MODIFIED CODE: GPROF AND VALGRIND REPORT ANALYSIS

This file consists of the analysis of gprof and valgrind reports of the modified code. First of all, we have to put (return *this) in line 95 of our classify.h file. After that we have to compile our files using make command and then run our files using make run command. Our modified code was showing giving output like:

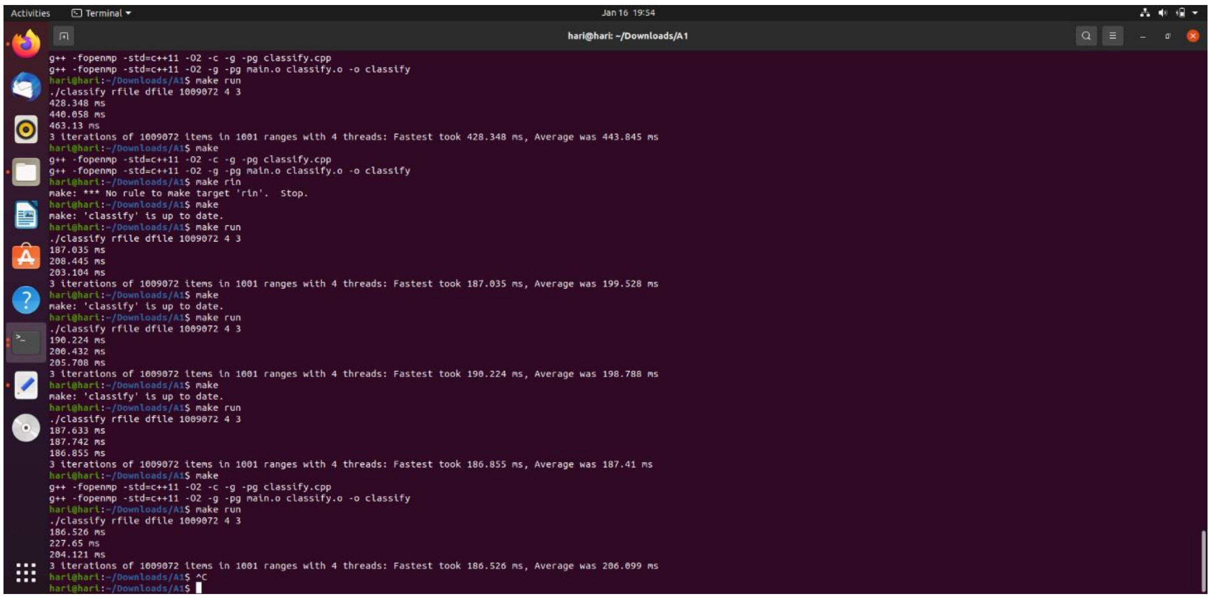
186.526 ms

227.65 ms

204.121 ms

3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 186.526 ms, Average was 206.099 ms

Which is also shown in the image below:



```
hari@hari: ~/Downloads/A1
g++ -fopenmp -std=c++11 -O2 -c -g -pg classify.cpp
g++ -fopenmp -std=c++11 -O2 -g -pg main.o classify.o -o classify
hari@hari:~/Downloads/A1$ make run
./classify rfile dfile 1009072 4 3
428.348 ms
440.058 ms
463.13 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 428.348 ms, Average was 443.845 ms
hari@hari:~/Downloads/A1$ make
g++ -fopenmp -std=c++11 -O2 -c -g -pg classify.cpp
g++ -fopenmp -std=c++11 -O2 -g -pg main.o classify.o -o classify
hari@hari:~/Downloads/A1$ make rin
make: *** No rule to make target 'rin'.  Stop.
hari@hari:~/Downloads/A1$ make
make: 'classify' is up to date.
hari@hari:~/Downloads/A1$ make run
./classify rfile dfile 1009072 4 3
187.035 ms
208.445 ms
203.104 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 187.035 ms, Average was 199.528 ms
hari@hari:~/Downloads/A1$ make
make: 'classify' is up to date.
hari@hari:~/Downloads/A1$ make run
./classify rfile dfile 1009072 4 3
190.224 ms
200.432 ms
205.708 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 190.224 ms, Average was 198.788 ms
hari@hari:~/Downloads/A1$ make
make: 'classify' is up to date.
hari@hari:~/Downloads/A1$ make run
./classify rfile dfile 1009072 4 3
187.633 ms
187.742 ms
186.855 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 186.855 ms, Average was 187.41 ms
hari@hari:~/Downloads/A1$ make
g++ -fopenmp -std=c++11 -O2 -c -g -pg classify.cpp
g++ -fopenmp -std=c++11 -O2 -g -pg main.o classify.o -o classify
hari@hari:~/Downloads/A1$ make run
./classify rfile dfile 1009072 4 3
186.526 ms
227.65 ms
204.121 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 186.526 ms, Average was 206.099 ms
hari@hari:~/Downloads/A1$ ^C
hari@hari:~/Downloads/A1$
```

GPROF ANALYSIS

For running gprof, we just have to put gprof in the command and deviate the analysis of gprof report i.e. gmon.out in our gprof_report.out file. The below text shows the result we got after we have used gprof i.e. basically the result of our gprof_report.out file:

....

....

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self           self     total
 time   seconds   seconds   calls  ms/call  ms/call  name
 98.04     1.33    1.33                        readRanges(char const*)
  0.74     1.34    0.01       3     3.34     3.34  classify(Data&, Ranges const&,
unsigned int)
  0.74     1.35    0.01                        readData(char const*, unsigned int)
  0.74     1.36    0.01                        repeatrun(unsigned int, Data&, Ranges
const&, unsigned int)
  0.00     1.36    0.00       3     0.00     3.34  timedwork(Data&, Ranges const&,
unsigned int)
  0.00     1.36    0.00       1     0.00     0.00
_GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj
  0.00     1.36    0.00       1     0.00     0.00
_GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj
 %         the percentage of the total running time of the
time       program used by this function.
cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.
 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.
calls      the number of times this function was invoked, if
           this function is profiled, else blank.
 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.
 total     the average number of milliseconds spent in this
```

```
ms/call    function and its descendents per call, if this
             function is profiled, else blank.
name       the name of the function.  This is the minor sort
          for this listing. The index shows the location of
             the function in the gprof listing. If the index is
             in parenthesis it shows where it would appear in the gprof listing if it
were to be printed.
```

```
                          Call graph (explanation follows)
granularity: each sample hit covers 2 byte(s) for 0.73% of 1.36 seconds
index % time    self  children    called     name
                                             <spontaneous>
[1]     97.8    1.33    0.00                 readRanges(char const*) [1]
-----------------------------------------------
                                             <spontaneous>
[2]      1.5    0.01    0.01                repeatrun(unsigned int, Data&, Ranges const&,
unsigned int) [2]
              0.00    0.01       3/3          timedwork(Data&, Ranges const&, unsigned
int) [4]
-----------------------------------------------
              0.01    0.00       3/3          timedwork(Data&, Ranges const&, unsigned
int) [4]
[3]      0.7    0.01    0.00       3        classify(Data&, Ranges const&, unsigned int)
[3]
```

```
-----------------------------------------------

             0.00    0.01       3/3           repeatrun(unsigned int, Data&, Ranges
const&, unsigned int) [2]

[4]      0.7    0.00    0.01       3         timedwork(Data&, Ranges const&, unsigned
int) [4]

             0.01    0.00       3/3           classify(Data&, Ranges const&, unsigned int)
[3]

-----------------------------------------------

                                          <spontaneous>

[5]      0.7    0.01    0.00                readData(char const*, unsigned int) [5]

-----------------------------------------------

             0.00    0.00       1/1           __libc_csu_init [18]

[13]     0.0    0.00    0.00       1
_GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj [13]

-----------------------------------------------

             0.00    0.00       1/1           __libc_csu_init [18]

[14]     0.0    0.00    0.00       1
_GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj [14]

-----------------------------------------------


 This table describes the call tree of the program, and was sorted by

 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the

 index number at the left hand margin lists the current function.

 The lines above it list the functions that called this function,

 and the lines below it list the functions this one called.

 This line lists:

     index       A unique number given to each element of the table.
```

Index numbers are sorted numerically.

The index number is printed next to every function name so it is easier to look up where the function is in the table.

% time This is the percentage of the `total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

selfThis is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

selfThis is the amount of time that was propagated directly from the function into this parent.

children	This is the amount of time that was propagated from the function's children into this parent.
called	This is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'.
name	This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self	This is the amount of time that was propagated directly from the child into the function.
children	This is the amount of time that was propagated from the child's children to the function.
called	This is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.
name	This is the name of the child. The child's index

```
                    number is printed after it.  If the child is a
                    member of a cycle, the cycle number is printed
                    between the name and the index number.
 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
Copyright (C) 2012-2020 Free Software Foundation, Inc.
Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
Index by function name
  [13] _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj (classify.cpp) [3]
classify(Data&, Ranges const&, unsigned int) [4] timedwork(Data&, Ranges
const&, unsigned int)
  [14] _GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj (main.cpp) [5]
readData(char const*, unsigned int)
   [1] readRanges(char const*) [2] repeatrun(unsigned int, Data&, Ranges
const&, unsigned int)
*************************************************************
```

Now for analysis of this gprof_report file we can see that initially in the flat profile we are given the time in seconds of the various functions and the number of calls made to the function. We conclude from there that readRanges(char const*) function is the function which is taking 97.8% of time and time taken is 1.33 seconds which is obviously less than the time taken by the original code. Hence, we have somewhat optimized this function and its callee's in our modified code.

Next, we have to analyse the call tree of the program. In this we can see that we are given an index to all the functions and just tell about which function calls which other functions i.e. parent and child functions are also there. In this part of the file also we can see that we have slightly optimized our functions as now it is making less or same number of calls but the time taken is reduced somewhat.

VALGRIND ANALYSIS

Now we have to analyse valgrind report of our modified code and for this we will be analysing two of the main aspects of our valgrind report which are:

Memory leaking:

For this we will be using --leak-check=full to see all the memory checkings i.e. all the loss records of memory and the leak summary of the file.

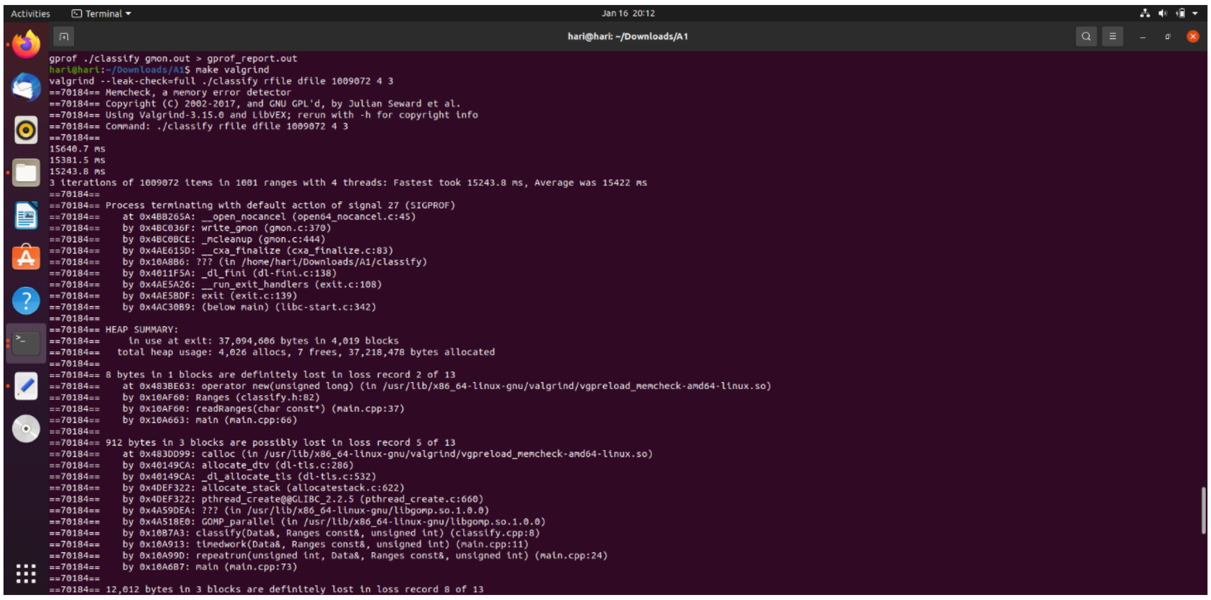
15640.7 ms

15381.5 ms

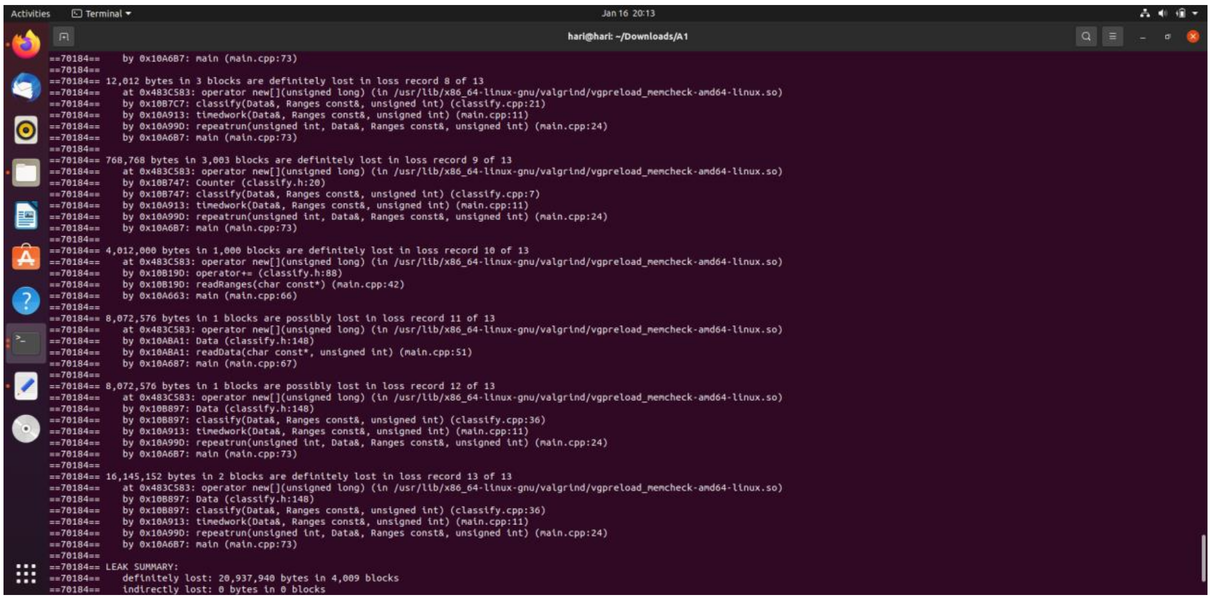
15243.8 ms

3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 15243.8 ms, Average was 15422 ms. This output shows the time taken for the execution of valgrind in the three iterations in our modified code and we can see that the time taken is less than the original code and hence our modified code have executed the valgrind quickly.

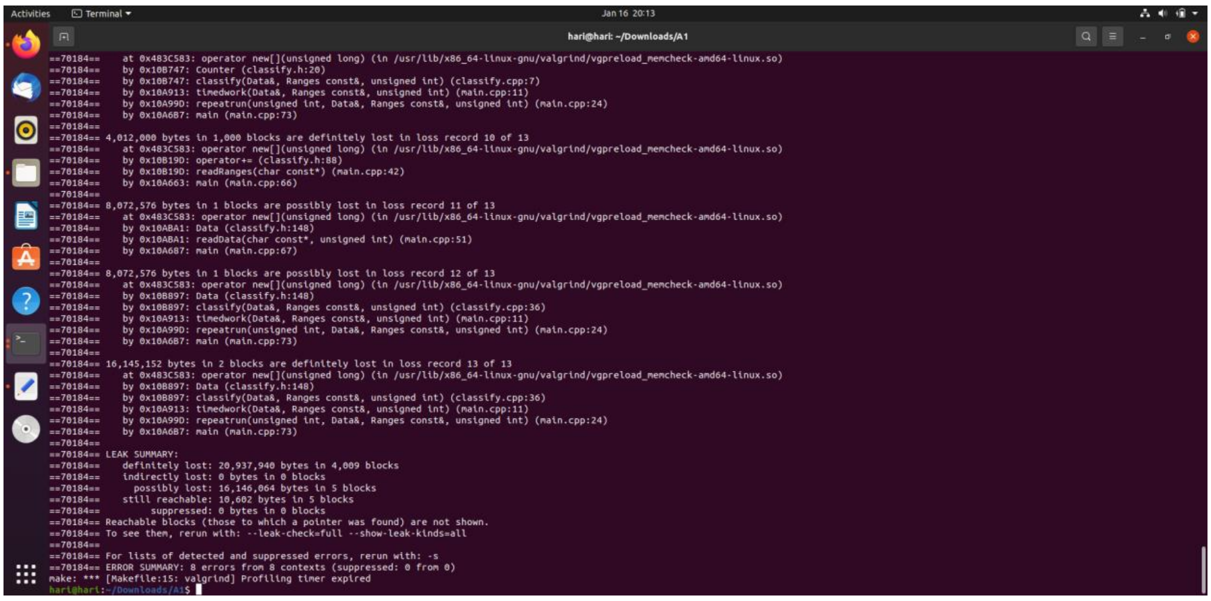
The below snapshot shows the result of the above command:



```
gprof ./classify gmon.out > gprof_report.out
hari@hari:~/Downloads/A1$ make valgrind
valgrind --leak-check=full ./classify rfile dfile 1009072 4 3
==70184== Memcheck, a memory error detector
==70184== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==70184== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==70184== Command: ./classify rfile dfile 1009072 4 3
==70184==
15640.7 ms
15381.5 ms
15243.8 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 15243.8 ms, Average was 15422 ms
==70184==
==70184== Process terminating with default action of signal 27 (SIGPROF)
==70184==    at 0x4BB265A: __open_nocancel (open64_nocancel.c:45)
==70184==    by 0x4BC036F: write_gmon (gmon.c:370)
==70184==    by 0x4BC0BCE: _mcleanup (gmon.c:444)
==70184==    by 0x4AE615D: __cxa_finalize (cxa_finalize.c:83)
==70184==    by 0x10A8B6: ??? (in /home/hari/Downloads/A1/classify)
==70184==    by 0x4011F5A: _dl_fini (dl-fini.c:138)
==70184==    by 0x4AE5A26: __run_exit_handlers (exit.c:108)
==70184==    by 0x4AE5BDF: exit (exit.c:139)
==70184==    by 0x4AC30B9: (below main) (libc-start.c:342)
==70184==
==70184== HEAP SUMMARY:
==70184==     in use at exit: 37,094,606 bytes in 4,019 blocks
==70184==   total heap usage: 4,026 allocs, 7 frees, 37,218,478 bytes allocated
==70184==
==70184== 8 bytes in 1 blocks are definitely lost in loss record 2 of 13
==70184==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10AF60: Ranges (classify.h:82)
==70184==    by 0x10AF60: readRanges(char const*) (main.cpp:37)
==70184==    by 0x10A663: main (main.cpp:66)
==70184==
==70184== 912 bytes in 3 blocks are possibly lost in loss record 5 of 13
==70184==    at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x40149CA: allocate_dtv (dl-tls.c:286)
==70184==    by 0x40149CA: _dl_allocate_tls (dl-tls.c:532)
==70184==    by 0x4DEF322: allocate_stack (allocatestack.c:622)
==70184==    by 0x4DEF322: pthread_create@@GLIBC_2.2.5 (pthread_create.c:660)
==70184==    by 0x4A59DEA: ??? (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
==70184==    by 0x4A518E0: GOMP_parallel (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
==70184==    by 0x10B7A3: classify(Data&, Ranges const&, unsigned int) (classify.cpp:8)
==70184==    by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==70184==    by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== 12,012 bytes in 3 blocks are definitely lost in loss record 8 of 13
```



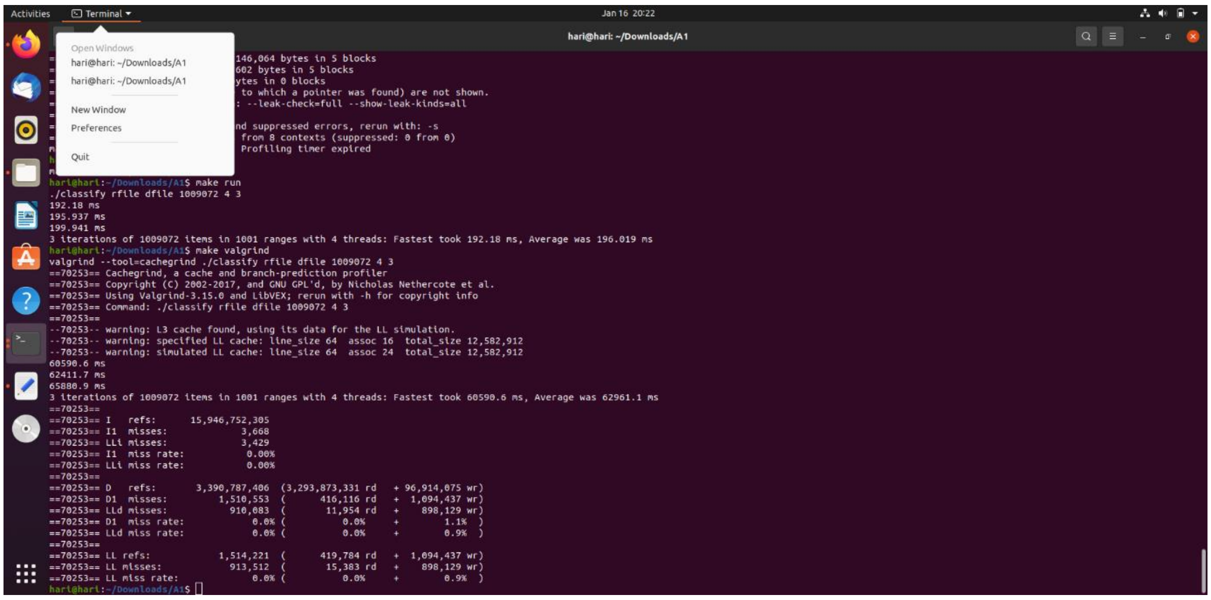
```
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== 12,012 bytes in 3 blocks are definitely lost in loss record 8 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B7C7: classify(Data&, Ranges const&, unsigned int) (classify.cpp:21)
==70184==    by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==70184==    by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== 768,768 bytes in 3,003 blocks are definitely lost in loss record 9 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B747: Counter (classify.h:20)
==70184==    by 0x10B747: classify(Data&, Ranges const&, unsigned int) (classify.cpp:7)
==70184==    by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==70184==    by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== 4,012,000 bytes in 1,000 blocks are definitely lost in loss record 10 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B19D: operator+= (classify.h:88)
==70184==    by 0x10B19D: readRanges(char const*) (main.cpp:42)
==70184==    by 0x10A663: main (main.cpp:66)
==70184==
==70184== 8,072,576 bytes in 1 blocks are possibly lost in loss record 11 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10ABA1: Data (classify.h:148)
==70184==    by 0x10ABA1: readData(char const*, unsigned int) (main.cpp:51)
==70184==    by 0x10A687: main (main.cpp:67)
==70184==
==70184== 8,072,576 bytes in 1 blocks are possibly lost in loss record 12 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B897: Data (classify.h:148)
==70184==    by 0x10B897: classify(Data&, Ranges const&, unsigned int) (classify.cpp:36)
==70184==    by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==70184==    by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== 16,145,152 bytes in 2 blocks are definitely lost in loss record 13 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B897: Data (classify.h:148)
==70184==    by 0x10B897: classify(Data&, Ranges const&, unsigned int) (classify.cpp:36)
==70184==    by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==70184==    by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== LEAK SUMMARY:
==70184==    definitely lost: 20,937,940 bytes in 4,009 blocks
==70184==    indirectly lost: 0 bytes in 0 blocks
```



```
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B747: Counter (classify.h:20)
==70184==    by 0x10B747: classify(Data&, Ranges const&, unsigned int) (classify.cpp:7)
==70184==    by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==70184==    by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== 4,012,000 bytes in 1,000 blocks are definitely lost in loss record 10 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B19D: operator+= (classify.h:88)
==70184==    by 0x10B19D: readRanges(char const*) (main.cpp:42)
==70184==    by 0x10A663: main (main.cpp:66)
==70184==
==70184== 8,072,576 bytes in 1 blocks are possibly lost in loss record 11 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10ABA1: Data (classify.h:148)
==70184==    by 0x10ABA1: readData(char const*, unsigned int) (main.cpp:51)
==70184==    by 0x10A687: main (main.cpp:67)
==70184==
==70184== 8,072,576 bytes in 1 blocks are possibly lost in loss record 12 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B897: Data (classify.h:148)
==70184==    by 0x10B897: classify(Data&, Ranges const&, unsigned int) (classify.cpp:36)
==70184==    by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==70184==    by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== 16,145,152 bytes in 2 blocks are definitely lost in loss record 13 of 13
==70184==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==70184==    by 0x10B897: Data (classify.h:148)
==70184==    by 0x10B897: classify(Data&, Ranges const&, unsigned int) (classify.cpp:36)
==70184==    by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==70184==    by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==70184==    by 0x10A6B7: main (main.cpp:73)
==70184==
==70184== LEAK SUMMARY:
==70184==    definitely lost: 20,937,940 bytes in 4,009 blocks
==70184==    indirectly lost: 0 bytes in 0 blocks
==70184==      possibly lost: 16,146,064 bytes in 5 blocks
==70184==    still reachable: 10,602 bytes in 5 blocks
==70184==         suppressed: 0 bytes in 0 blocks
==70184== Reachable blocks (those to which a pointer was found) are not shown.
==70184== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==70184==
==70184== For lists of detected and suppressed errors, rerun with: -s
==70184== ERROR SUMMARY: 8 errors from 8 contexts (suppressed: 0 from 0)
make: *** [Makefile:15: valgrind] Profiling timer expired
hari@hari:~/Downloads/A1$
```

Cache related details

For this we will be using --tool=cachegrind to see all the cache related details i.e. the accessing of the various cache lines. It simulates the machine with first level instructions and data caches I1 and D1 backed by D2. So, in our command we get all the information like number of instructions missed, executed etc. and also gives us the percentage of missing rates in different caches. The below snapshot shows the result of the above command:


Activities
Terminal
Jan 16 20:22
hari@hari: ~/Downloads/A1
Open Windows
hari@hari: ~/Downloads/A1
hari@hari: ~/Downloads/A1
New Window
Preferences
Quit

```
146,064 bytes in 5 blocks
602 bytes in 5 blocks
ytes in 0 blocks
 to which a pointer was found) are not shown.
: --leak-check=full --show-leak-kinds=all

nd suppressed errors, rerun with: -s
 from 8 contexts (suppressed: 0 from 0)
 Profiling timer expired
hari@hari:~/Downloads/A1$ make run
./classify rfile dfile 1009072 4 3
192.18 ms
195.937 ms
199.941 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 192.18 ms, Average was 196.019 ms
hari@hari:~/Downloads/A1$ make valgrind
valgrind --tool=cachegrind ./classify rfile dfile 1009072 4 3
==70253== Cachegrind, a cache and branch-prediction profiler
==70253== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==70253== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==70253== Command: ./classify rfile dfile 1009072 4 3
==70253==
--70253-- warning: L3 cache found, using its data for the LL simulation.
--70253-- warning: specified LL cache: line_size 64  assoc 16  total_size 12,582,912
--70253-- warning: simulated LL cache: line_size 64  assoc 24  total_size 12,582,912
60590.6 ms
62411.7 ms
65880.9 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 60590.6 ms, Average was 62961.1 ms
==70253==
==70253== I   refs:      15,946,752,305
==70253== I1  misses:             3,668
==70253== LLi misses:             3,429
==70253== I1  miss rate:           0.00%
==70253== LLi miss rate:           0.00%
==70253==
==70253== D   refs:       3,390,787,406  (3,293,873,331 rd   + 96,914,075 wr)
==70253== D1  misses:         1,510,553  (      416,116 rd   +  1,094,437 wr)
==70253== LLd misses:           910,083  (       11,954 rd   +    898,129 wr)
==70253== D1  miss rate:            0.0% (          0.0%     +        1.1%  )
==70253== LLd miss rate:            0.0% (          0.0%     +        0.9%  )
==70253==
==70253== LL refs:            1,514,221  (      419,784 rd   +  1,094,437 wr)
==70253== LL misses:            913,512  (       15,383 rd   +    898,129 wr)
==70253== LL miss rate:             0.0% (          0.0%     +        0.9%  )
hari@hari:~/Downloads/A1$
```