# Build Instructions

To build the code run cmd ‘make’

**Note:**

- makefile uses the mpicc 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 ‘*lab4_hierarchical’.*

# Execution Instructions

- Reserve 5 nodes 28 ppn using qsub command on owens.
- Execute: *“mpirun -np 5 ./lab4_hierarchical <path_to_input_file>”*

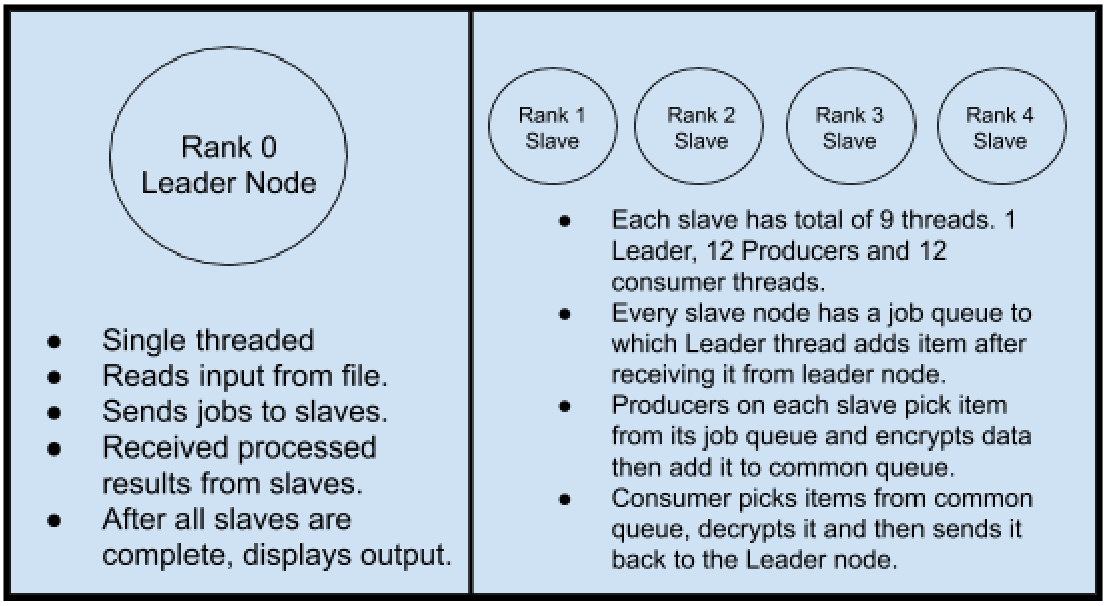
# Output

**Output format:** <position> <command> <p rank> <p tid> <encoded key> <c rank> <c tid> <decoded key>
*<position> :* the line number not counting invalid input lines

# Implementation

Program uses 5 processes to process the input file. Rank 0 process acts as the leader process and is responsible for reading the input file, sending jobs to other processes, collecting the results back and finally displaying the results on the console. Each of the slave processes is running 12 producers and 12 consumer threads that work on the job sent by the leader.

After reading an item from the input file, the leader process adds to the input queue of one of the slave processes. The producer thread on the slave process and picks an item from the input queue and calls the transform function to get the encrypted value. After this, the producer adds the item to the common queue from where it is picked by the consumer thread and decrypted. After the consumer thread has completed decryption it sends the final data back to the leader process where it gets added to the output queue. After all slave processes have completed, the leader process serially displays the output on STDOUT and then terminates execution.



- Single threaded
- Reads input from file.
- Sends jobs to slaves.
- Received processed results from slaves.
- After all slaves are complete, displays output.

- Each slave has total of 9 threads. 1 Leader, 12 Producers and 12 consumer threads.
- Every slave node has a job queue to which Leader thread adds item after receiving it from leader node.
- Producers on each slave pick item from its job queue and encrypts data then add it to common queue.
- Consumer picks items from common queue, decrypts it and then sends it back to the Leader node.

# Running Times

**NOTE: The below runtimes are without the MPI changes that were recommended by Dr. Jones. I tried with the changes and I was seeing significant performance improvement (12P12C/node took 1.54mins instead of 12mins for t1000 test case).**

| | Lab 4 (MPI) | | | | Lab 3 (OpenMP) | | | |
|---|---|---|---|---|---|---|---|---|
| | Time(2) | | Time(1) | | Time(2) | | Time(1) | |
| | P | C | Real | User | P | C | Real | User |
| **PC_data_x1** | 9 | 38 | 00: 04.891 | 00:00.011 | 2 | 16 | 00:01.713 | 00:13.766 |
| **PC_data_t00100** | 94 | 141 | 00:11.323 | 00:00.017 | 12 | 21 | 00:02.462 | 00:40.880 |
| **PC_data_t01000** | 910 | 964 | 01:03.733 | 00:00.017 | 148 | 162 | 00:10.662 | 04:24.239 |
| **PC_data_t05000** | 4699 | 4756 | 05:05.679 | 00:00.017 | 749 | 763 | 00:48.497 | 21:55.943 |
| **PC_data_t10000** | 9336 | 9393 | 09:52:996 | 00:00.012 | 1498 | 1514 | 01:35.085 | 43:32.081 |

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

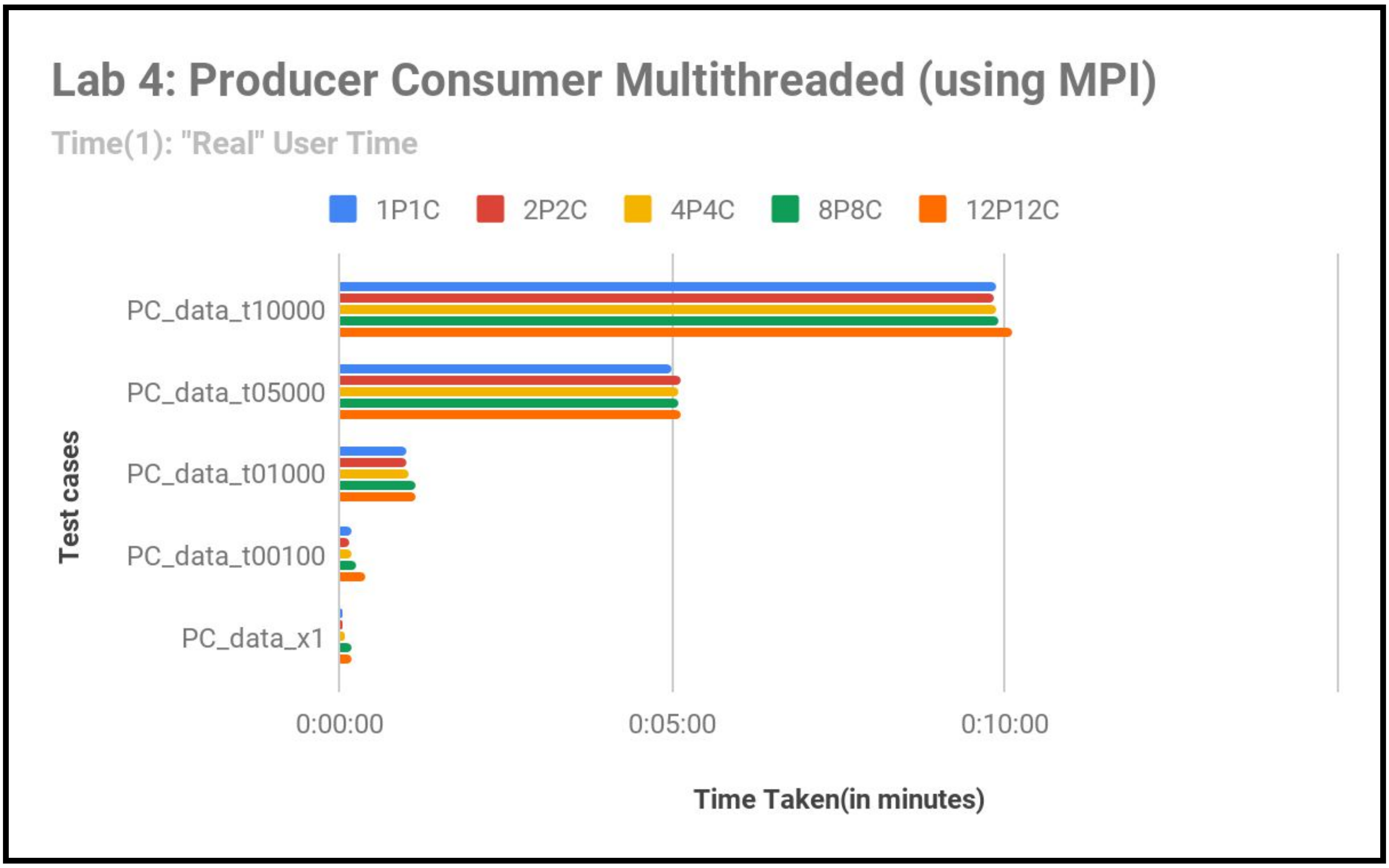
| | Lab 4 (MPI) | | | | Lab 2 (PThreads) | | | |
|---|---|---|---|---|---|---|---|---|
| | Time(2) | | Time(1) | | Time(2) | | Time(1) | |
| | P | C | Real | User | P | C | Real | User |
| **PC_data_x1** | 9 | 38 | 00: 04.891 | 00:00.011 | 3 | 32 | 00:02 | 0:4.192 |
| **PC_data_t00100** | 94 | 141 | 00:11.323 | 00:00.017 | 15 | 32 | 00:02 | 0:27.158 |
| **PC_data_t01000** | 910 | 964 | 01:03.733 | 00:00.017 | 143 | 160 | 00:11 | 4:1.991 |
| **PC_data_t05000** | 4699 | 4756 | 05:05.679 | 00:00.017 | 736 | 750 | 00:47 | 20:27.571 |
| **PC_data_t10000** | 9336 | 9393 | 09:52:996 | 00:00.012 | 1456 | 1470 | 01:32 | 40:30.495 |

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

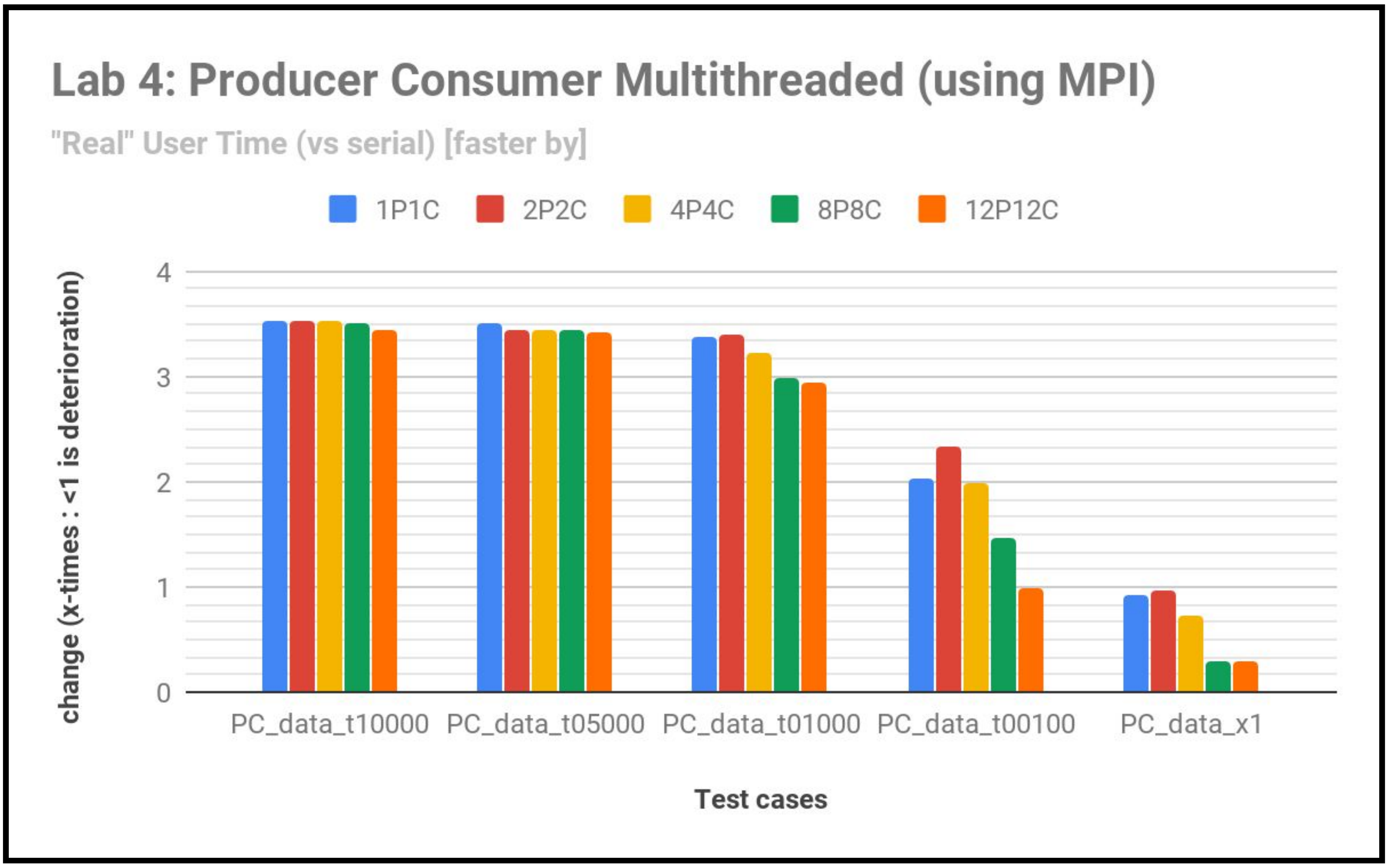
| | Lab 4 (MPI) | | | | Lab 1 (Serial) | | | |
|---|---|---|---|---|---|---|---|---|
| | Time(2) | | Time(1) | | Time(2) | | Time(1) | |
| | P | C | Real | User | P | C | Real | User |
| **PC_data_x1** | 9 | 38 | 00: 04.891 | 00:00.011 | 2 | 1 | 0:03.585 | 0:03.513 |
| **PC_data_t00100** | 94 | 141 | 00:11.323 | 00:00.017 | 10 | 12 | 0:22.411 | 0:22.341 |
| **PC_data_t01000** | 910 | 964 | 01:03.733 | 00:00.017 | 110 | 95 | 3:26.070s | 3:25.995 |
| **PC_data_t05000** | 4699 | 4756 | 05:05.679 | 00:00.017 | 518 | 535 | 17:33.385 | 17:33.147 |
| **PC_data_t10000** | 9336 | 9393 | 09:52:996 | 00:00.012 | 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.

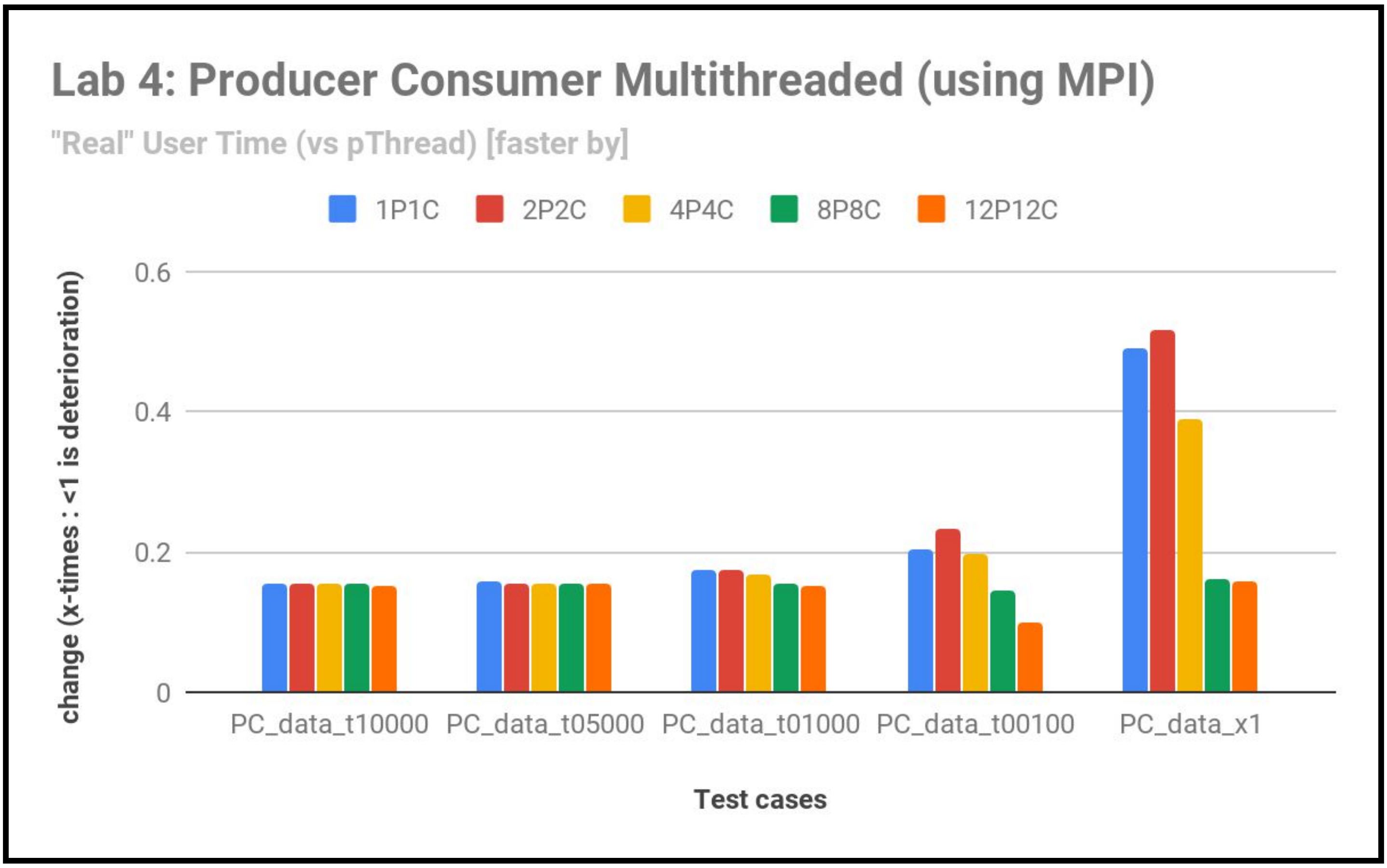
# Scalability



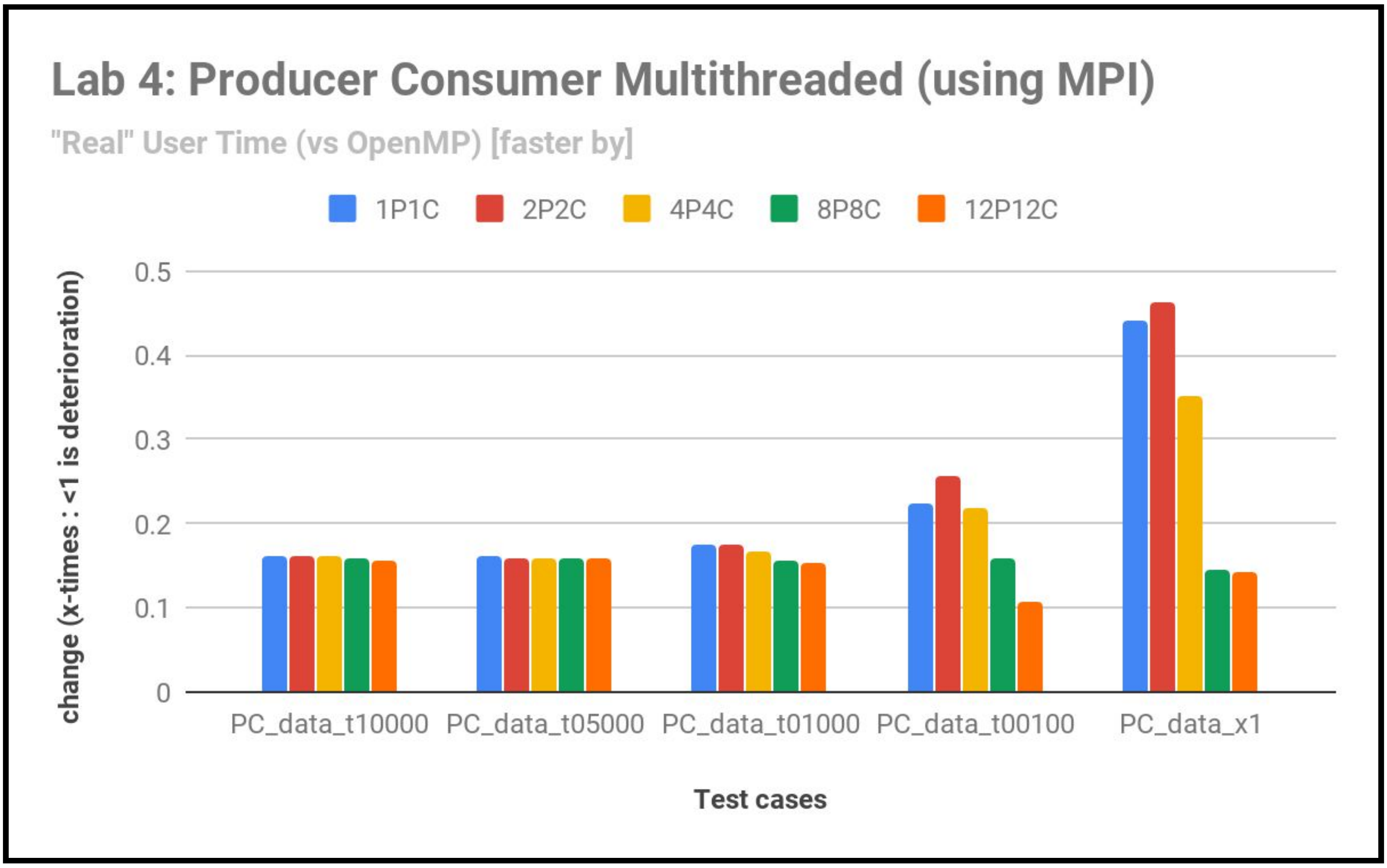
**Graph1:** Performance of MPI implementation for various thread (per slave node) counts.



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



**Graph3:** Comparison of MPI (different thread counts) with pThread(16P16C) implementation.



**Graph3:** Comparison of MPI (different thread counts) with OpenMP(16P16C) implementation.

| | **Serial** | **pThread** | **OpenMP** | **1P1C** | **2P2C** | **4P4C** | **8P8C** | **12P12C** |
|---|---|---|---|---|---|---|---|---|
| **PC_data_t10000** | 34:48.672 | 01:32.050 | 01:35.085 | 09:53.082 | 09:50.826 | 09:52.996 | 09:55.026 | 10:06.447 |
| **PC_data_t05000** | 17:33.385 | 00:47.439 | 00:48.497 | 05:00.274 | 05:06.836 | 05:05.679 | 05:05.550 | 05:07.477 |
| **PC_data_t01000** | 03:26.070 | 00:10.632 | 00:10.662 | 01:01.095 | 01:00.669 | 01:03.733 | 01:08.810 | 01:09.874 |
| **PC_data_t00100** | 00:22.411 | 00:02.237 | 00:02.462 | 00:10.990 | 00:09.623 | 00:11.323 | 00:15.406 | 00:22.803 |
| **PC_data_x1** | 00:03.585 | 00:01.910 | 00:01.713 | 00:03.885 | 00:03.691 | 00:04.891 | 00:11.903 | 00:12.078 |

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

# Unexpected Results

- For the same level of threading [(4 Producer, 4 Consumer)X4 Nodes], the time taken by the program to complete is much slower when compared to pThread(16 Producer, 16 Consumer) or OpenMP (16 Producer, 16 Consumer).