Name: K Srirama Srikar
Roll No.: 142301013
Lab: 7

Part 1 Spatial Locality

```
142301013 @ lab_7 $  gcc -O2 -o spatial_locality spatial_locality.c -lrt
142301013 @ lab_7 $  ./spatial_locality
Row-major time    : 0.012513 s, sum=63984000000.000000
Column-major time: 0.044799 s, sum=63984000000.000000
142301013 @ lab_7 $
```

Here we can see that the time taken for Column Major access is more than that of the Row Major access this is because the column major access jumps over every column for an element and hence the traversal takes more time.
We can see the difference in the cache misses below.

```
142301013 @ lab_7 $  valgrind --tool=cachegrind ./spatial_locality
==10022== Cachegrind, a cache and branch-prediction profiler
==10022== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10022== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==10022== Command: ./spatial_locality
==10022==
--10022-- warning: L3 cache found, using its data for the LL simulation.
Row-major time    : 0.121495 s, sum=63984000000.000000
==10022==
==10022== I   refs:      176,214,202
==10022== I1  misses:          1,465
==10022== LLi misses:          1,457
==10022== I1  miss rate:        0.00%
==10022== LLi miss rate:        0.00%
==10022==
==10022== D   refs:       32,052,523  (16,038,326 rd   + 16,014,197 wr)
==10022== D1  misses:      4,002,356  ( 2,001,694 rd   +  2,000,662 wr)
==10022== LLd misses:      4,002,070  ( 2,001,446 rd   +  2,000,624 wr)
==10022== D1  miss rate:        12.5% (      12.5%     +       12.5%  )
==10022== LLd miss rate:        12.5% (      12.5%     +       12.5%  )
==10022==
==10022== LL refs:         4,003,821  ( 2,003,159 rd   +  2,000,662 wr)
==10022== LL misses:       4,003,527  ( 2,002,903 rd   +  2,000,624 wr)
==10022== LL miss rate:          1.9% (       1.0%     +       12.5%  )
142301013 @ lab_7 $
```

This is the cachegrind output for the row traversal and as we can see the prercentage of D1 data misses is 12.5% and in the below image we have the D1 data misses for column traversal and we can see it is at 56.2%. This shows us why the use of spatial locality results in faster access time.

```
142301013 @ lab_7 $  valgrind --tool=cachegrind ./spatial_locality
==10061== Cachegrind, a cache and branch-prediction profiler
==10061== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10061== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==10061== Command: ./spatial_locality
==10061==
--10061-- warning: L3 cache found, using its data for the LL simulation.
Column-major time: 0.236256 s, sum=63984000000.000000
==10061==
==10061== I   refs:      176,214,092
==10061== I1  misses:          1,464
==10061== LLi misses:          1,456
==10061== I1  miss rate:        0.00%
==10061== LLi miss rate:        0.00%
==10061==
==10061== D   refs:       32,052,484  (16,038,304 rd   + 16,014,180 wr)
==10061== D1  misses:     18,002,356  (16,001,694 rd   +  2,000,662 wr)
==10061== LLd misses:      3,984,687  ( 1,984,063 rd   +  2,000,624 wr)
==10061== D1  miss rate:        56.2% (      99.8%     +       12.5%  )
==10061== LLd miss rate:        12.4% (      12.4%     +       12.5%  )
==10061==
==10061== LL refs:        18,003,820  (16,003,158 rd   +  2,000,662 wr)
==10061== LL misses:       3,986,143  ( 1,985,519 rd   +  2,000,624 wr)
==10061== LL miss rate:          1.9% (       1.0%     +       12.5%  )
142301013 @ lab_7 $
```

This is the cachegrind output for column traversal

Part 2 Temporal Locality

```
142301013 @ lab_7 $  gcc -o temporal_locality temporal_locality.c
142301013 @ lab_7 $  ./temporal_locality
 Single pass    :  0.003192 s , sum = 499999500000
 Repeated pass :  0.022452 s , sum = 4999995000000
142301013 @ lab_7 $
```

Here we can see that the time taken for the repeated pass is 0.022452s and for a single pass is 0.003192s. If it were the same for all the passes we would have gotten a time of 0.031920s but because of the temporal cache as we are accessing the same array multiple times, we store this in the cache and hence we the program runs faster for repeated access.

Part 3 Matrix Multiplication

```
142301013 @ lab_7 $  gcc -O2 -o matrix_multiply matrix_multiply.c -lrt
142301013 @ lab_7 $  ./matrix_multiply
Naive matrix multiplication time     : 0.429831 s
Block matrix multiplication time     : 0.001513 s
142301013 @ lab_7 $
```

As we can see the blocked matrix multiplication is considerably much faster compared to the naive matrix multiplication.

This is because in the naive approach we are accessing the elements of A in the row major order but we are accessing the elements of B in the column major order this results in a lot of cache misses and hence the time taken also increases. In the blocked approach we are accessing the a small block and the access pattern is predictable so we can make use of the temporal locality and spatial locality to access the elements faster and also we are accessing the memory as a block rather than a row or column so this also helps in the faster access.

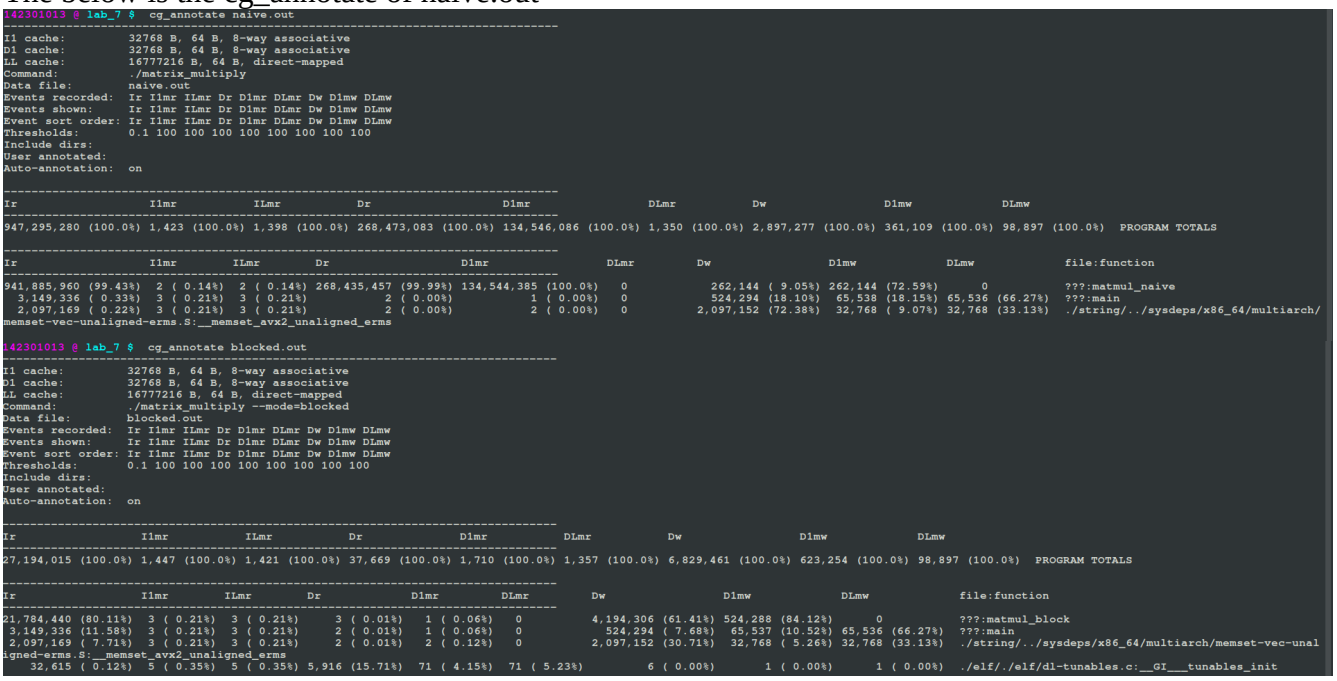
```
142301013 @ lab_7 $  gcc -O2 -o matrix_multiply matrix_multiply.c -lrt
142301013 @ lab_7 $  ./matrix_multiply
Naive matrix multiplication time     : 0.449200 s
142301013 @ lab_7 $  valgrind --tool=cachegrind --cachegrind-out-file=naive.out ./matrix_multiply
==12343== Cachegrind, a cache and branch-prediction profiler
==12343== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==12343== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==12343== Command: ./matrix_multiply
==12343==
--12343-- warning: L3 cache found, using its data for the LL simulation.
Naive matrix multiplication time     : 2.497468 s
==12343==
==12343== I   refs:      947,295,280
==12343== I1  misses:          1,423
==12343== LLi misses:          1,398
==12343== I1  miss rate:        0.00%
==12343== LLi miss rate:        0.00%
==12343==
==12343== D   refs:      271,370,360  (268,473,083 rd   + 2,897,277 wr)
==12343== D1  misses:    134,907,195  (134,546,086 rd   +   361,109 wr)
==12343== LLd misses:        100,247  (      1,350 rd   +    98,897 wr)
==12343== D1  miss rate:        49.7% (       50.1%     +      12.5%  )
==12343== LLd miss rate:         0.0% (        0.0%     +       3.4%  )
==12343==
==12343== LL refs:       134,908,618  (134,547,509 rd   +   361,109 wr)
==12343== LL misses:         101,645  (      2,748 rd   +    98,897 wr)
==12343== LL miss rate:          0.0% (        0.0%     +       3.4%  )
142301013 @ lab_7 $
```

The above image is the cachegrind output of naive matrix multiplication.

```
142301013 @ lab_7 $  gcc -O2 -o matrix_multiply matrix_multiply.c -lrt
142301013 @ lab_7 $  ./matrix_multiply
Block matrix multiplication time     : 0.001386 s
142301013 @ lab_7 $  valgrind --tool=cachegrind --cachegrind-out-file=blocked.out ./matrix_multiply --mode=blocked
==12226== Cachegrind, a cache and branch-prediction profiler
==12226== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==12226== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==12226== Command: ./matrix_multiply --mode=blocked
==12226==
--12226-- warning: L3 cache found, using its data for the LL simulation.
Block matrix multiplication time     : 0.036249 s
==12226==
==12226== I   refs:      27,194,015
==12226== I1  misses:         1,447
==12226== LLi misses:         1,421
==12226== I1  miss rate:       0.01%
==12226== LLi miss rate:       0.01%
==12226==
==12226== D   refs:       6,867,130  (37,669 rd   + 6,829,461 wr)
==12226== D1  misses:       624,964  ( 1,710 rd   +   623,254 wr)
==12226== LLd misses:       100,254  ( 1,357 rd   +    98,897 wr)
==12226== D1  miss rate:        9.1% (   4.5%     +       9.1%  )
==12226== LLd miss rate:        1.5% (   3.6%     +       1.4%  )
==12226==
==12226== LL refs:          626,411  ( 3,157 rd   +   623,254 wr)
==12226== LL misses:        101,675  ( 2,778 rd   +    98,897 wr)
==12226== LL miss rate:         0.3% (   0.0%     +       1.4%  )
142301013 @ lab_7 $
```

The above it the output of blocked matrix multiplication. As we can see there is a huge D1 misses in naive matrix multiplication but very less difference in the case of blocked matrix multiplication.

The below is the cg_annotate of naive.out



```
142301013 @ lab_7 $  cg_annotate naive.out
--------------------------------------------------------------------------------
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         32768 B, 64 B, 8-way associative
LL cache:         16777216 B, 64 B, direct-mapped
Command:          ./matrix_multiply
Data file:        naive.out
Events recorded:  Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Events shown:     Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Event sort order: Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Thresholds:       0.1 100 100 100 100 100 100 100 100
Include dirs:
User annotated:
Auto-annotation:  on

--------------------------------------------------------------------------------
Ir                  I1mr           ILmr           Dr                  D1mr                DLmr           Dw                  D1mw           DLmw
--------------------------------------------------------------------------------
947,295,280 (100.0%) 1,423 (100.0%) 1,398 (100.0%) 268,473,083 (100.0%) 134,546,086 (100.0%) 1,350 (100.0%) 2,897,277 (100.0%) 361,109 (100.0%) 98,897 (100.0%)  PROGRAM TOTALS

--------------------------------------------------------------------------------
Ir                  I1mr        ILmr        Dr                  D1mr                DLmr         Dw                  D1mw            DLmw            file:function
--------------------------------------------------------------------------------
941,885,960 (99.43%)  2 ( 0.14%)  2 ( 0.14%) 268,435,457 (99.99%) 134,544,385 (100.0%)   0            262,144 ( 9.05%) 262,144 (72.59%)      0            ???:matmul_naive
  3,149,336 ( 0.33%)  3 ( 0.21%)  3 ( 0.21%)           2 ( 0.00%)           1 ( 0.00%)   0            524,294 (18.10%)  65,538 (18.15%) 65,536 (66.27%)  ???:main
  2,097,169 ( 0.22%)  3 ( 0.21%)  3 ( 0.21%)           2 ( 0.00%)           2 ( 0.00%)   0          2,097,152 (72.38%)  32,768 ( 9.07%) 32,768 (33.13%)  ./string/../sysdeps/x86_64/multiarch/
memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms

142301013 @ lab_7 $  cg_annotate blocked.out
--------------------------------------------------------------------------------
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         32768 B, 64 B, 8-way associative
LL cache:         16777216 B, 64 B, direct-mapped
Command:          ./matrix_multiply --mode=blocked
Data file:        blocked.out
Events recorded:  Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Events shown:     Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Event sort order: Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Thresholds:       0.1 100 100 100 100 100 100 100 100
Include dirs:
User annotated:
Auto-annotation:  on

--------------------------------------------------------------------------------
Ir                  I1mr           ILmr           Dr                  D1mr                DLmr           Dw                  D1mw            DLmw
--------------------------------------------------------------------------------
27,194,015 (100.0%) 1,447 (100.0%) 1,421 (100.0%) 37,669 (100.0%) 1,710 (100.0%) 1,357 (100.0%) 6,829,461 (100.0%) 623,254 (100.0%) 98,897 (100.0%)  PROGRAM TOTALS

--------------------------------------------------------------------------------
Ir                  I1mr        ILmr        Dr                D1mr             DLmr         Dw                  D1mw            DLmw            file:function
--------------------------------------------------------------------------------
21,784,440 (80.11%)  3 ( 0.21%)  3 ( 0.21%)      3 ( 0.01%)    1 ( 0.06%)    0          4,194,306 (61.41%) 524,288 (84.12%)      0            ???:matmul_block
 3,149,336 (11.58%)  3 ( 0.21%)  3 ( 0.21%)      2 ( 0.01%)    1 ( 0.06%)    0            524,294 ( 7.68%)  65,537 (10.52%) 65,536 (66.27%)  ???:main
 2,097,169 ( 7.71%)  3 ( 0.21%)  3 ( 0.21%)      2 ( 0.01%)    2 ( 0.12%)    0          2,097,152 (30.71%)  32,768 ( 5.26%) 32,768 (33.13%)  ./string/../sysdeps/x86_64/multiarch/memset-vec-unal
igned-erms.S:__memset_avx2_unaligned_erms
    32,615 ( 0.12%)  5 ( 0.35%)  5 ( 0.35%) 5,916 (15.71%)  71 ( 4.15%)  71 ( 5.23%)          6 ( 0.00%)        1 ( 0.00%)      1 ( 0.00%)  ./elf/./elf/dl-tunables.c:__GI___tunables_init
```

And the above is the cg_annotate of blocked.out

Part 4 Matrix Transpose

```
142301013 @ lab_7 $  gcc -o transpose transpose.c
142301013 @ lab_7 $  ./transpose
 Naive transpose time                        : 0.007633 s
 Blocked transpose (blockSize=008) time : 0.007414 s
 Blocked transpose (blockSize=016) time : 0.007764 s
 Blocked transpose (blockSize=032) time : 0.006696 s
 Blocked transpose (blockSize=064) time : 0.004816 s
 Blocked transpose (blockSize=128) time : 0.003071 s
142301013 @ lab_7 $
```

This is the time taken for the matrix transpose operation for varying block sizes for an 1024x1024 matrix. And as we it is the lowest for a block size of 128 in this case.
The below is the cachegrind output for all the different cases.

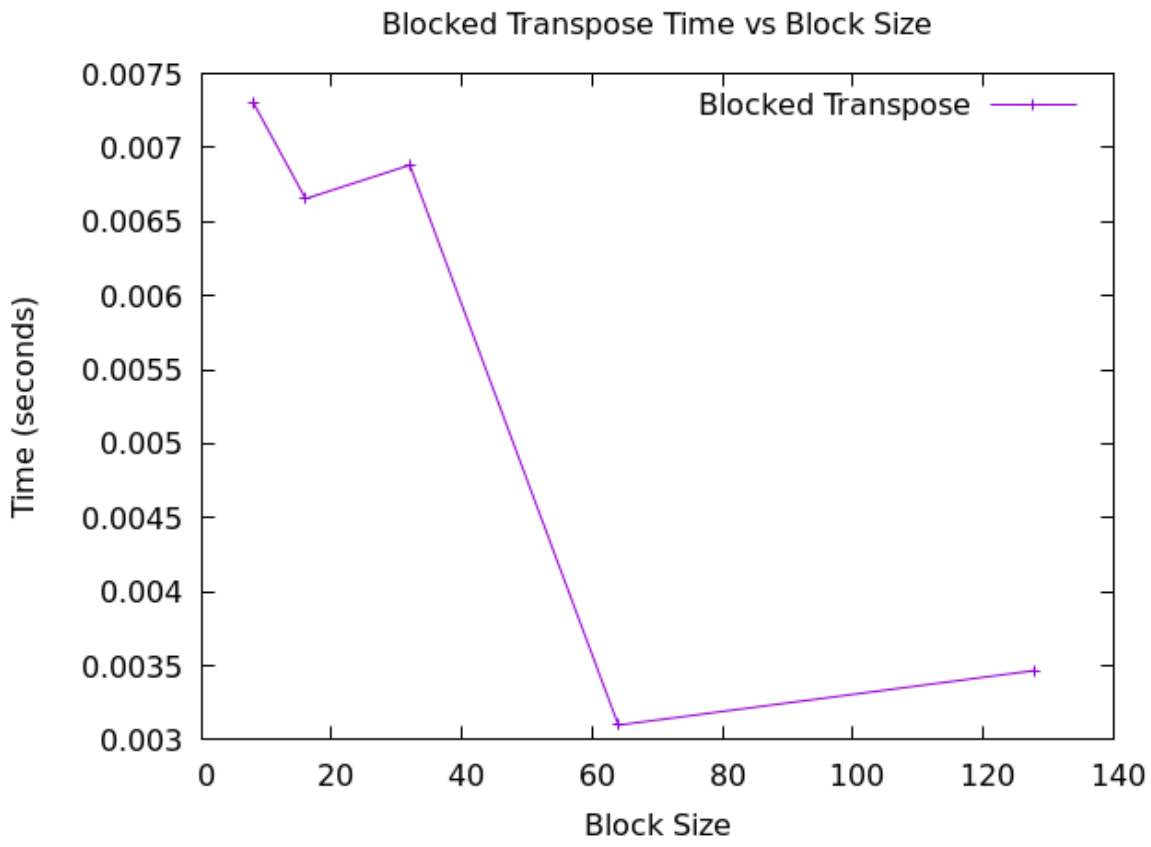
```
142301013 @ lab_7 $  gcc -o transpose transpose.c
142301013 @ lab_7 $  ./transpose
 Blocked transpose (blockSize=008) time : 0.009036 s
142301013 @ lab_7 $  valgrind --tool=cachegrind --cachegrind-out-file=a.out ./transpose --mode=blocked
==15873== Cachegrind, a cache and branch-prediction profiler
==15873== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==15873== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==15873== Command: ./transpose --mode=blocked
==15873==
--15873-- warning: L3 cache found, using its data for the LL simulation.
 Blocked transpose (blockSize=008) time : 0.064095 s
==15873==
==15873== I   refs:      55,679,459
==15873== I1  misses:         1,468
==15873== LLi misses:         1,462
==15873== I1  miss rate:       0.00%
==15873== LLi miss rate:       0.00%
==15873==
==15873== D   refs:      22,338,005  (19,030,024 rd   + 3,307,981 wr)
==15873== D1  misses:       569,204  (   132,767 rd   +   436,437 wr)
==15873== LLd misses:       264,233  (     1,452 rd   +   262,781 wr)
==15873== D1  miss rate:        2.5% (       0.7%     +      13.2%  )
==15873== LLd miss rate:        1.2% (       0.0%     +       7.9%  )
==15873==
==15873== LL refs:          570,672  (   134,235 rd   +   436,437 wr)
==15873== LL misses:        265,695  (     2,914 rd   +   262,781 wr)
==15873== LL miss rate:         0.3% (       0.0%     +       7.9%  )
142301013 @ lab_7 $
142301013 @ lab_7 $  gcc -o transpose transpose.c
142301013 @ lab_7 $  ./transpose
 Blocked transpose (blockSize=016) time : 0.008009 s
142301013 @ lab_7 $  valgrind --tool=cachegrind --cachegrind-out-file=a.out ./transpose --mode=blocked
==15921== Cachegrind, a cache and branch-prediction profiler
==15921== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==15921== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==15921== Command: ./transpose --mode=blocked
==15921==
--15921-- warning: L3 cache found, using its data for the LL simulation.
 Blocked transpose (blockSize=016) time : 0.063972 s
==15921==
==15921== I   refs:      54,614,004
==15921== I1  misses:         1,473
==15921== LLi misses:         1,467
==15921== I1  miss rate:       0.00%
==15921== LLi miss rate:       0.00%
==15921==
==15921== D   refs:      21,649,560  (18,419,465 rd   + 3,230,095 wr)
==15921== D1  misses:     1,444,148  (   132,767 rd   + 1,311,381 wr)
==15921== LLd misses:       264,233  (     1,452 rd   +   262,781 wr)
==15921== D1  miss rate:        6.7% (       0.7%     +      40.6%  )
==15921== LLd miss rate:        1.2% (       0.0%     +       8.1%  )
==15921==
==15921== LL refs:        1,445,621  (   134,240 rd   + 1,311,381 wr)
==15921== LL misses:        265,700  (     2,919 rd   +   262,781 wr)
==15921== LL miss rate:         0.3% (       0.0%     +       8.1%  )
142301013 @ lab_7 $
```

```
142301013 @ lab_7 $  gcc -o transpose transpose.c
142301013 @ lab_7 $  ./transpose
 Blocked transpose (blockSize=032) time : 0.004253 s
142301013 @ lab_7 $  valgrind --tool=cachegrind --cachegrind-out-file=a.out ./transpose --mode=blocked
==15947== Cachegrind, a cache and branch-prediction profiler
==15947== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==15947== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==15947== Command: ./transpose --mode=blocked
==15947==
--15947-- warning: L3 cache found, using its data for the LL simulation.
 Blocked transpose (blockSize=032) time : 0.061889 s
==15947==
==15947== I   refs:      54,118,114
==15947== I1  misses:         1,468
==15947== LLi misses:         1,462
==15947== I1  miss rate:       0.00%
==15947== LLi miss rate:       0.00%
==15947==
==15947== D   refs:      21,329,906  (18,135,684 rd   + 3,194,222 wr)
==15947== D1  misses:     1,444,148  (   132,767 rd   + 1,311,381 wr)
==15947== LLd misses:       264,233  (     1,452 rd   +   262,781 wr)
==15947== D1  miss rate:        6.8% (       0.7%     +      41.1%  )
==15947== LLd miss rate:        1.2% (       0.0%     +       8.2%  )
==15947==
==15947== LL refs:        1,445,616  (   134,235 rd   + 1,311,381 wr)
==15947== LL misses:        265,695  (     2,914 rd   +   262,781 wr)
==15947== LL miss rate:         0.4% (       0.0%     +       8.2%  )
142301013 @ lab_7 $
142301013 @ lab_7 $  gcc -o transpose transpose.c
142301013 @ lab_7 $  ./transpose
 Blocked transpose (blockSize=064) time : 0.004546 s
142301013 @ lab_7 $  valgrind --tool=cachegrind --cachegrind-out-file=a.out ./transpose --mode=blocked
==15975== Cachegrind, a cache and branch-prediction profiler
==15975== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==15975== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==15975== Command: ./transpose --mode=blocked
==15975==
--15975-- warning: L3 cache found, using its data for the LL simulation.
 Blocked transpose (blockSize=064) time : 0.061303 s
==15975==
==15975== I   refs:      53,879,405
==15975== I1  misses:         1,468
==15975== LLi misses:         1,462
==15975== I1  miss rate:       0.00%
==15975== LLi miss rate:       0.00%
==15975==
==15975== D   refs:      21,176,230  (17,999,176 rd   + 3,177,054 wr)
==15975== D1  misses:     1,444,148  (   132,767 rd   + 1,311,381 wr)
==15975== LLd misses:       264,233  (     1,452 rd   +   262,781 wr)
==15975== D1  miss rate:        6.8% (       0.7%     +      41.3%  )
==15975== LLd miss rate:        1.2% (       0.0%     +       8.3%  )
==15975==
==15975== LL refs:        1,445,616  (   134,235 rd   + 1,311,381 wr)
==15975== LL misses:        265,695  (     2,914 rd   +   262,781 wr)
==15975== LL miss rate:         0.4% (       0.0%     +       8.3%  )
142301013 @ lab_7 $
142301013 @ lab_7 $  gcc -o transpose transpose.c
142301013 @ lab_7 $  ./transpose
 Blocked transpose (blockSize=128) time : 0.004085 s
142301013 @ lab_7 $  valgrind --tool=cachegrind --cachegrind-out-file=a.out ./transpose --mode=blocked
==15824== Cachegrind, a cache and branch-prediction profiler
==15824== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==15824== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==15824== Command: ./transpose --mode=blocked
==15824==
--15824-- warning: L3 cache found, using its data for the LL simulation.
 Blocked transpose (blockSize=128) time : 0.060377 s
==15824==
==15824== I   refs:      53,762,233
==15824== I1  misses:         1,465
==15824== LLi misses:         1,459
==15824== I1  miss rate:       0.00%
==15824== LLi miss rate:       0.00%
==15824==
==15824== D   refs:      21,100,885  (17,932,240 rd   + 3,168,645 wr)
==15824== D1  misses:     1,444,147  (   132,766 rd   + 1,311,381 wr)
==15824== LLd misses:       264,232  (     1,451 rd   +   262,781 wr)
==15824== D1  miss rate:        6.8% (       0.7%     +      41.4%  )
==15824== LLd miss rate:        1.3% (       0.0%     +       8.3%  )
==15824==
==15824== LL refs:        1,445,612  (   134,231 rd   + 1,311,381 wr)
==15824== LL misses:        265,691  (     2,910 rd   +   262,781 wr)
==15824== LL miss rate:         0.4% (       0.0%     +       8.3%  )
142301013 @ lab_7 $
```

The above is the cachegrind output for the different block sizes and the below is the output for the naive approach

```
142301013 @ lab_7 $  gcc -o transpose transpose.c
142301013 @ lab_7 $  ./transpose
 Naive transpose time                    : 0.011051 s
142301013 @ lab_7 $  valgrind --tool=cachegrind --cachegrind-out-file=a.out ./transpose
==15584== Cachegrind, a cache and branch-prediction profiler
==15584== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==15584== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==15584== Command: ./transpose
==15584==
--15584-- warning: L3 cache found, using its data for the LL simulation.
 Naive transpose time                    : 0.050463 s
==15584==
==15584== I   refs:      50,508,283
==15584== I1  misses:         1,432
==15584== LLi misses:         1,426
==15584== I1  miss rate:       0.00%
==15584== LLi miss rate:       0.00%
==15584==
==15584== D   refs:      18,933,651  (15,772,219 rd   + 3,161,432 wr)
==15584== D1  misses:     1,444,140  (   132,764 rd   + 1,311,376 wr)
==15584== LLd misses:       264,228  (     1,449 rd   +   262,779 wr)
==15584== D1  miss rate:        7.6% (       0.8%     +      41.5%  )
==15584== LLd miss rate:        1.4% (       0.0%     +       8.3%  )
==15584==
==15584== LL refs:        1,445,572  (   134,196 rd   + 1,311,376 wr)
==15584== LL misses:        265,654  (     2,875 rd   +   262,779 wr)
==15584== LL miss rate:         0.4% (       0.0%     +       8.3%  )
142301013 @ lab_7 $
```

The below are the graphs for different values plotted in gnuplot



As we can see here we are getting the optimal block size at 64 for this particular run.
This can be because of the system specifications, size of the cache as it impacts the amount that can be stored in a block and hence we get the difference.
Now below is the image of the D1 cache misses of different block sizes and as we can see it almost same from block size 16 onwards this might be because the data might be too big to be fit in L1 cache so we are going for L2 cache. This is just an assumption form my observation.