CSE 5441
Programming Assignment 3
Submitted by: Biplob Biswas (biswas.102)

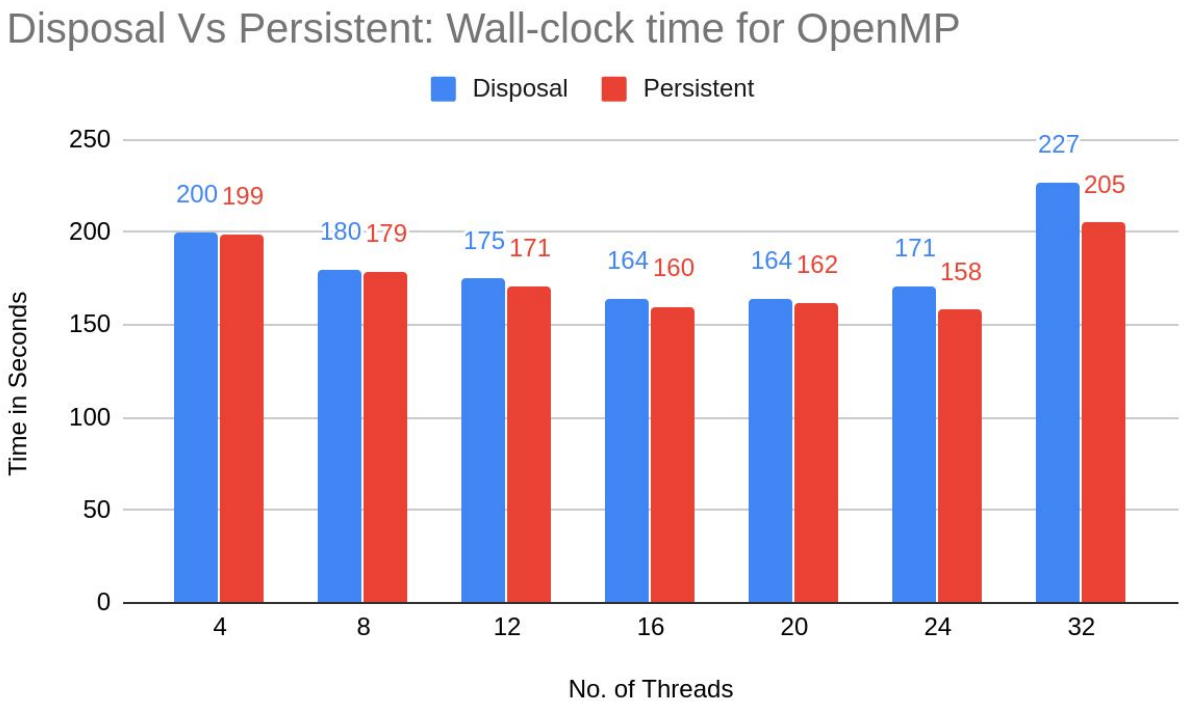
Setup and general observation

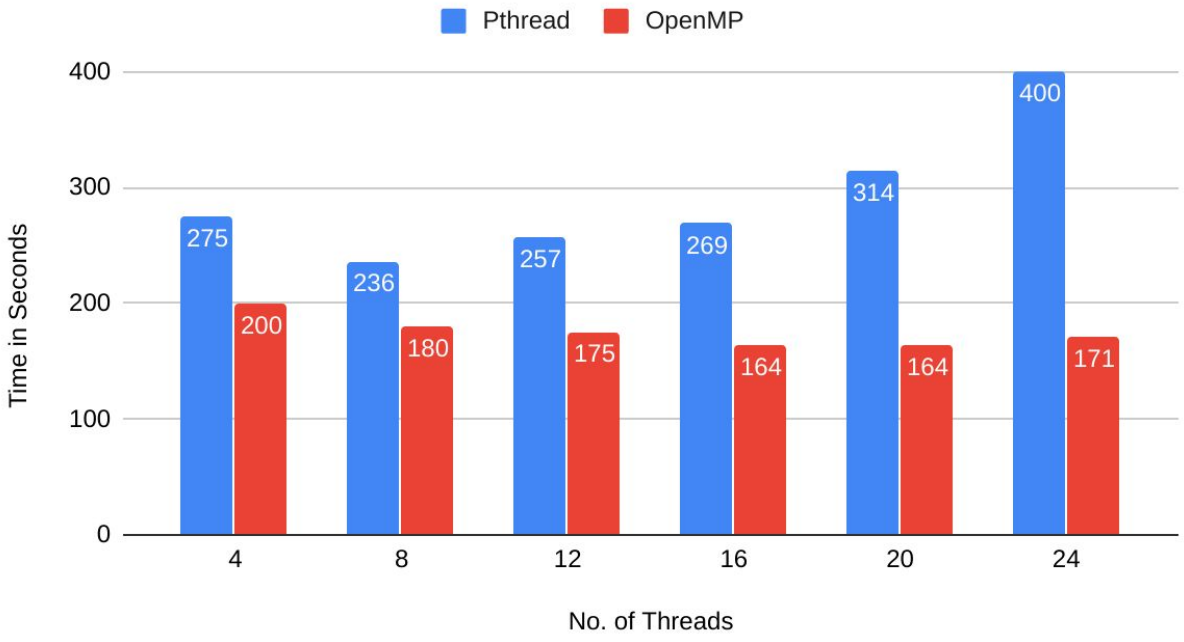
- All the tests were run with parameters `AFFECT_RATE = 0.02` and `EPSILON = 0.02`.
- All timing results are collected using the Unix `time` command.
- The program that ran for the longest time (`6m40.307s`) was the one using PThreads with 24 disposable threads.
- Each run of the program converged after `794818` iterations.
- OpenMP provided the exact number of threads requested every time.

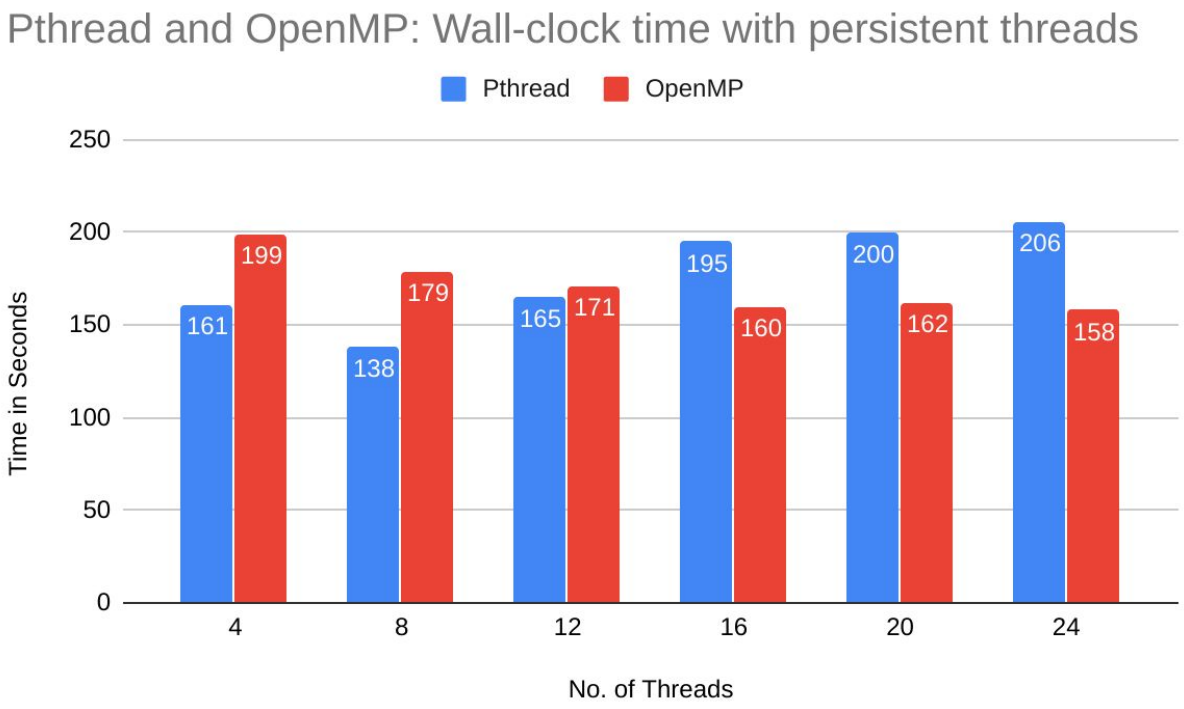
Timing results

Timing results for OpenMP:

No. of threads	Disposable			Persistent		
	real	user	sys	real	user	sys
4	3m19.888s	12m49.850s	0m29.438s	3m19.148s	12m48.000s	0m28.334s
8	3m0.164s	22m50.121s	1m10.759s	2m59.275s	22m43.377s	1m10.375s
12	2m55.298s	33m11.524s	1m51.657s	2m51.177s	32m24.194s	1m49.426s
16	2m43.752s	41m5.943s	2m32.244s	2m40.208s	40m11.286s	2m31.299s
20	2m44.374s	51m28.400s	3m17.595s	2m42.305s	50m53.156s	3m11.777s
24	2m50.955s	64m9.111s	4m11.575s	2m38.729s	59m34.017s	3m54.113s
32	3m47.656s	45m26.578s	60m38.094s	3m24.877s	41m0.050s	54m27.913s


Disposal Vs Persistent: Wall-clock time for OpenMP
Disposal
Persistent
250
200
150
100
50
0
Time in Seconds
200 199
180 179
175 171
164 160
164 162
171 158
227 205
4
8
12
16
20
24
32
No. of Threads


PthreadsVs OpenMP: Wall-clock time with disposable threads
Pthread
OpenMP
400
300
200
100
0
Time in Seconds
275 200
236 180
257 175
269 164
314 164
400 171
4
8
12
16
20
24
No. of Threads



Questions and Answers

- **How would you characterize the computational workload of our sample program?**

  The workload was very good for parallelizing, and the block sharing was good for better cache utilization. The thread synchronization part was the only bottleneck, which may have caused many threads to wait.

- **Which threading mechanism, pthreads or OpenMP, provided the best results in your case?**

  With disposable threads, OpenMP outperformed PThread by a great margin. The more the number of threads deployed, the better the performance was. For 32 threads, OpenMP program ran more than 50% faster than the PThreads program.

  However, for persistent threads, the performances of Pthread was better for smaller thread number, but with thread count higher than 12, OpenMP beats PThreads significantly.

- **Would you say that pthread is more flexible, less flexible or the same as OpenMP?**
  In Pthread, we can get an exact number of thread we specify, but in OpenMP, we may not have the number of threads we request. In this sense, Pthread provides more flexibility than OpenMP.
  On the contrary, OpenMP was the easiest to implement, without any doubt. It needed only a little modification in the serial version of the program. Using OpenMP, creating parallel and critical regions, setting up barriers and load distribution requires only a `#pragma` directive, while PThreads requires careful design and usage of the API functions.

- **How well did your OpenMP programs meet the design criterion of preserving your original serial program? What constructs did you use that would cause your program not to compile with OpenMP disabled?**

  OpenMP is meet the criterion of preserving my original serial program very well. Apart from the pragma directive, I did not use any such constructs that would cause my program not to compile with OpenMP disabled. Considering the ease of implementation and the performance, OpenMP would be preferable over PThreads. However, OpenMP does not provide any control over the number of threads. If this becomes a really crucial issue, PThreads would be preferable over OpenMP.

- **Observation:**

  1. Unlike PThreads, OpenMP did not offer significantly better performance from persistent threads than from disposable threads. With fewer number of threads their performances were almost the same. As the number of threads increased, the performance of persistent threads became only a little better.

  2. Another finding was the `sys` component of the timing results. Unlike PThreads, the disposable threads spends more time in kernel mode than persistent threads.