Build Instructions

To build the code run cmd 'make'

Note:

- makefile uses the icc (icc 18.0.3) compiler.
- makefile uses '-g -Wall' compiler flag and '-fopenmp' linker flag while compiling the file.
- makefile uses absolute path '*/fs/project/PAS1653/transform.o*' to link the obj file.
- The generated output file is called '*lab3_omp'.*

Implementation

Program uses multiple producers and multiple consumers and both read/write to a common queue. The producer threads read cmd-key pairs from the STDIN and call the corresponding transform method to encrypt the key and then insert the processed item in the common queue. The consumer picks items from the common queue and then calls the corresponding transform method to decrypt the encrypted key and then adds the output to the output queue.

Once all producer and consumer threads have completed, the main thread serially displays the output on STDOUT.

Running Times

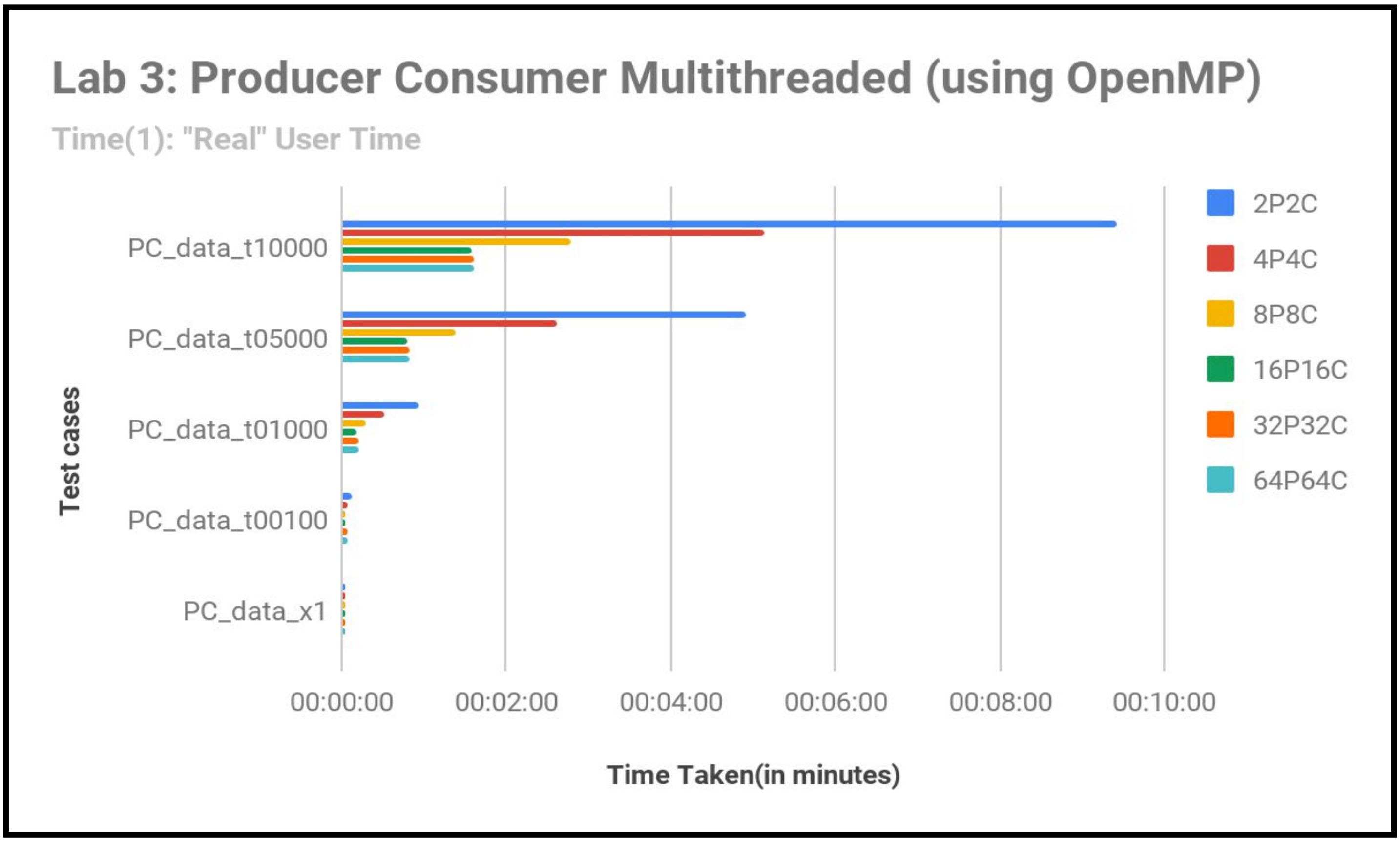
	Lab 3 (OpenMP)				Lab 1 (Serial)			
	Time(2)		Time(1)		Time(2)		Time(1)	
	P	C	Real	User	P	C	Real	User
PC_data_x1	2	16	00:01.713	00:13.766	2	1	0:03.585	0:03.513
PC_data_t00100	12	21	00:02.462	00:40.880	10	12	0:22.411	0:22.341
PC_data_t01000	148	162	00:10.662	04:24.239	110	95	3:26.070s	3:25.995
PC_data_t05000	749	763	00:48.497	21:55.943	518	535	17:33.385	17:33.147
PC_data_t10000	1498	1514	01:35.085	43:32.081	1045	1043	34:48.672	34:48.429

NOTE: Producer/Consumer time is in seconds and Real/User time is in mm:ss.ms format.

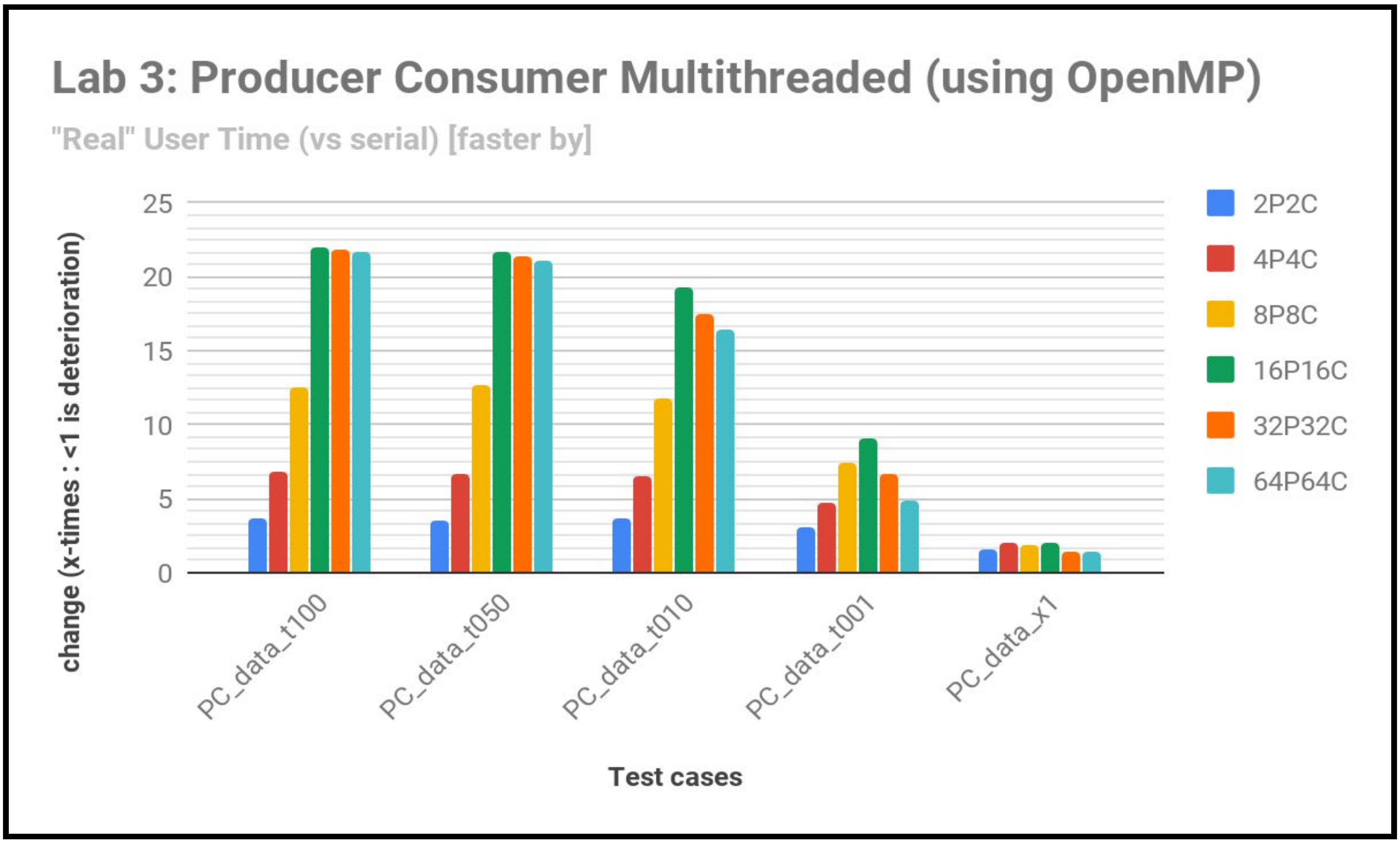
	Lab 3 (OpenMP)				Lab 2 (PThreads)			
	Time(2)		Time(1)		Time(2)		Time(1)	
	P	C	Real	User	P	C	Real	User
PC_data_x1	2	16	00:01.713	00:13.766	3	32	00:02	0:4.192
PC_data_t00100	12	21	00:02.462	00:40.880	15	32	00:02	0:27.158
PC_data_t01000	148	162	00:10.662	04:24.239	143	160	00:11	4:1.991
PC_data_t05000	749	763	00:48.497	21:55.943	736	750	00:47	20:27.571
PC_data_t10000	1498	1514	01:35.085	43:32.081	1456	1470	01:32	40:30.495

NOTE: Producer/Consumer time is in seconds and Real/User time is in mm:ss.ms format.

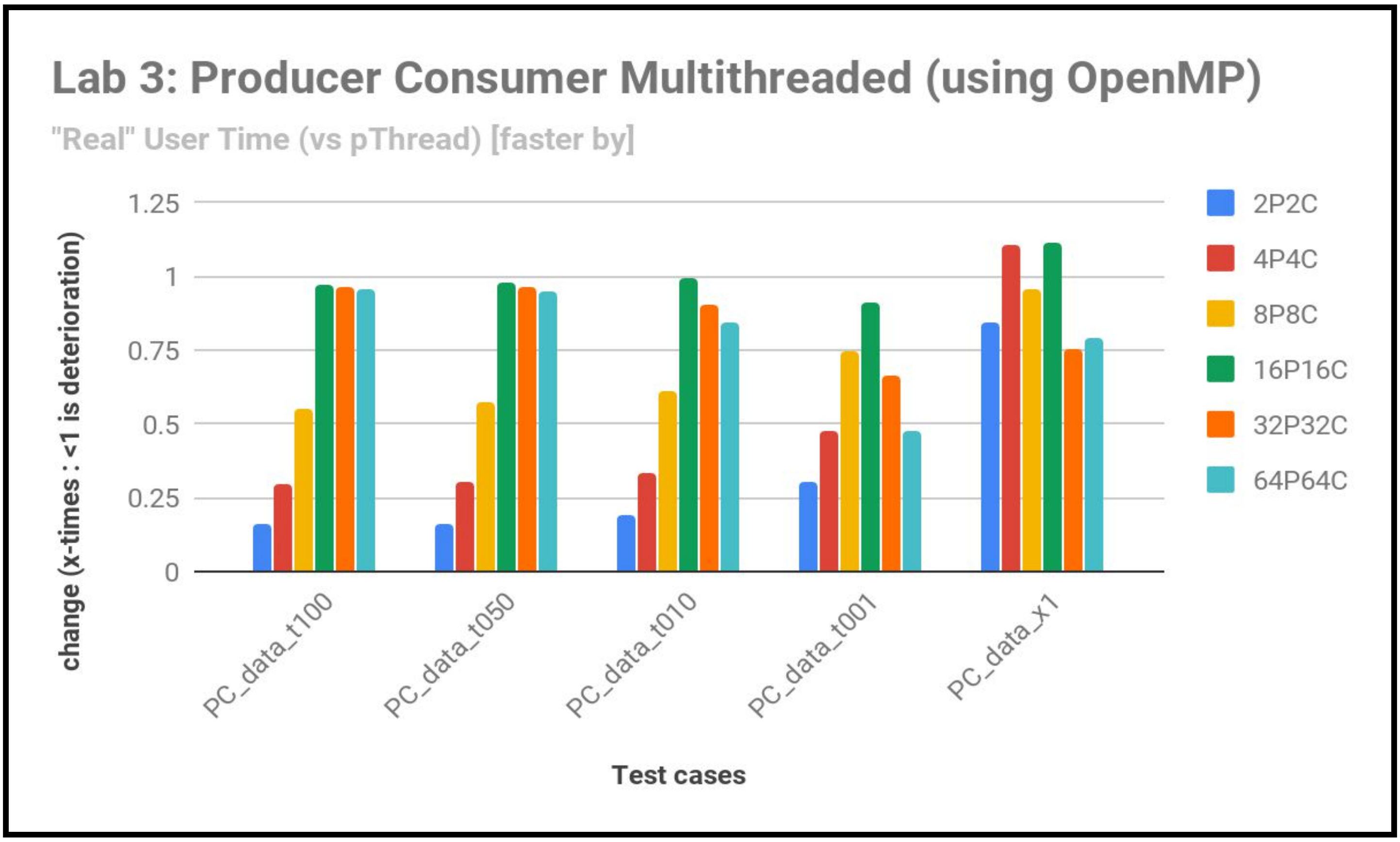
Scalability



Graph1: Performance of OpenMP implementation for various thread counts.



Graph2: Comparison of OpenMP implementation (different thread counts) with serial implementation.



Graph2: Comparison of OpenMP (different thread counts) with pThread(16P16C) implementation.

	Serial	pThread	2P2C	4P4C	8P8C	16P16C	32P32C	64P64C
PC_data_t10000	34:48.672	01:32.050	09:24.398	05:08.619	02:46.474	01:35.085	01:35.829	01:36.216
PC_data_t05000	17:33.385	00:47.439	04:55.166	02:36.839	01:23.159	00:48.497	00:49.204	00:49.874
PC_data_t01000	03:26.070	00:10.632	00:56.730	00:31.583	00:17.470	00:10.662	00:11.743	00:12.570
PC_data_t00100	00:22.411	00:02.237	00:07.344	00:04.683	00:03.012	00:02.462	00:03.353	00:04.663
PC_data_x1	00:03.585	00:01.910	00:02.266	00:01.724	00:01.991	00:01.713	00:02.528	00:02.421

NOTE: The above data is of Real time returned by Time(1) and is in mm:ss.ms format.

Reason for selecting current number of threads?

I am using 16 producer threads and 16 consumer threads based on the performance observed above. The performance started deteriorating after the 16P16C configuration. The reason why I chose 16P16C is that it provided almost the same performance as 32P32C with half the number of threads.
This can be seen in the above tables.

Unexpected Results

- For testcase ‘PC_data_x1’, 3 Producers threads are more than enough to read all input and due to this I was expecting an increased execution time (due to thread creation overhead and critical section execution) as the number of threads increased. But, the execution time only started increasing (very minor degradation in performance) after 16P16C configuration.