

Name	shufle1	shuffle2	noshuf1	noshuf2	Lock contention in noshuffle (%)
Facesim	67349267	69318236	79467004	109263805	75.8
Bodytrack	27978586	29623745	47873941	51593086	80.24
Ferret	14785308	14803749	25430922	27886364	19.61
Fluidanimate	16791306	25402971	38054571	40416899	34.2
Freqmine	7554782	12991124	14377865	19651438	40.9
Swaptions	14663977	15051494	18088587	19034136	20.47

Conclusion

- Shuffling framework implemented here demonstrates that programs with high lock-contentions are very sensitive to the distribution of threads across multiple cores in a Multiprocessor System,
- For programs in which shuffling is done many number of times, there is an added overhead of performing shuffling each time as well , which results in increase in the overall execution time. Thus this observation reinforces the importance of the right choice of sampling interval (time interval for collecting the lock times of the threads) that we chose. Same is the argument for a good shuffling interval (time interval between each application of shuffling) to be chosen, which affects the LLC misses in the critical path.
- We evaluated the shuffling framework on the above mentioned programs in the PARSEC Benchmark and we observe that there is up to 46% reduction in execution time with the average reduction in execution time being 36% (in accordance with the data collected for for the above mentioned programs)