**ASSIGNMENT – 5**

**PERFORMANCE PROFILING WITH PERF**

**REPORT**

**AIM :-**

* To analyze the performance of two matrix multiplication algorithms [Naïve multiplication and Blocked multiplication] in terms of instructions , cache references , cache misses/miss-rate , L1-dcache-misses/miss-rate , L1-icache-misses/miss-rate using PERF.

**TOOL :-** PERF

**PROCEDURE :-**

* First , we design two different algorithms for matrix multiplication i) Naïve multiplication ii) Blocked multiplication.
* In Naïve method , we do the matrix multication as we do on paper.
* In Blocked method , we use the concept of divide and conquer where we divide the matrix into matrices of required block size and multiply blockwise.
* The implementation of both algorithms is done in “template.cpp” .
* Now , for testing , we generate input square matrices of any size containing random values in “gen\_input.cpp”.
* This can be done using commands :-

# $ g++ gen\_input.cpp

# $ ./a.out 128 input128

* Now , “template.cpp” can be run by :-

# $ g++ template.cpp

# $ ./a.out block\_size inputfile outputfile



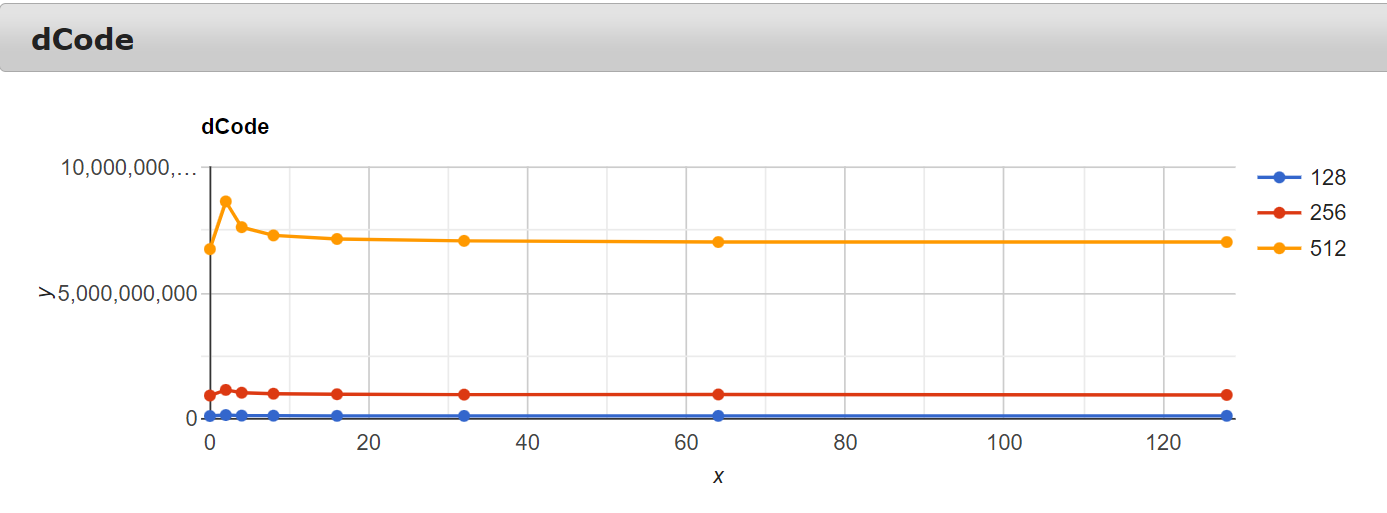
Note :- Here , blocksize is ‘0’ means we are using naïve algorithm or else the blocked algorithm of given blocksize.

* As the output file is ready , we can now use perf for analyzing the performance by using the command :-
* “sudo perf stat -e instructions,cache-references,cache-misses,L1-dcache-loads,L1-dcache-load-misses,L1-icache-loads,L1-icache-load-misses,LLC-loads,LLC-load-misses -r runs ./a.out block\_size inputfile outputfile”
* Here , this command displays the required stats only and calibrated as an average of the given number of runs.

**OBSERVATIONS :-**

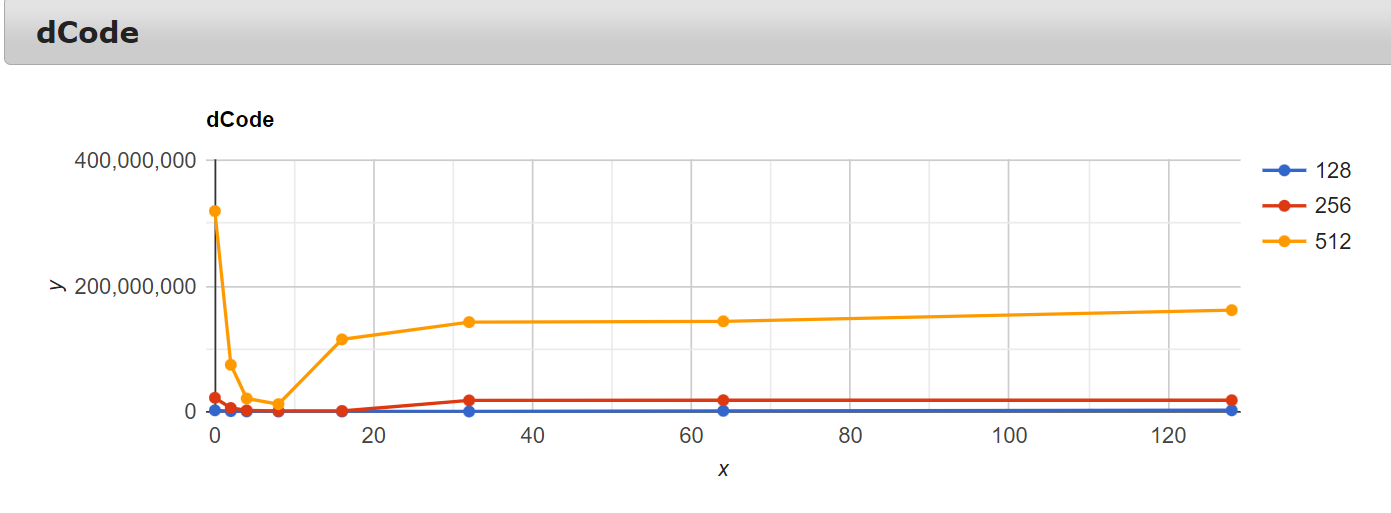
* We analyze the performance of both algorithms for blocksizes ‘128’ , ‘256’ , ‘512’.
* Note that the result data sheets are attached at the end for reference.

1. **INSTRUCTIONS :-**

****

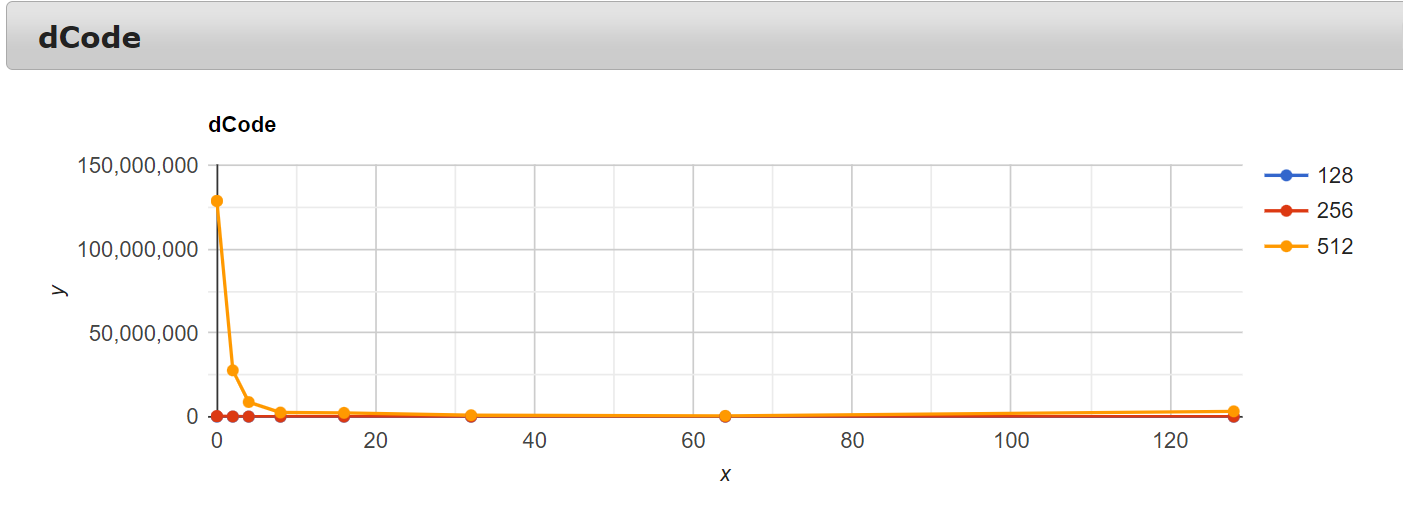
* + From the graph , it is clear that the number of instructions for blocked algorithm is always greater than or equal to the naïve algorithm.
  + Moreover , the instructions number decreases as the block size increases.
  + This observation is consistent for all three sizes.

1. **CACHE REFERENCES :-**



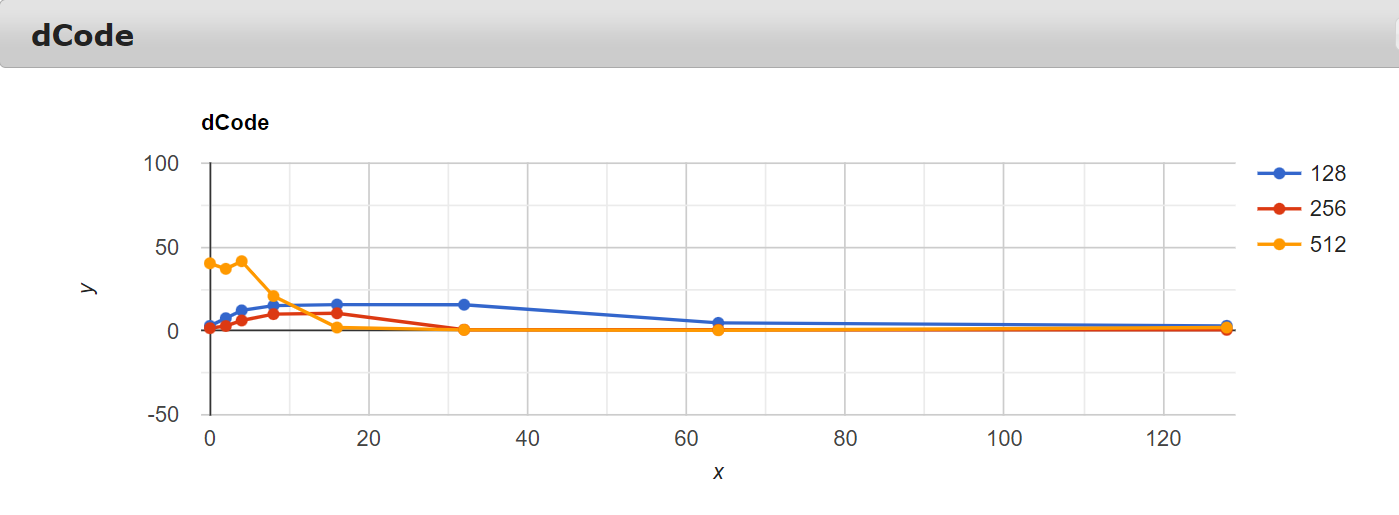
* From the graph , it is clear that the cache references for blocked algorithm are less than that of the naïve algorithm.
* Moreover , the number decreases till a certain block size and then increases thereafter.
* Here , the number decreases till blocksize – ‘16’ for matrix sizes – ‘128’ , ‘256’ but for matrix size – ‘512’ , the number decreases till blocksize – ‘8’ and then increases.

1. **CACHE MISSES :-**



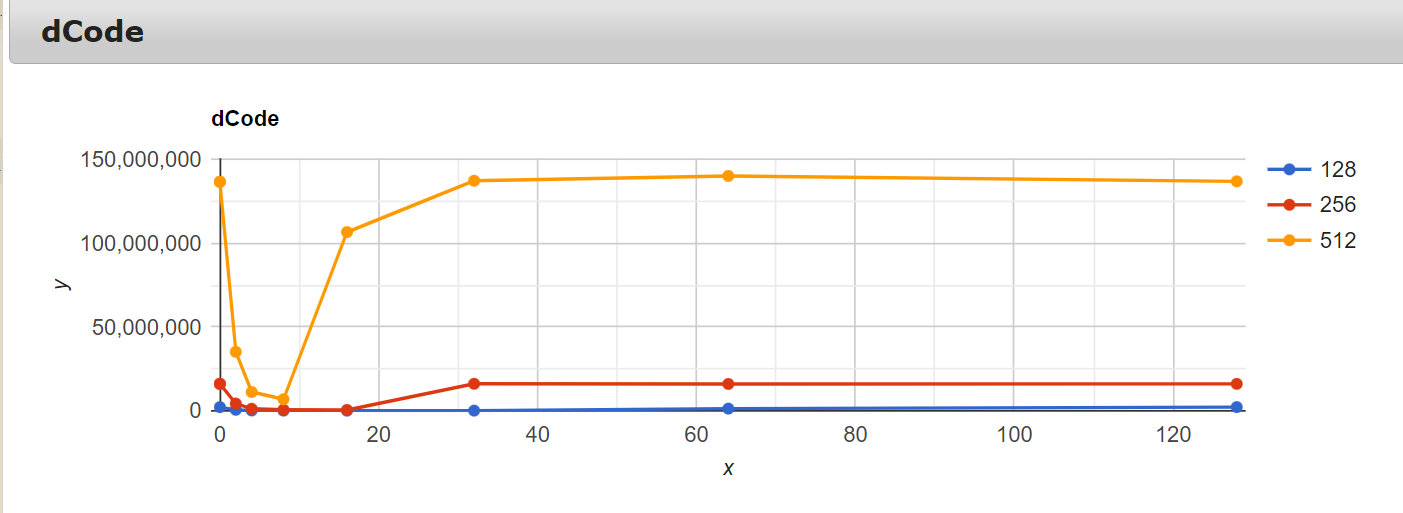
* + - From the graph , it is clear that the cache misses for blocked algorithm are less than that of the naïve algorithm.
    - Moreover , the number decreases as the block size increases (with very - very minor zizag trend).
    - Here , this observation is much clearly evident in matrix size – ‘512’ whereas some flaws are seen in matrix size – ‘128’.

1. **CACHE MISS RATE :-**



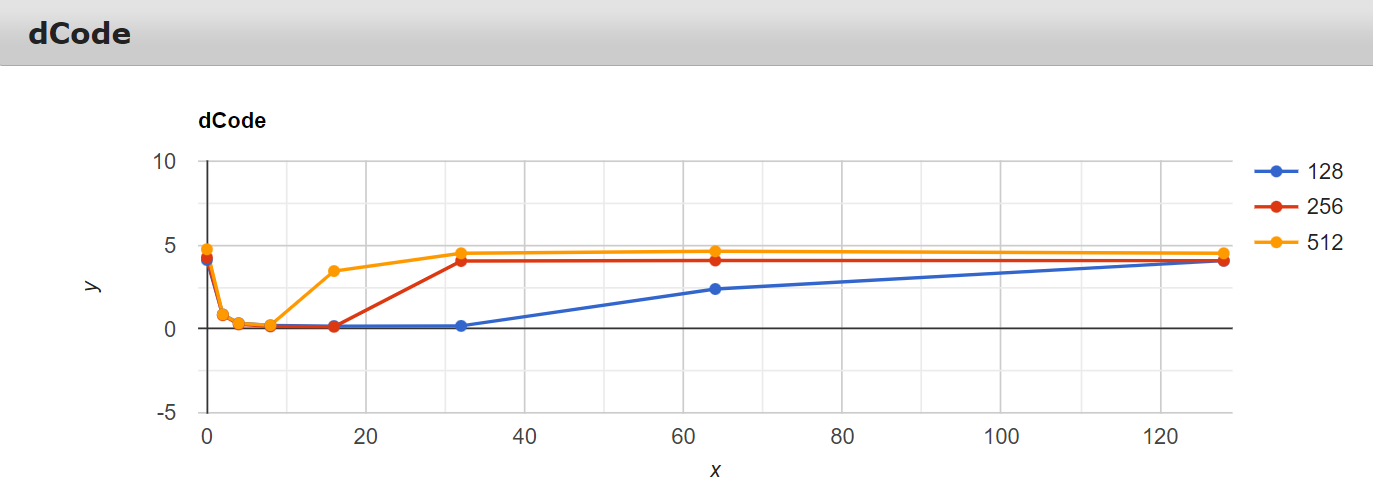
* + From the graph , for matrix sizes - ‘128’ , ‘256’ the cache miss rate for blocked algorithm is greater than that of the naïve alogorithm and the cache miss rate decreases as the block size increases.
  + For matrix size – ‘512’ the cache miss rate follows a zigzag trend which is a bit unpredictable.The values drop dramatically at block sizes – ‘16’ to ‘128’.

1. **L1 DCACHE LOAD MISSES :-**



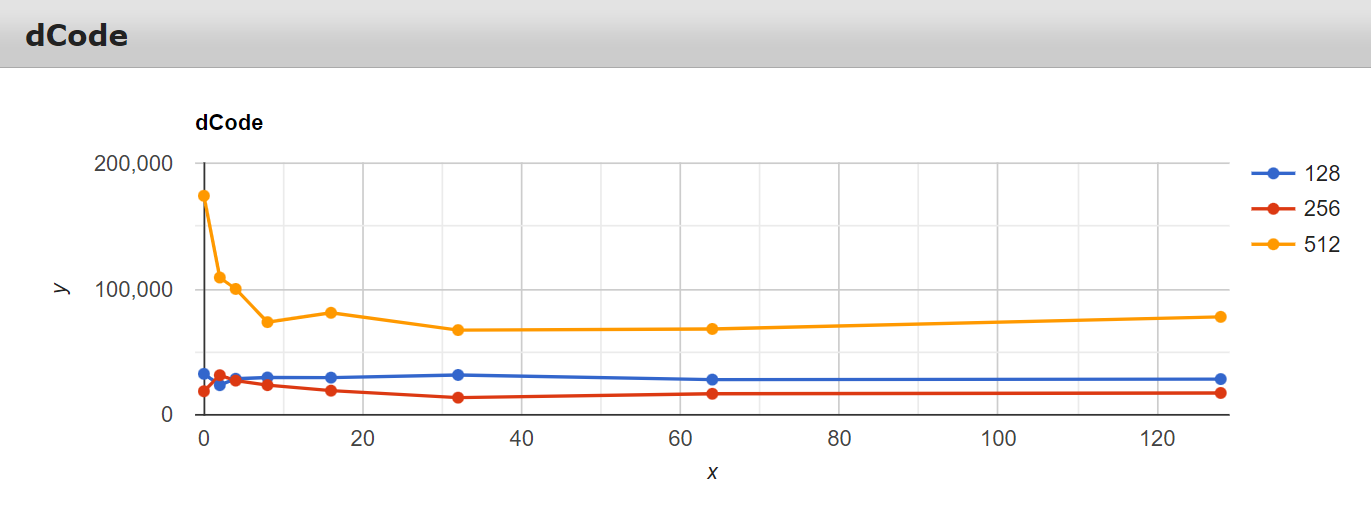
* + From the graph it is clear that the L1-dcache-load-misses for blocked algorithm are less than or equal to that of the naïve algorithm.
  + Moreover , the number decreases till a certain block size and then increases thereafter.
  + Here , the number decreases till blocksize – ‘16’ for matrix sizes – ‘128’ , ‘256’ but for matrix size – ‘512’ , the number decreases till blocksize – ‘8’ and then increases.

1. **L1 DCACHE MISS RATE :-**



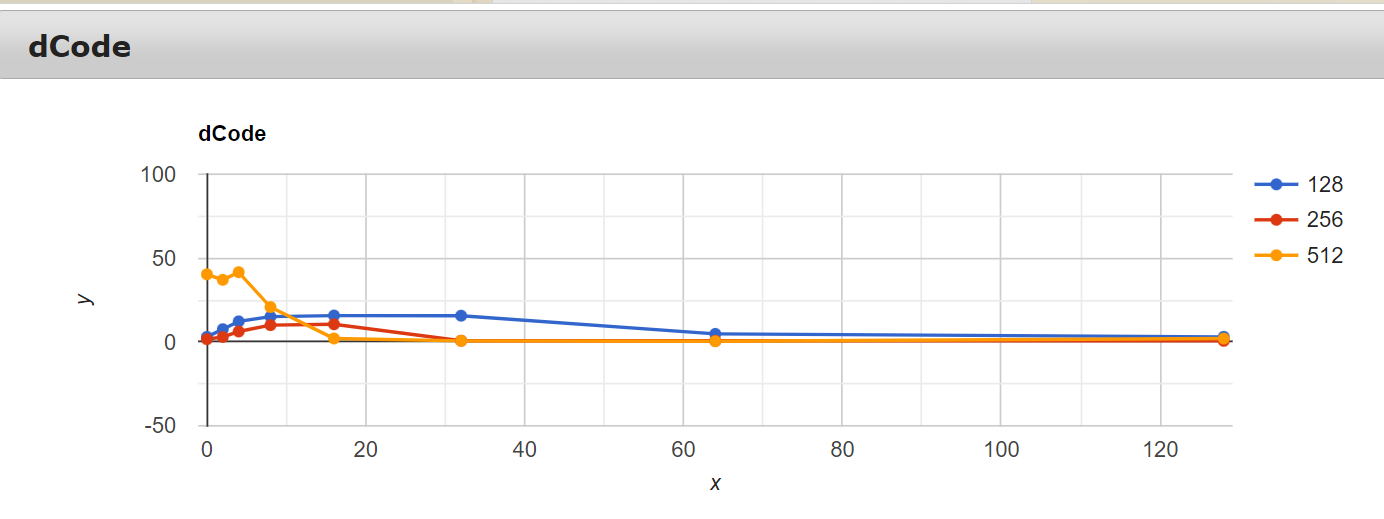
* + From the graph it is clear that the L1-dcache-load-misses for blocked algorithm are less than or equal to that of the naïve algorithm.
  + Moreover , the number decreases till a certain block size and then increases thereafter.
  + Here , the number decreases till blocksize – ‘16’ for matrix sizes – ‘128’ , ‘256’ but for matrix size – ‘512’ , the number decreases till blocksize – ‘8’ and then increases.

1. **L1 ICACHE LOAD MISSES :-**



* + From the graph L1-cache-load-misses for blocked algorithm are less than that of naïve algorithm which is clearly evident in block size – ‘128’ , ‘512’.
  + But this trend is inconsistent to block size – ‘256’ which is a bit unpredictable.

1. **L1 ICACHE MISS RATE :-**

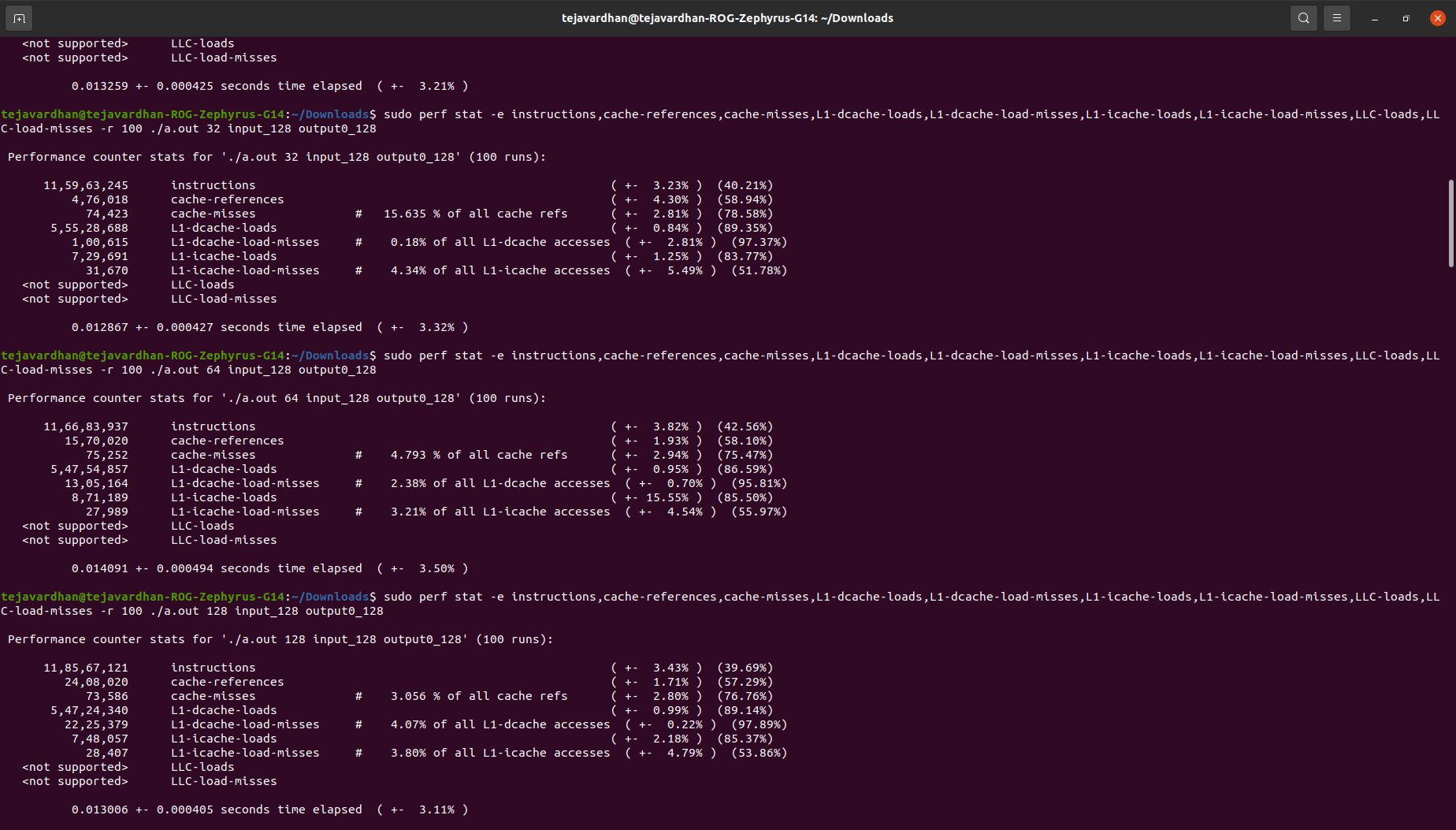
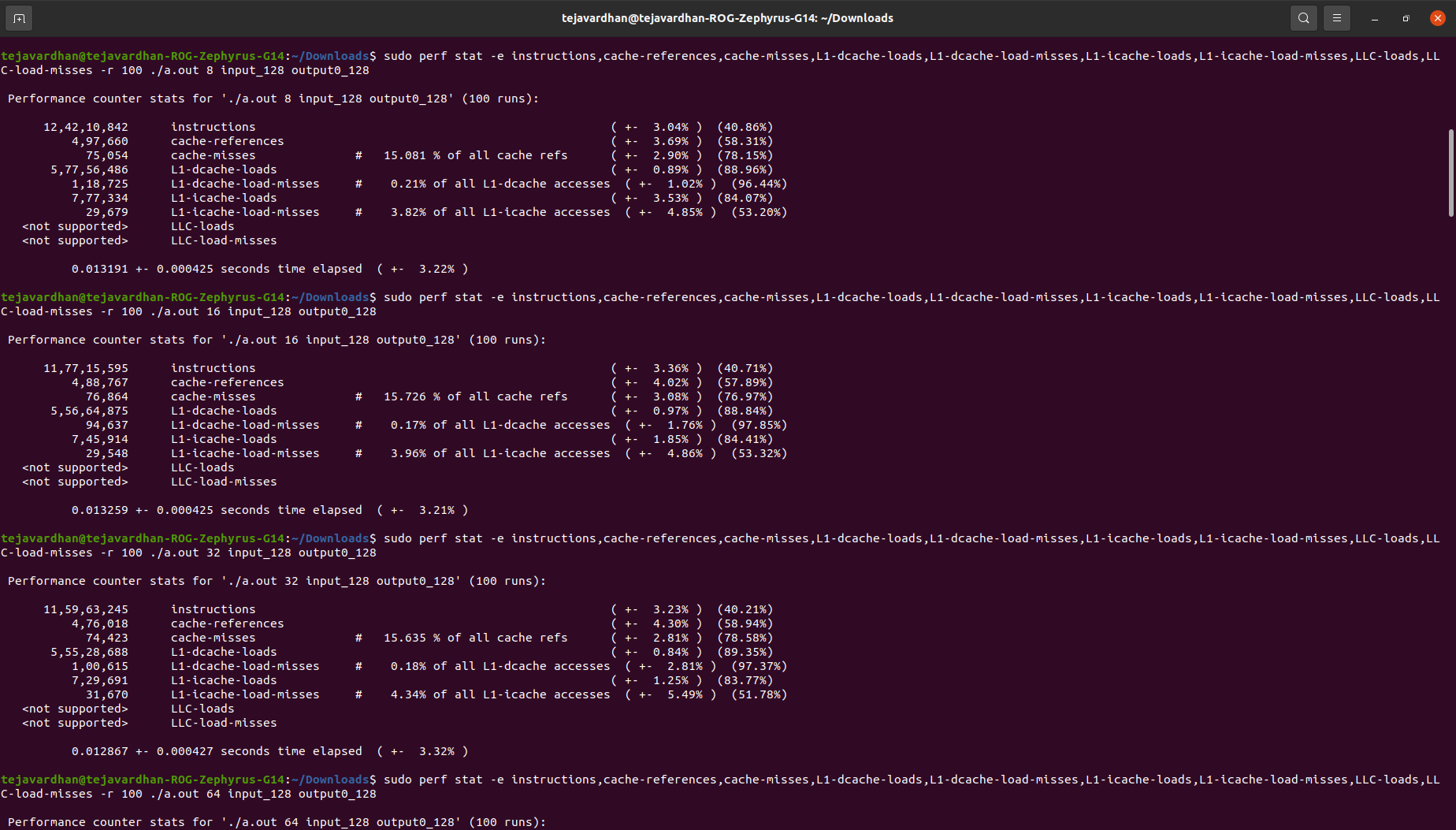
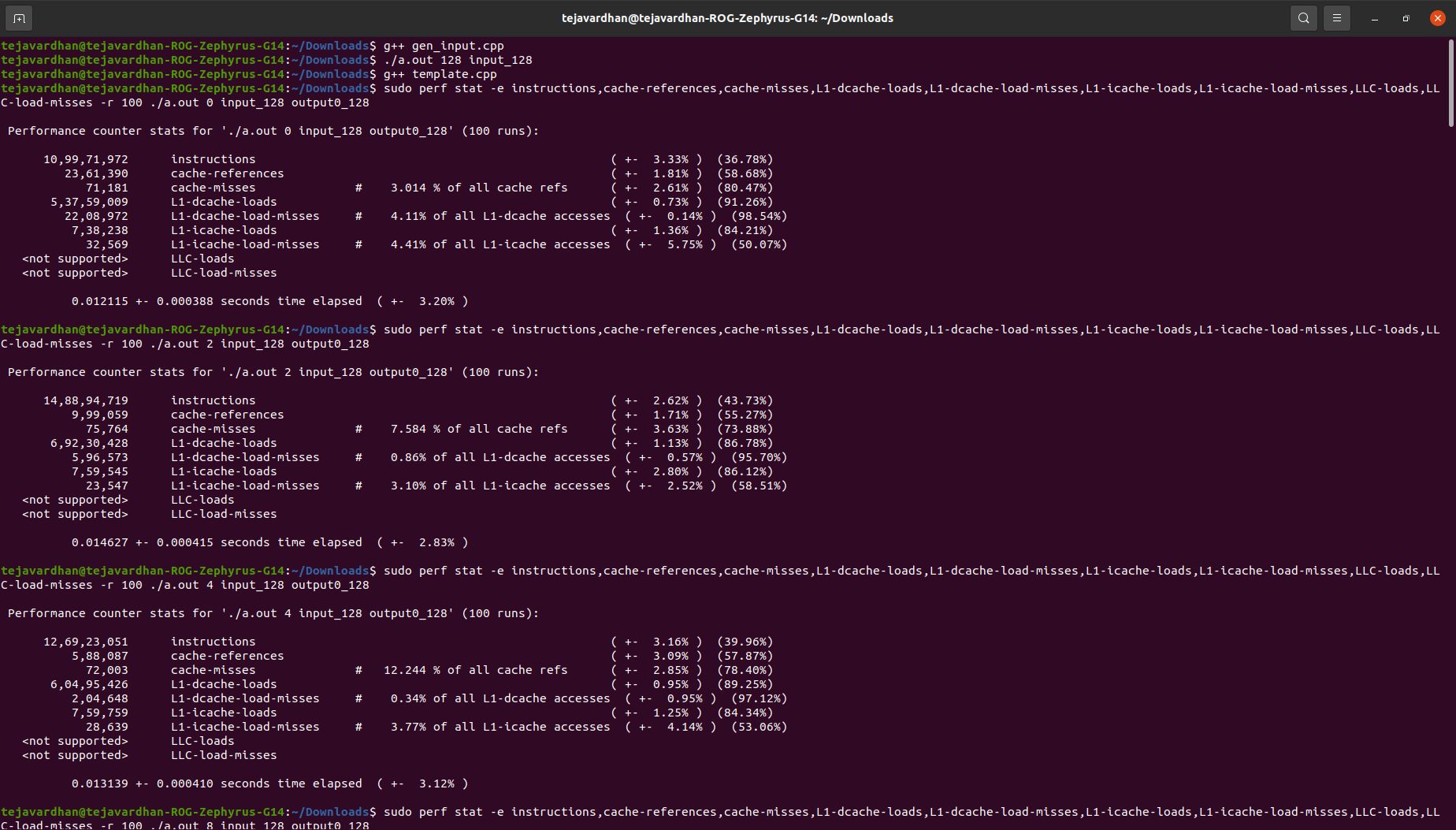


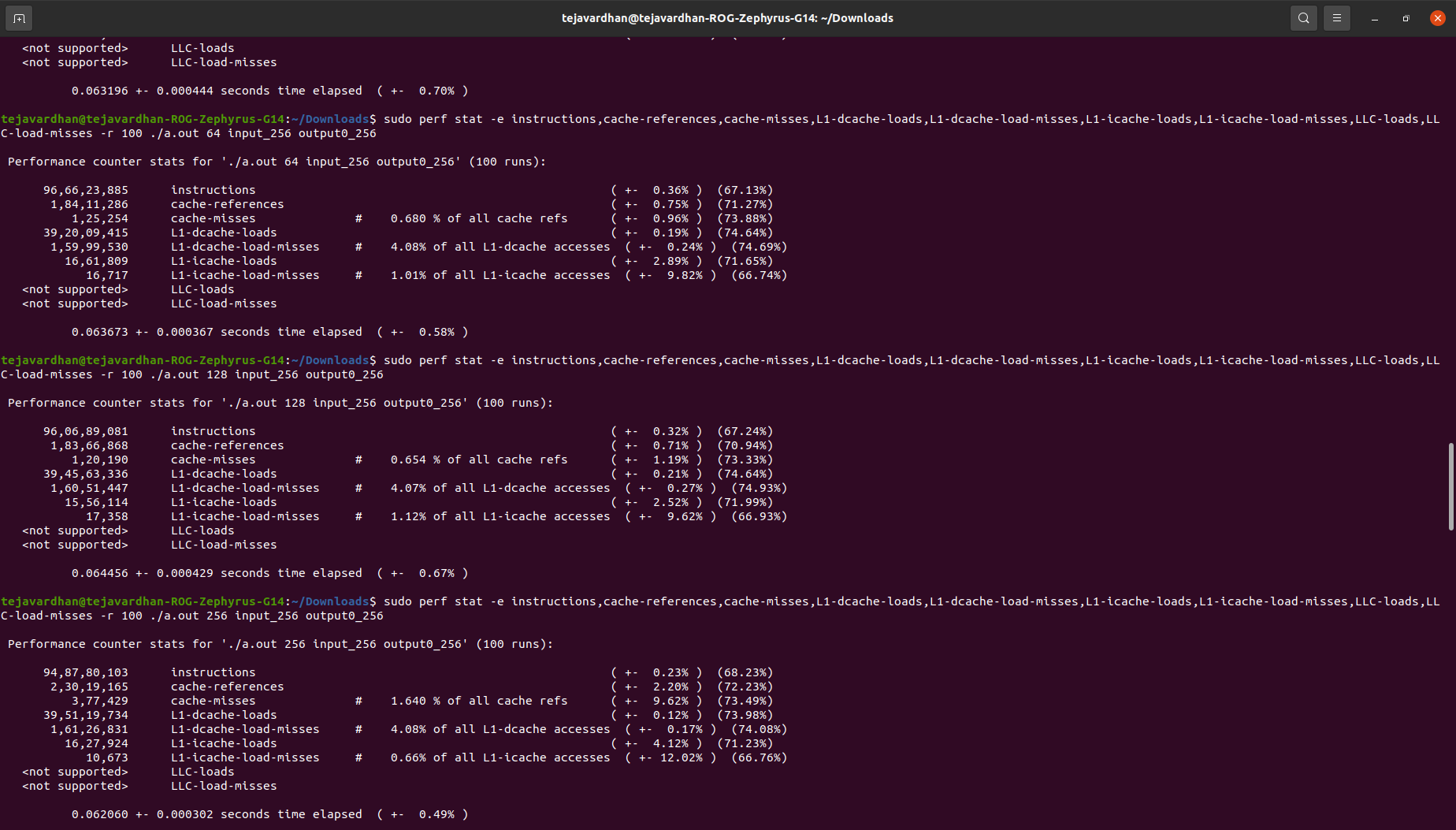
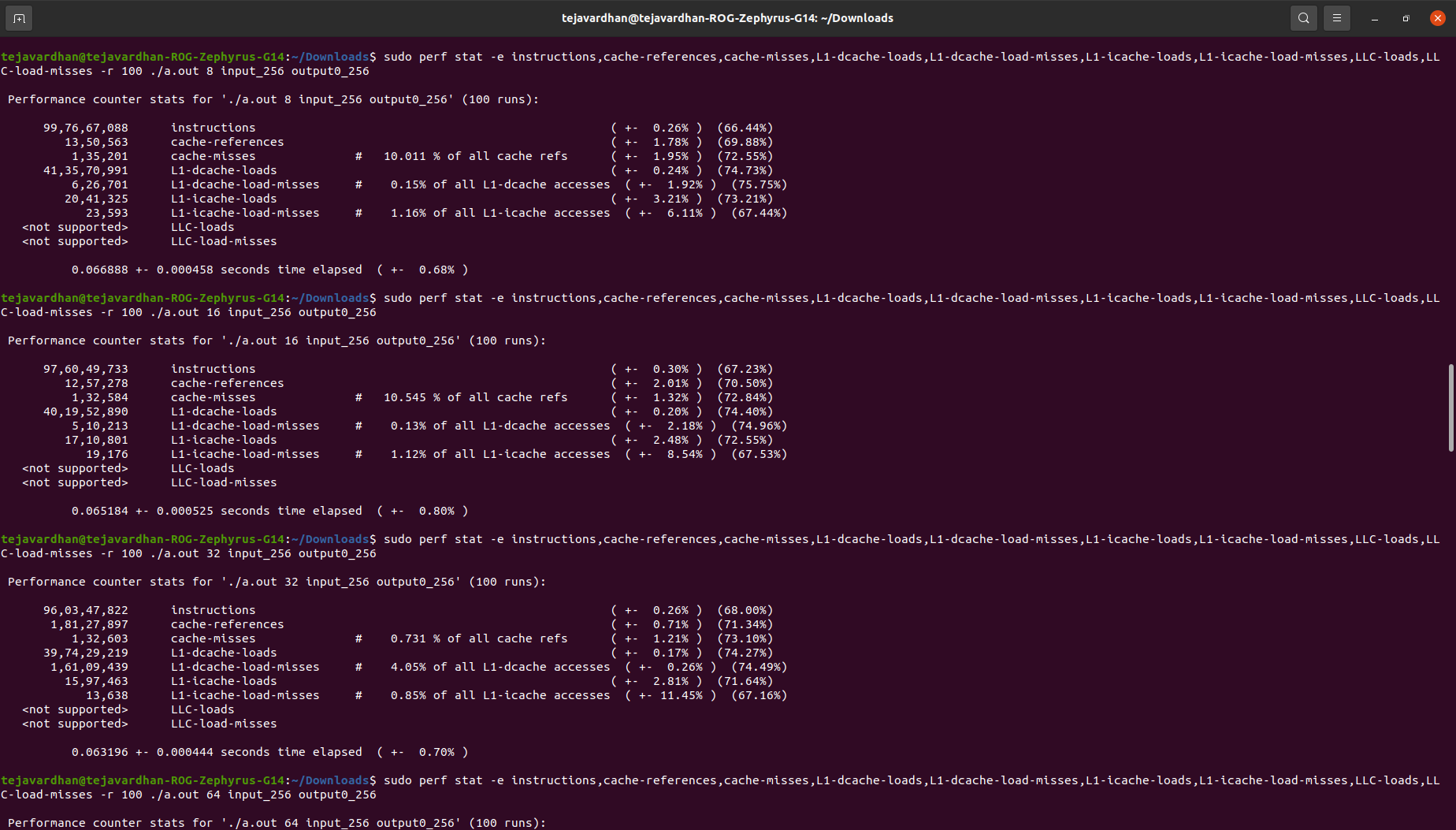
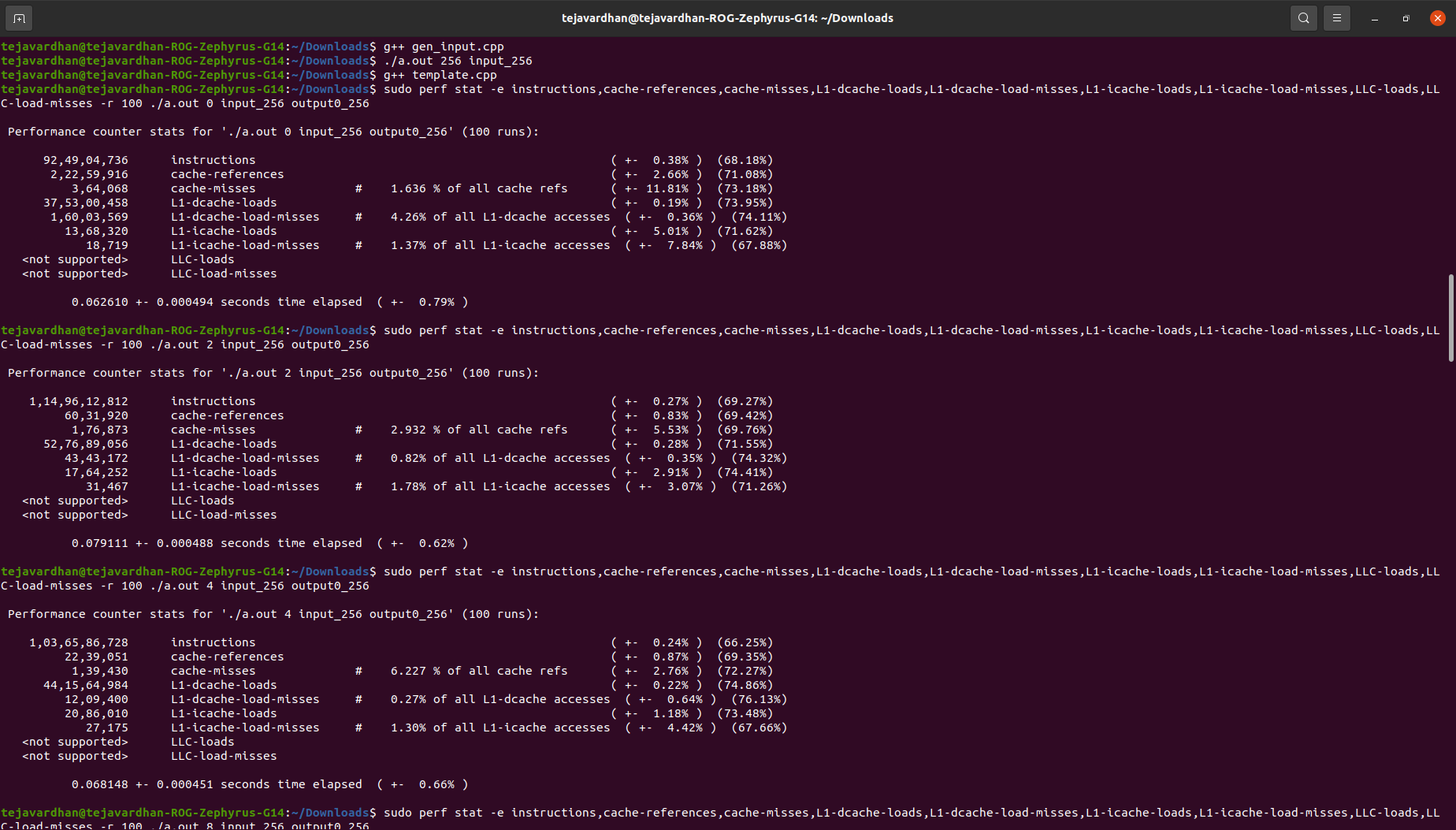
* + In this graph , there is no such perfect trend that is being followed.

**RESULT :-**

* + Blocked algorithm is more efficient than the naïve algorithm due to the added advantage of it using the ‘PRINCIPLE OF LOCALITY’.
  + Here , even though the number of instructions are more for blocked algorithm than the naïve algorithm , many other metrics are quite optimal for blocked algorithm.
  + The ideal block size for matrix sizes – ‘128’ , ‘256’ is ‘16’ and for matrix size – ‘512’ is ‘8’ especially due to lower L1-dcache-misses.
  + Overall , the ideal block size is ‘16’ considering all metrics of all three sizes.

**RESULT DATA SHEETS :-**

**MATRIX SIZE – ‘128’** 

**MATRIX SIZE – ‘256’**

**MATRIX SIZE – ‘512’**

