

ORIGINAL CODE: GPROF AND VALGRIND REPORT ANALYSIS

This file consists of the analysis of gprof and valgrind reports of the original 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 original code was showing giving output like:

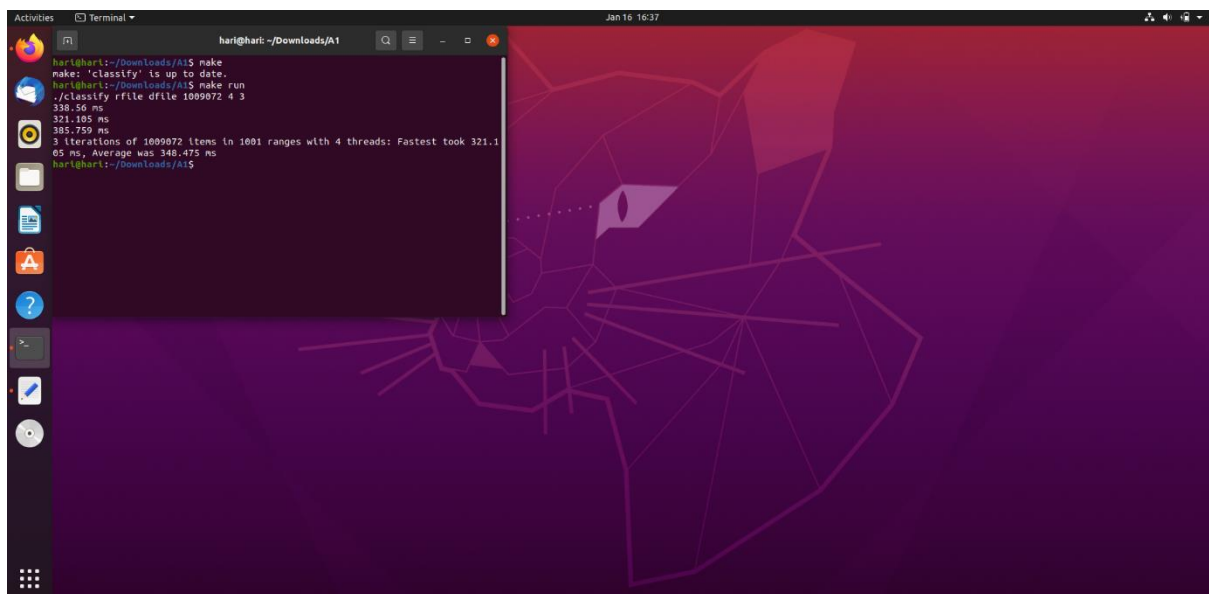
338.56 ms

321.105 ms

385.759 ms

3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 321.105 ms, Average was 348.475 ms.

Which is also shown in the image below:



```
hari@hari:~/Downloads/A1$ make
make: 'classify' is up to date.
hari@hari:~/Downloads/A1$ make run
./classify rfile dfile 1009072 4 3
338.56 ms
321.105 ms
385.759 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 321.105 ms, Average was 348.475 ms
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 | Ts/call | Ts/call | name |
| 100.27 | 3.59 | 3.59 | | | | readRanges(char const*) |
| 0.00 | 3.59 | 0.00 | 3 | 0.00 | 0.00 | classify(Data&, Ranges const&, unsigned int) |
| 0.00 | 3.59 | 0.00 | 3 | 0.00 | 0.00 | timedwork(Data&, Ranges const&, unsigned int) |
| 0.00 | 3.59 | 0.00 | 1 | 0.00 | 0.00 | _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj |
| 0.00 | 3.59 | 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.

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

Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.28% of 3.59 seconds

index % time self children called name

<spontaneous>

[1] 100.0 3.59 0.00 readRanges(char const*) [1]

0.00 0.00 3/3 timedwork(Data&, Ranges const&, unsigned
int) [10]

[9] 0.0 0.00 0.00 3 classify(Data&, Ranges const&, unsigned int)
[9]

0.00 0.00 3/3 repeatrun(unsigned int, Data&, Ranges
const&, unsigned int) [14]

[10] 0.0 0.00 0.00 3 timedwork(Data&, Ranges const&, unsigned
int) [10]

0.00 0.00 3/3 classify(Data&, Ranges const&, unsigned int)
[9]

```

0.00  0.00  1/1  __libc_csu_init [18]
[11]  0.0  0.00  0.00  1
_GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj [11]
-----
0.00  0.00  1/1  __libc_csu_init [18]
[12]  0.0  0.00  0.00  1
_GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj [12]
-----

```

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:

self This 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

[11] _GLOBAL__sub_I__Z8classifyR4DataRK6Rangesj (classify.cpp) [1]
readRanges(char const*) [10] timedwork(Data&, Ranges const&, unsigned int)

[12] _GLOBAL__sub_I__Z9timedworkR4DataRK6Rangesj (main.cpp) [9]
classify(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 almost all the time i.e. 100.27% time and time taken is 3.59 seconds. Hence, we have to basically optimize 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 tells about which function calls which other functions i.e. parent and child functions are also there.

VALGRIND ANALYSIS

Now we have to analyze valgrind report of our original 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 checking i.e. all the loss records of memory and the leak summary of the file.

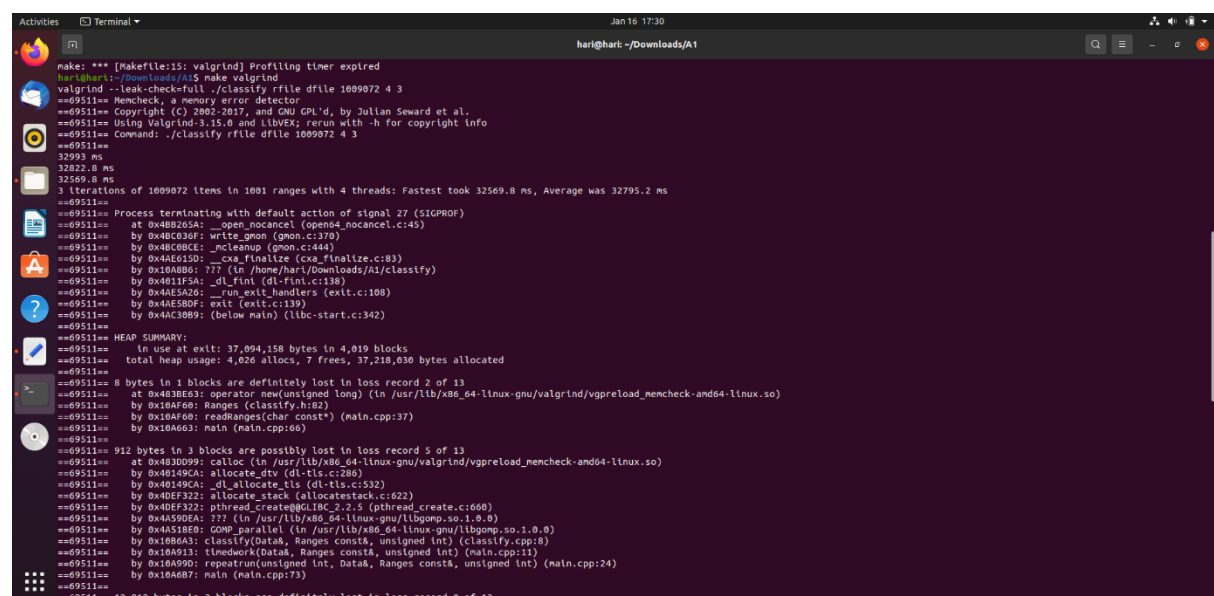
32993 ms

32822.8 ms

32569.8 ms

3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 32569.8 ms, Average was 32795.2 ms. This output shows the time taken for the execution of valgrind in the three iterations.

The below snapshots show the result of the above command:

A terminal window showing the output of a valgrind command. The output includes the valgrind version (3.15.0), the command being run (make valgrind), and the results of three iterations of a program. The results show the fastest iteration took 32569.8 ms, the average was 32795.2 ms, and the total heap usage was 4,026 allocs, 7 frees, 37,218,030 bytes allocated. The output also shows several memory leaks, including 8 bytes in 1 block, 912 bytes in 3 blocks, and 12,012 bytes in 3 blocks, all of which are definitely lost. The terminal window has a title bar with 'Activities', 'Terminal', and 'Jan 16 17:30'. The user is 'hari@hari: ~/Downloads/A1'.

```
make: *** [Makefile:15: valgrind] Profiling timer expired
hari@hari:~/Downloads/A1$ make valgrind
valgrind --leak-check=full ./classify rfile dfile 1009072 4 3
==69511== Memcheck, a memory error detector
==69511== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==69511== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==69511== Command: ./classify rfile dfile 1009072 4 3
==69511==
32993 ms
32822.8 ms
32569.8 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 32569.8 ms, Average was 32795.2 ms
==69511==
==69511== Process terminating with default action of signal 27 (SIGPROF)
==69511== at 0x4BB265A: __open_nocancel (open64_nocancel.c:45)
==69511== by 0x4B8C86F: write_gmon (gmon.c:370)
==69511== by 0x4B8C8CE: _ncleanup (gmon.c:444)
==69511== by 0x4AE615D: __cxa_finalize (cxa_finalize.c:83)
==69511== by 0x10A8B6: ??? (in /home/hari/Downloads/A1/classify)
==69511== by 0x48115A: _dl_fini (dl-fini.c:130)
==69511== by 0x4AE5A26: __run_exit_handlers (exit.c:108)
==69511== by 0x4AE5BDF: exit (exit.c:139)
==69511== by 0x4AC3B89: (below main) (libc-start.c:342)
==69511==
==69511== HEAP SUMMARY:
==69511==   in use at exit: 37,894,158 bytes in 4,019 blocks
==69511== total heap usage: 4,026 allocs, 7 frees, 37,218,030 bytes allocated
==69511==
==69511== 8 bytes in 1 blocks are definitely lost in loss record 2 of 13
==69511== at 0x4B8E631: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10AF00: Ranges (classify.h:82)
==69511== by 0x10AF00: readRanges(char const*) (main.cpp:37)
==69511== by 0x10A603: main (main.cpp:66)
==69511==
==69511== 912 bytes in 3 blocks are possibly lost in loss record 5 of 13
==69511== at 0x4B3D099: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x48149CA: allocate_dtv (dl-tls.c:286)
==69511== by 0x48149CA: _dl_allocate_tls (dl-tls.c:532)
==69511== by 0x4DEF322: allocate_stack (allocatetest.c:622)
==69511== by 0x4DEF322: pthread_create@GLIBC_2.2.5 (pthread_create.c:668)
==69511== by 0x4A59DEA: ??? (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
==69511== by 0x4A518E0: GOMP_parallel (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
==69511== by 0x10B6A3: classify(Data&, Ranges const&, unsigned int) (classify.cpp:8)
==69511== by 0x10A913: timedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==69511== by 0x10A99D: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== 12,012 bytes in 3 blocks are definitely lost in loss record 8 of 13
```



```
Jan 16 17:31
har@har: ~/Downloads/A1

==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== 12,012 bytes in 3 blocks are definitely lost in loss record 8 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B6C7: classify(Data&, Ranges const&, unsigned int) (classify.cpp:19)
==69511== by 0x10A913: tinedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==69511== by 0x10A990: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== 768,768 bytes in 3,003 blocks are definitely lost in loss record 9 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B6C7: classify(Data&, Ranges const&, unsigned int) (classify.cpp:7)
==69511== by 0x10A913: tinedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==69511== by 0x10A990: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== 4,012,000 bytes in 1,000 blocks are definitely lost in loss record 10 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B190: operator++ (classify.h:88)
==69511== by 0x10B190: readRanges(char const*) (main.cpp:42)
==69511== by 0x10A6B7: main (main.cpp:66)
==69511==
==69511== 8,072,576 bytes in 1 blocks are possibly lost in loss record 11 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10A8A1: readData(char const*, unsigned int) (main.cpp:51)
==69511== by 0x10A6B7: main (main.cpp:67)
==69511==
==69511== 8,072,576 bytes in 1 blocks are definitely lost in loss record 12 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B797: Data (classify.h:148)
==69511== by 0x10B797: classify(Data&, Ranges const&, unsigned int) (classify.cpp:34)
==69511== by 0x10A913: tinedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==69511== by 0x10A990: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== 16,145,152 bytes in 2 blocks are possibly lost in loss record 13 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B797: Data (classify.h:148)
==69511== by 0x10B797: classify(Data&, Ranges const&, unsigned int) (classify.cpp:34)
==69511== by 0x10A913: tinedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==69511== by 0x10A990: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== LEAK SUMMARY:
==69511== definitely lost: 12,865,364 bytes in 4,008 blocks
==69511== indirectly lost: 0 bytes in 0 blocks

Jan 16 17:31
har@har: ~/Downloads/A1

==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B6C7: Counter (classify.h:20)
==69511== by 0x10B6C7: classify(Data&, Ranges const&, unsigned int) (classify.cpp:7)
==69511== by 0x10A913: tinedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==69511== by 0x10A990: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== 4,012,000 bytes in 1,000 blocks are definitely lost in loss record 10 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B190: operator++ (classify.h:88)
==69511== by 0x10B190: readRanges(char const*) (main.cpp:42)
==69511== by 0x10A6B7: main (main.cpp:66)
==69511==
==69511== 8,072,576 bytes in 1 blocks are possibly lost in loss record 11 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10A8A1: Data (classify.h:148)
==69511== by 0x10A8A1: readData(char const*, unsigned int) (main.cpp:51)
==69511== by 0x10A6B7: main (main.cpp:67)
==69511==
==69511== 8,072,576 bytes in 1 blocks are definitely lost in loss record 12 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B797: Data (classify.h:148)
==69511== by 0x10B797: classify(Data&, Ranges const&, unsigned int) (classify.cpp:34)
==69511== by 0x10A913: tinedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==69511== by 0x10A990: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== 16,145,152 bytes in 2 blocks are possibly lost in loss record 13 of 13
==69511== at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-and64-linux.so)
==69511== by 0x10B797: Data (classify.h:148)
==69511== by 0x10B797: classify(Data&, Ranges const&, unsigned int) (classify.cpp:34)
==69511== by 0x10A913: tinedwork(Data&, Ranges const&, unsigned int) (main.cpp:11)
==69511== by 0x10A990: repeatrun(unsigned int, Data&, Ranges const&, unsigned int) (main.cpp:24)
==69511== by 0x10A6B7: main (main.cpp:73)
==69511==
==69511== LEAK SUMMARY:
==69511== definitely lost: 12,865,364 bytes in 4,008 blocks
==69511== indirectly lost: 0 bytes in 0 blocks
==69511== possibly lost: 24,218,640 bytes in 0 blocks
==69511== still reachable: 10,154 bytes in 5 blocks
==69511== suppressed: 0 bytes in 0 blocks
==69511==
==69511== Reachable blocks (those to which a pointer was found) are not shown.
==69511== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==69511==
==69511== For lists of detected and suppressed errors, rerun with: -s
==69511== ERROR SUMMARY: 8 errors from 8 contexts (suppressed: 0 from 0)
make: *** [makefile:15: valgrind] Profiling timer expired
har@har: ~/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 L1 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 results of the above command:

```
Activities Terminal Jan 16 17:52 harl@harl: ~/Downloads/A1

==09511== Indirectly lost: 0 bytes in 0 blocks
==09511== possibly lost: 24,219,640 bytes in 6 blocks
==09511== still reachable: 10,154 bytes in 5 blocks
==09511== suppressed: 0 bytes in 0 blocks
==09511== Reachable blocks (those to which a pointer was found) are not shown.
==09511== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==09511==
==09511== For lists of detected and suppressed errors, rerun with: -s
==09511== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make: *** [Makefile:15: valgrind] Profiling timer expired
harl@harl:~/Downloads/A1$ make valgrind
valgrind --tool=cachegrind ./classify rfile dfile 1009072 4 3
==09566== Cachegrind, a cache and branch-prediction profiler
==09566== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==09566== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==09566== Command: ./classify rfile dfile 1009072 4 3
--09566-- warning: L3 cache found, using its data for the LL simulation.
--09566-- warning: specified LL cache: line_size 64 assoc 16 total_size 12,582,912
--09566-- warning: simulated LL cache: line_size 64 assoc 24 total_size 12,582,912
138630 ms
138418 ms
116513 ms
3 iterations of 1009072 items in 1001 ranges with 4 threads: Fastest took 116513 ms, Average was 128522 ms
==09566==
==09566== Process terminating with default action of signal 27 (SIGPROF)
==09566== at 0x4A49132: ??? (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
==09566== by 0x4A46767: ??? (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
==09566== by 0x4006080: start_thread (pthread_create.c:477)
==09566== by 0x4B4B292: clone (clone.S:95)
==09566==
==09566== 1 refs: 34,135,983,520
==09566== LL misses: 3,488
==09566== LL1 misses: 3,268
==09566== LL miss rate: 0.00%
==09566== LL1 miss rate: 0.00%
==09566==
==09566== D refs: 6,436,089,939 (6,336,152,995 rd + 99,936,944 wr)
==09566== D1 misses: 381,920,327 ( 380,447,987 rd + 1,472,440 wr)
==09566== L1d misses: 1,315,313 ( 389,943 rd + 925,370 wr)
==09566== D1 miss rate: 5.9% ( 6.0% + 1.5% )
==09566== L1d miss rate: 0.0% ( 0.0% + 0.9% )
==09566==
==09566== LL refs: 381,923,815 ( 380,451,375 rd + 1,472,440 wr)
==09566== LL misses: 1,315,573 ( 392,383 rd + 923,190 wr)
==09566== LL miss rate: 0.0% ( 0.0% + 0.9% )
==09566==
make: *** [Makefile:15: valgrind] Profiling timer expired
harl@harl:~/Downloads/A1$
```