Document COL-380

Sourav (CS1190404)

January 2022

1 Debugging :

Before executing the code some important thing that I do for running the code and analysis the code using the tools gprof and valgrind.
1.) In classify.h in function operator+= function add the return *this; at the end of function(at line number 95) and the code working fine.
2.) For running the gprof analysis tool firstly I make some command in makefile to run the gprof that make a gprof function call n makefile that gives the data of analysis_gprof.out that contains the analysis of gprof and also for running the gprof I add -pg in every executable file's running command so that for running the gprof file I wanted to make a gmon.out file such that I analyze the data using the gprof.
3.) For running the valgrind analysis tool I just add some command for analysis of simple valgrind and analysis of leaked valgrind and analysis of cache valgrind tool.

2 Makefile :

In this I just want to tell how to execute commands of Makefile.
1.) For Compile Run the code : **make run**
2.) For Gprof analysis file : **make gprof**
3.) For Valgrind analysis: **make valgrind**
4.) For Valgrind Leak Memory analysis : **make valgrind_leak_check**
5.) For Valgrind Cache analysis : **make valgrind_cache**

3 Original Code Analysis:

In this section we discuss about the analysis report of Gprof and Valgrind and it's total cache behaviour of the original code.

3.1 Compile & Run Analysis:

```
radhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$ make run
g++ -std=c++11 -O2 -c -pg main.cpp
g++ -std=c++11 -O2 -c -pg classify.cpp
g++ -std=c++11 -O2 -fopenmp -pg main.o classify.o -o classify
./classify rfile dfile 1009072 4 3
590.73 ms
602.16 ms
592.763 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 590.73 ms, Average was 595.218 ms
```

In the above picture shown that the command used make run for compile and running the code and the instructions shown in the image first compile the files and make a output file classify and then run the file on 4 threads and for 3 iterations and time taken by each iteration and the fastest time taken in these 3 iterations and the average time taken by all of them.

3.2 Gprof Analysis:

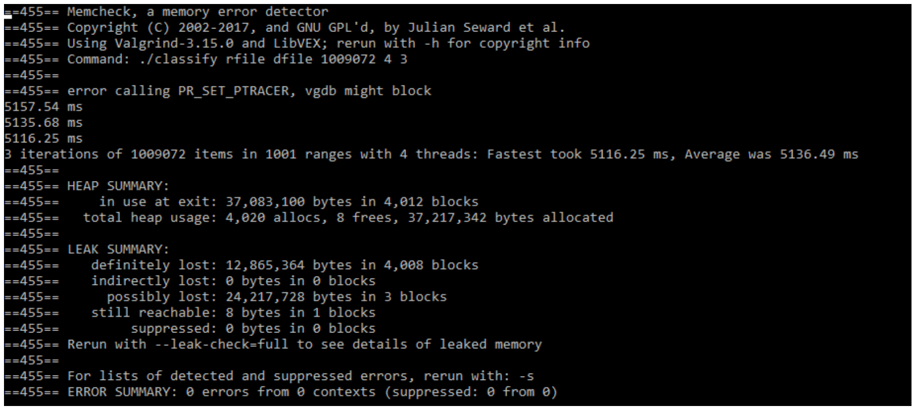
```
radhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$ make gprof
gprof classify gmon.out > analysis_gprof.out
```

In the above image shown that the command use make gprof for analysis give us the output file analysis_gprof.out that contains the analysis of the code that contains number of function calls happened on a function and their time taken by them.

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        3     0.00     0.00  classify(Data&, Ranges const&, unsigned int)
  0.00      0.00     0.00        3     0.00     0.00  timedwork(Data&, Ranges const&, unsigned int)
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj
^L
                        Call graph


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       3/3           timedwork(Data&, Ranges const&, unsigned int) [9]
[8]      0.0    0.00    0.00       3         classify(Data&, Ranges const&, unsigned int) [8]
-----------------------------------------------
                0.00    0.00       3/3           repeatrun(unsigned int, Data&, Ranges const&, unsigned int) [14]
[9]      0.0    0.00    0.00       3         timedwork(Data&, Ranges const&, unsigned int) [9]
                0.00    0.00       3/3           classify(Data&, Ranges const&, unsigned int) [8]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [18]
[10]     0.0    0.00    0.00       1         _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj [10]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [18]
[11]     0.0    0.00    0.00       1         _GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj [11]
-----------------------------------------------
^L
Index by function name

  [10] _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj [8] classify(Data&, Ranges const&, unsigned int)
  [11] _GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj [9] timedwork(Data&, Ranges const&, unsigned int)
```

In the above shown image this is the analysis report when we call the gprof function in this file we can easily able to see that the data analysis shown in different profile i.e call tree and flat we are given the time in seconds of the various functions and the number of calls made to the function.In this we can able to see the child parent relations of the functions and how many times the parent and the child called in the program.

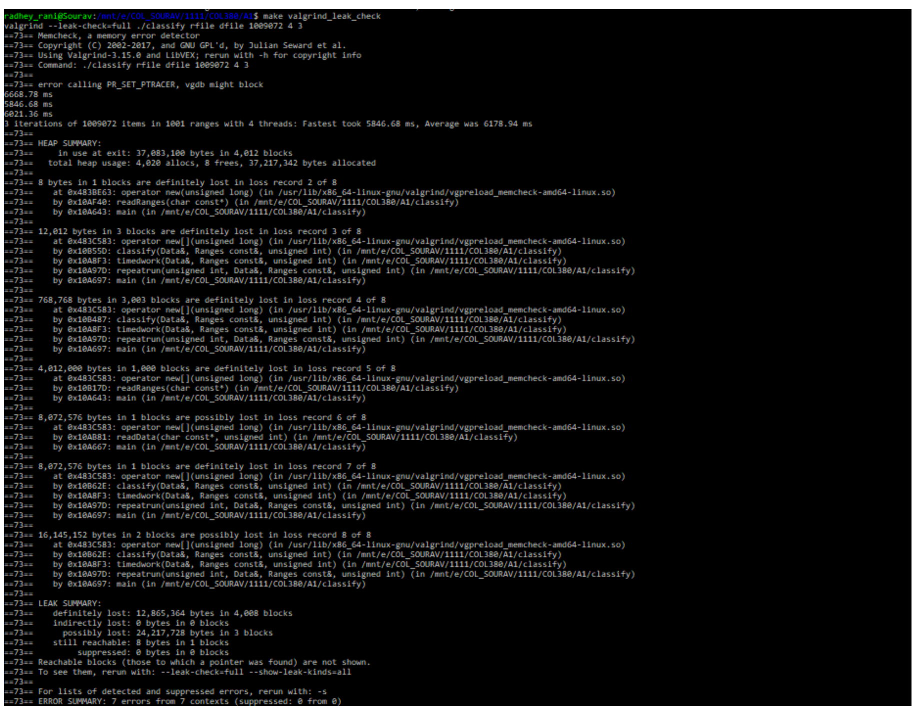
3.3 Valgrind Analysis:



```
==455== Memcheck, a memory error detector
==455== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==455== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==455== Command: ./classify rfile dfile 1009072 4 3
==455==
==455== error calling PR_SET_PTRACER, vgdb might block
5157.54 ms
5135.68 ms
5116.25 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 5116.25 ms, Average was 5136.49 ms
==455==
==455== HEAP SUMMARY:
==455==     in use at exit: 37,083,100 bytes in 4,012 blocks
==455==   total heap usage: 4,020 allocs, 8 frees, 37,217,342 bytes allocated
==455==
==455== LEAK SUMMARY:
==455==    definitely lost: 12,865,364 bytes in 4,008 blocks
==455==    indirectly lost: 0 bytes in 0 blocks
==455==      possibly lost: 24,217,728 bytes in 3 blocks
==455==    still reachable: 8 bytes in 1 blocks
==455==         suppressed: 0 bytes in 0 blocks
==455== Rerun with --leak-check=full to see details of leaked memory
==455==
==455== For lists of detected and suppressed errors, rerun with: -s
==455== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

In the above image we can see hat we execute the command for just valgrind report then we can get the data for heap summary used and the leak summary and also the execution time taken by the 4 threads for 3 iterations.

3.4 Valgrind_leak Analysis:



```
radhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$ make valgrind_leak_check
valgrind --leak-check=full ./classify rfile dfile 1009072 4 3
==73== Memcheck, a memory error detector
==73== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==73== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==73== Command: ./classify rfile dfile 1009072 4 3
==73==
==73== error calling PR_SET_PTRACER, vgdb might block
6668.78 ms
5846.68 ms
6021.36 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 5846.68 ms, Average was 6178.94 ms
==73==
==73== HEAP SUMMARY:
==73==     in use at exit: 37,083,100 bytes in 4,012 blocks
==73==   total heap usage: 4,020 allocs, 8 frees, 37,217,342 bytes allocated
==73==
==73== 8 bytes in 1 blocks are definitely lost in loss record 2 of 8
==73==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==73==    by 0x10AF40: readRanges(char const*) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A643: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==
==73== 12,012 bytes in 3 blocks are definitely lost in loss record 3 of 8
==73==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==73==    by 0x10B55D: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==
==73== 768,768 bytes in 3,003 blocks are definitely lost in loss record 4 of 8
==73==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==73==    by 0x10B487: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==
==73== 4,012,000 bytes in 1,000 blocks are definitely lost in loss record 5 of 8
==73==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==73==    by 0x10B17D: readRanges(char const*) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A643: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==
==73== 8,072,576 bytes in 1 blocks are possibly lost in loss record 6 of 8
==73==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==73==    by 0x10AB81: readData(char const*, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A667: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==
==73== 8,072,576 bytes in 1 blocks are definitely lost in loss record 7 of 8
==73==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==73==    by 0x10B62E: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==
==73== 16,145,152 bytes in 2 blocks are possibly lost in loss record 8 of 8
==73==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==73==    by 0x10B62E: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==73==
==73== LEAK SUMMARY:
==73==    definitely lost: 12,865,364 bytes in 4,008 blocks
==73==    indirectly lost: 0 bytes in 0 blocks
==73==      possibly lost: 24,217,728 bytes in 3 blocks
==73==    still reachable: 8 bytes in 1 blocks
==73==         suppressed: 0 bytes in 0 blocks
==73== Reachable blocks (those to which a pointer was found) are not shown.
==73== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==73==
==73== For lists of detected and suppressed errors, rerun with: -s
==73== ERROR SUMMARY: 7 errors from 7 contexts (suppressed: 0 from 0)
```

In the above image we can see hat we execute the command for valgrind full memory leak report then we can get the data for leak memory details and also the execution time taken by the 4 threads for 3 iterations.

3.5 Valgrind_cache Analysis:

```
==80== Cachegrind, a cache and branch-prediction profiler
==80== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==80== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==80== Command: ./classify rfile dfile 1009072 4 3
==80==
--80-- warning: L3 cache found, using its data for the LL simulation.
==80== error calling PR_SET_PTRACER, vgdb might block
13076.7 ms
12673.1 ms
12469.6 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 12469.6 ms, Average was 12739.8 ms
==80==
==80== I   refs:      9,130,338,799
==80== I1  misses:            3,177
==80== LLi misses:            3,057
==80== I1  miss rate:          0.00%
==80== LLi miss rate:          0.00%
==80==
==80== D   refs:      1,867,189,313  (1,774,636,071 rd   + 92,553,242 wr)
==80== D1  misses:       96,505,332  (   95,390,132 rd   +  1,115,200 wr)
==80== LLd misses:       96,492,524  (   95,378,381 rd   +  1,114,143 wr)
==80== D1  miss rate:           5.2% (          5.4%     +        1.2%  )
==80== LLd miss rate:           5.2% (          5.4%     +        1.2%  )
==80==
==80== LL refs:          96,508,509  (   95,393,309 rd   +  1,115,200 wr)
==80== LL misses:        96,495,581  (   95,381,438 rd   +  1,114,143 wr)
==80== LL miss rate:            0.9% (          0.9%     +        1.2%  )
```

In the above image we can see hat we execute the command for valgrind cache report then we can get the data for cache details and also the execution time taken by the 4 threads for 3 iterations and also in this we are able to see that I refs which will tell about the number of instruction that are executed during the running of the code and I1 misses tells about the cache read misses during the running of the code and also LLi misses tells us about the instruction read misses during the running of the code. and after that gives the data in percentage(for both LLi and I1).
In 2nd part the D refs tells us about the number of memory read and write misses during the running of the code and D1 misses tells us about the memory read and write misses and also in their bracket tells us the how many read and write calls are happend in the memory. LLd misses tells us about the how many memory read and write calls are missed after that the data is given in the percentage for both read and write command.
In 3rd part tells us about the total read and write data is misses during the execution of the read and write operations are happens and the data given in the percent.

4 Optimization Idea

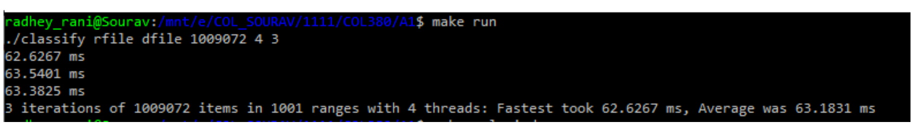
By analysing the behaviour of the cache in the original code I think about it that I can improve my cache behaviour in such a way that the number of instruction is executed is less that the original code and also focus on this that if instructions are less then the percentage of miss the instruction is very low such that my code is improved and my cache use less memory for read and write operations also.
And also look toward the time taken by the code to execute the iteration is quite high so I also think about that side that it is better if I am able to improve my time complexity then this given me benefit that if time complexity is improve

then the number of instruction calls is also decreased and that lead to a good optimization so by analysing the code.
I am find a function where I can use my algorithm improvement approach in file classify.cpp the last omp parallel function their in place of linear search we can easily use the binary search and as all know that the time complexity is improved much by using binary search that in place of O(n) time complexity it is now O(log n) for searching that implies that if their are approximately n instruction is used for searching while their is only log(n) i.e if n is large enough let's say 10^9 than firstly it execute 10^9 instructions now only log(10^9) approximately 18 instructions are executed and also number instruction are less and time is also taken by him is less so less instruction executed means less read and write operations happened means less memory used so this is quite big improvement in both time complexity as well as cache memory used and also cache instruction executed.
2nd Improvement that I do is in the 1st function which is also named as omp parallel by analyzing this function I get that we can improve it in a way such that the read and write operations i.e Drefs and the missing data we can reduced by applying this optimization so for optimizing this in a way such that the classify function calculate the perc time taken by the classify function and then change this so that it will reduced and also drefs and the missing rate is reduced and the results for optimized code shown below and I think there is lot of effect of these optimization and our code is improved so much and it is good enough.

5 Optimized Code Analysis:

In this section we discuss about the analysis report of Gprof and Valgrind and it's total cache behaviour of the optimized code.

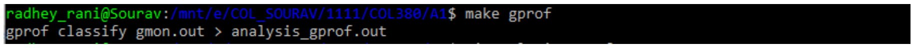
5.1 Compile & Run Analysis:



```
radhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$ make run
./classify rfile dfile 1009072 4 3
62.6267 ms
63.5401 ms
63.3825 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 62.6267 ms, Average was 63.1831 ms
```

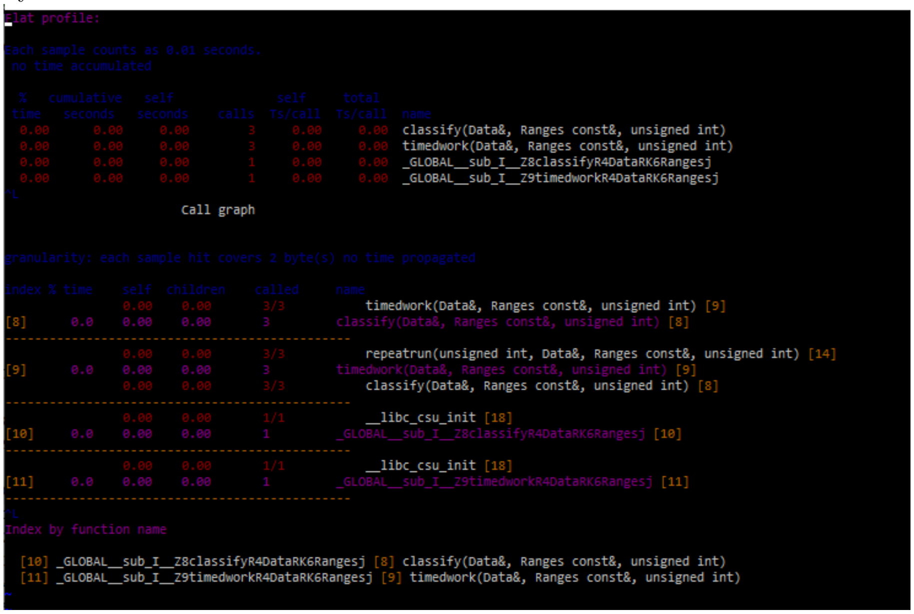
In the above picture shown that the command used make run for compile and running the code and the instructions shown in the image first compile the files and make a output file classify and then run the file on 4 threads and for 3 iterations and time taken by each iteration and the fastest time taken in these 3 iterations and the average time taken by all of them and clearly in this we will able to see that the improvement from our 2 optimized approach is approximately 9 times better in case of time so yes this is a nice improvement all over.

5.2 Gprof Analysis:



```
radhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$ make gprof
gprof classify gmon.out > analysis_gprof.out
```

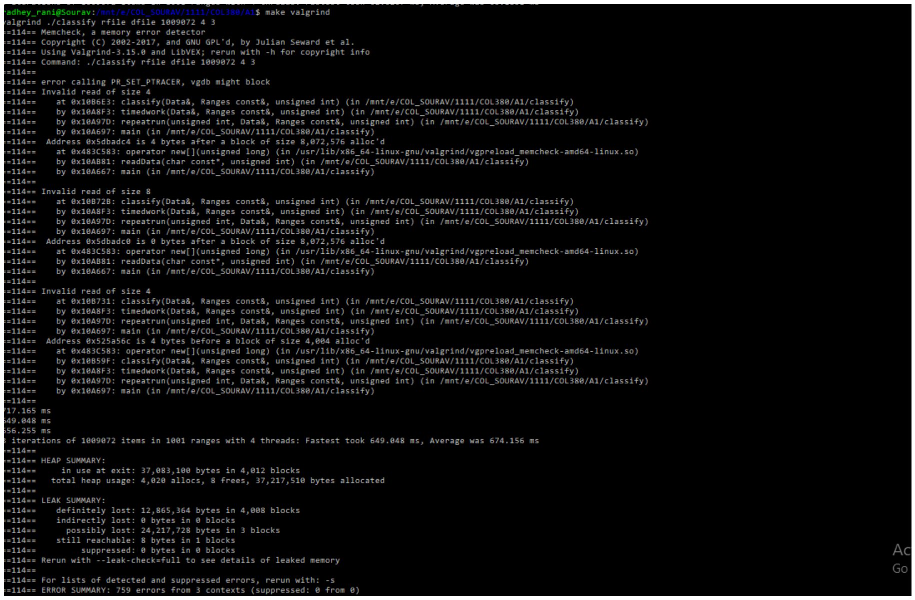
In the above image shown that the command use make gprof for analysis give us the output file analysis_gprof.out that contains the analysis of the code that contains number of function calls happened on a function and their time taken by them.



```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        3     0.00     0.00  classify(Data&, Ranges const&, unsigned int)
  0.00      0.00     0.00        3     0.00     0.00  timedwork(Data&, Ranges const&, unsigned int)
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj
^L
                        Call graph


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       3/3           timedwork(Data&, Ranges const&, unsigned int) [9]
[8]      0.0    0.00    0.00       3         classify(Data&, Ranges const&, unsigned int) [8]
-----------------------------------------------
                0.00    0.00       3/3           repeatrun(unsigned int, Data&, Ranges const&, unsigned int) [14]
[9]      0.0    0.00    0.00       3         timedwork(Data&, Ranges const&, unsigned int) [9]
                0.00    0.00       3/3           classify(Data&, Ranges const&, unsigned int) [8]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [18]
[10]     0.0    0.00    0.00       1         _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj [10]
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [18]
[11]     0.0    0.00    0.00       1         _GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj [11]
-----------------------------------------------
^L
Index by function name

  [10] _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj [8] classify(Data&, Ranges const&, unsigned int)
  [11] _GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj [9] timedwork(Data&, Ranges const&, unsigned int)
```

In the above shown image this is the analysis report when we call the gprof function in this file we can easily able to see that the data analysis shown in different profile i.e call tree and flat we are given the time in seconds of the various functions and the number of calls made to the function.In this we can able to see the child parent relations of the functions and how many times the parent and the child called in the program.

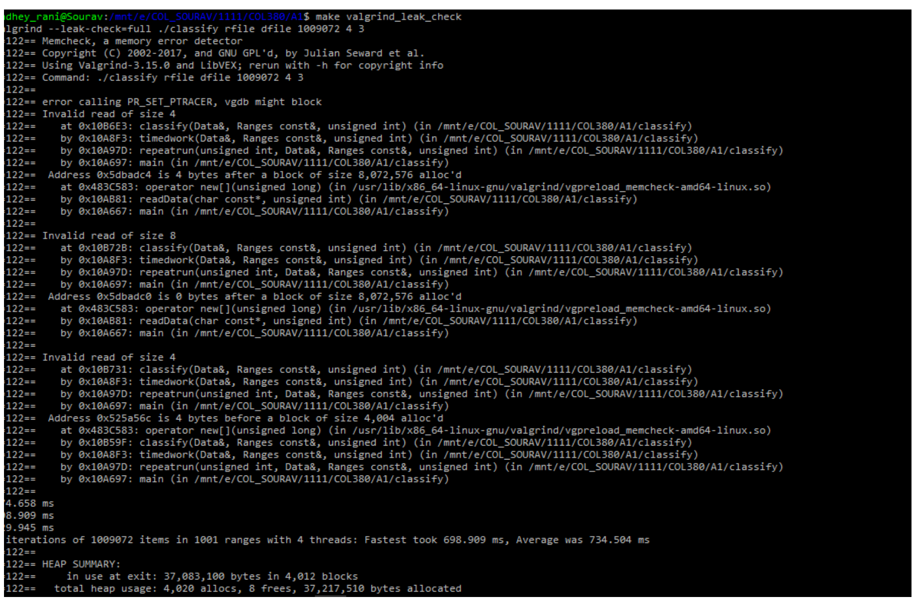
5.3 Valgrind Analysis:



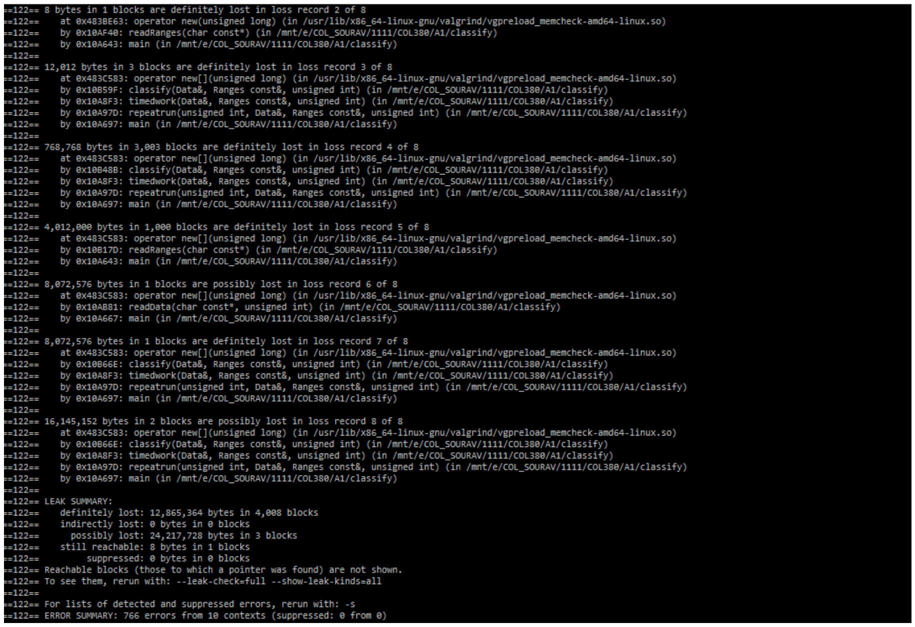
```
adhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$ make valgrind
valgrind ./classify rfile dfile 1009072 4 3
==114== Memcheck, a memory error detector
==114== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==114== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==114== Command: ./classify rfile dfile 1009072 4 3
==114==
==114== error calling PR_SET_PTRACER, vgdb might block
==114== Invalid read of size 4
==114==    at 0x10B6E3: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==  Address 0x5dbadc4 is 4 bytes after a block of size 8,072,576 alloc'd
==114==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==114==    by 0x10AB81: readData(char const*, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A667: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==
==114== Invalid read of size 8
==114==    at 0x10B72B: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==  Address 0x5dbadc0 is 0 bytes after a block of size 8,072,576 alloc'd
==114==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==114==    by 0x10AB81: readData(char const*, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A667: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==
==114== Invalid read of size 4
==114==    at 0x10B731: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==  Address 0x525a56c is 4 bytes before a block of size 4,004 alloc'd
==114==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==114==    by 0x10B59F: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==114==
717.165 ms
649.048 ms
656.255 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 649.048 ms, Average was 674.156 ms
==114==
==114== HEAP SUMMARY:
==114==     in use at exit: 37,083,100 bytes in 4,012 blocks
==114==   total heap usage: 4,020 allocs, 8 frees, 37,217,510 bytes allocated
==114==
==114== LEAK SUMMARY:
==114==    definitely lost: 12,865,364 bytes in 4,008 blocks
==114==    indirectly lost: 0 bytes in 0 blocks
==114==      possibly lost: 24,217,728 bytes in 3 blocks
==114==    still reachable: 8 bytes in 1 blocks
==114==         suppressed: 0 bytes in 0 blocks
==114== Rerun with --leak-check=full to see details of leaked memory
==114==
==114== For lists of detected and suppressed errors, rerun with: -s
==114== ERROR SUMMARY: 759 errors from 3 contexts (suppressed: 0 from 0)
```

In the above image we can see hat we execute the command for just valgrind report then we can get the data for heap summary used and the leak summary and also the execution time taken by the 4 threads for 3 iterations.

5.4 Valgrind_leak Analysis:



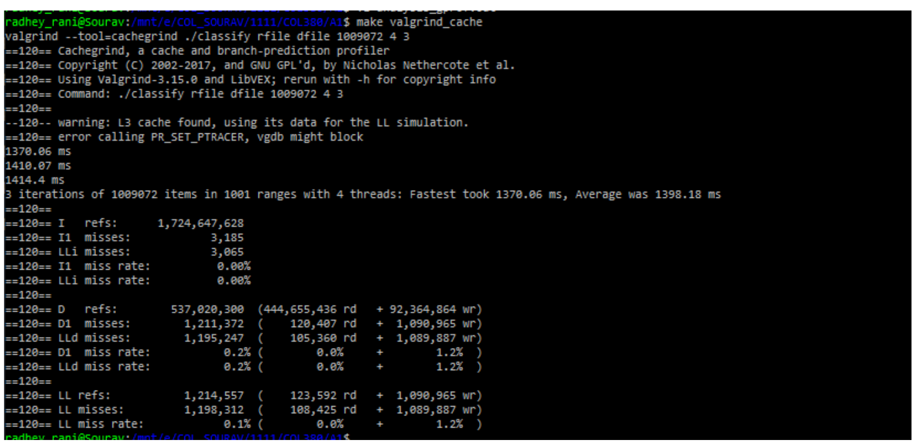
```
dhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$ make valgrind_leak_check
lgrind --leak-check=full ./classify rfile dfile 1009072 4 3
122== Memcheck, a memory error detector
122== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
122== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
122== Command: ./classify rfile dfile 1009072 4 3
122==
122== error calling PR_SET_PTRACER, vgdb might block
122== Invalid read of size 4
122==    at 0x10B6E3: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==  Address 0x5dbadc4 is 4 bytes after a block of size 8,072,576 alloc'd
122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
122==    by 0x10AB81: readData(char const*, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A667: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==
122== Invalid read of size 8
122==    at 0x10B72B: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==  Address 0x5dbadc0 is 0 bytes after a block of size 8,072,576 alloc'd
122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
122==    by 0x10AB81: readData(char const*, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A667: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==
122== Invalid read of size 4
122==    at 0x10B731: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==  Address 0x525a56c is 4 bytes before a block of size 4,004 alloc'd
122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
122==    by 0x10B59F: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
122==
4.658 ms
8.909 ms
9.945 ms
iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 698.909 ms, Average was 734.504 ms
122==
122== HEAP SUMMARY:
122==     in use at exit: 37,083,100 bytes in 4,012 blocks
122==   total heap usage: 4,020 allocs, 8 frees, 37,217,510 bytes allocated
```



```
==122== 8 bytes in 1 blocks are definitely lost in loss record 2 of 8
==122==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==122==    by 0x10AF40: readRanges(char const*) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A643: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==
==122== 12,012 bytes in 3 blocks are definitely lost in loss record 3 of 8
==122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==122==    by 0x10B59F: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==
==122== 768,768 bytes in 3,003 blocks are definitely lost in loss record 4 of 8
==122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==122==    by 0x10B48B: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==
==122== 4,012,000 bytes in 1,000 blocks are definitely lost in loss record 5 of 8
==122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==122==    by 0x10B17D: readRanges(char const*) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A643: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==
==122== 8,072,576 bytes in 1 blocks are possibly lost in loss record 6 of 8
==122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==122==    by 0x10AB81: readData(char const*, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A667: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==
==122== 8,072,576 bytes in 1 blocks are definitely lost in loss record 7 of 8
==122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==122==    by 0x10B66E: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==
==122== 16,145,152 bytes in 2 blocks are possibly lost in loss record 8 of 8
==122==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==122==    by 0x10B66E: classify(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A8F3: timedwork(Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A97D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==    by 0x10A697: main (in /mnt/e/COL_SOURAV/1111/COL380/A1/classify)
==122==
==122== LEAK SUMMARY:
==122==    definitely lost: 12,865,364 bytes in 4,008 blocks
==122==    indirectly lost: 0 bytes in 0 blocks
==122==      possibly lost: 24,217,728 bytes in 3 blocks
==122==    still reachable: 8 bytes in 1 blocks
==122==         suppressed: 0 bytes in 0 blocks
==122== Reachable blocks (those to which a pointer was found) are not shown.
==122== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==122==
==122== For lists of detected and suppressed errors, rerun with: -s
==122== ERROR SUMMARY: 766 errors from 10 contexts (suppressed: 0 from 0)
```

In the above image we can see hat we execute the command for valgrind full memory leak report then we can get the data for leak memory details and also the execution time taken by the 4 threads for 3 iterations.

5.5 Valgrind_cache Analysis:



```
radhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$ make valgrind_cache
valgrind --tool=cachegrind ./classify rfile dfile 1009072 4 3
==120== Cachegrind, a cache and branch-prediction profiler
==120== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==120== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==120== Command: ./classify rfile dfile 1009072 4 3
==120==
--120-- warning: L3 cache found, using its data for the LL simulation.
==120== error calling PR_SET_PTRACER, vgdb might block
1370.06 ms
1410.07 ms
1414.4 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 1370.06 ms, Average was 1398.18 ms
==120==
==120== I   refs:      1,724,647,628
==120== I1  misses:            3,185
==120== LLi misses:            3,065
==120== I1  miss rate:          0.00%
==120== LLi miss rate:          0.00%
==120==
==120== D   refs:        537,020,300  (444,655,436 rd   + 92,364,864 wr)
==120== D1  misses:        1,211,372  (    120,407 rd   +  1,090,965 wr)
==120== LLd misses:        1,195,247  (    105,360 rd   +  1,089,887 wr)
==120== D1  miss rate:           0.2% (        0.0%     +        1.2%  )
==120== LLd miss rate:           0.2% (        0.0%     +        1.2%  )
==120==
==120== LL refs:           1,214,557  (    123,592 rd   +  1,090,965 wr)
==120== LL misses:         1,198,312  (    108,425 rd   +  1,089,887 wr)
==120== LL miss rate:            0.1% (        0.0%     +        1.2%  )
radhey_rani@Sourav:/mnt/e/COL_SOURAV/1111/COL380/A1$
```

In the above image we can see hat we execute the command for valgrind cache report then we can get the data for cache details and also the execution time taken by the 4 threads for 3 iterations and also in this we are able to see that I refs which will tell about the number of instruction that are executed during the running of the code and I1 misses tells about the cache read misses during the running of the code and also LLi misses tells us about the instruction read misses

during the running of the code. and after that gives the data in percentage(for both LLi and I1).
In 2nd part the D refs tells us about the number of memory read and write misses during the running of the code and D1 misses tells us about the memory read and write misses and also in their bracket tells us the how many read and write calls are happend in the memory. LLd misses tells us about the how many memory read and write calls are missed after that the data is given in the percentage for both read and write command.
In 3rd part tells us about the total read and write data is misses during the execution of the read and write operations are happens and the data given in the percent.
And comparing our optimized cache behaviour with the original cache behaviour we see that with time the instruction call is also reduced up to 9 times and also able to see that the misses instruction is also less than the original one i.e when we talk in percent firstly it was approx 5.4% miss rate and now we can easily see that it is approximately zero.