Exercise 02

Profiling

The profiling technique is used in order to understand which parts of one application are critical. In other words, profiling may be used when you experience performance problems in your application or when you want to understand how to optimize it.

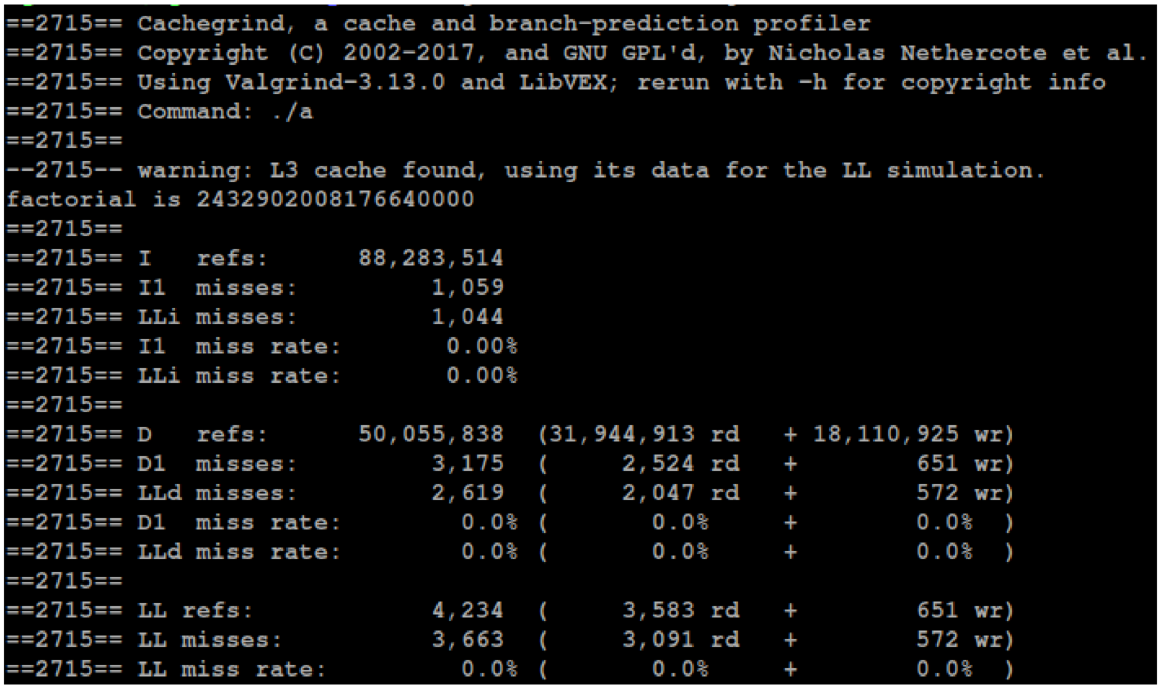
There are several manners of profiling and each one may provide different insights into the application being analyzed. How much time it spends in each part of the code, for example, is an information of particular interest. In parallel, another important information is what part of the code could be optimized.[1]

In this work we used the Valgrind tool to analyze the code "profile_me.c". The following command lines were used:

```
$ gcc -Wall profme.c -o a.o

$ valgrind --tool=cachegrind  ./a.o
```

Cachegrind simulates how the executable program interacts with the machine's cache hierarchy. It collects statistics about cache misses, simulating cache memory. The part that is specific to cachegrind is the summary of overall performance, counting instruction reads (I refs), first level instruction cache misses (I1 misses), last level cache instruction misses (LLi), data reads (D refs), first level data cache misses (D1 misses), last level cache data misses (LLd misses), and finally a summary of the last level cache accesses. The figure 1 shows those results.



```
==2715== Cachegrind, a cache and branch-prediction profiler
==2715== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==2715== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2715== Command: ./a
==2715==
--2715-- warning: L3 cache found, using its data for the LL simulation.
factorial is 2432902008176640000
==2715==
==2715== I   refs:      88,283,514
==2715== I1  misses:         1,059
==2715== LLi misses:         1,044
==2715== I1  miss rate:       0.00%
==2715== LLi miss rate:       0.00%
==2715==
==2715== D   refs:      50,055,838  (31,944,913 rd   + 18,110,925 wr)
==2715== D1  misses:         3,175  (     2,524 rd   +        651 wr)
==2715== LLd misses:         2,619  (     2,047 rd   +        572 wr)
==2715== D1  miss rate:        0.0% (       0.0%     +        0.0%  )
==2715== LLd miss rate:        0.0% (       0.0%     +        0.0%  )
==2715==
==2715== LL refs:            4,234  (     3,583 rd   +        651 wr)
==2715== LL misses:          3,663  (     3,091 rd   +        572 wr)
==2715== LL miss rate:         0.0% (       0.0%     +        0.0%  )
```

Figure 1 – Results for the Valgrind cachegrind option

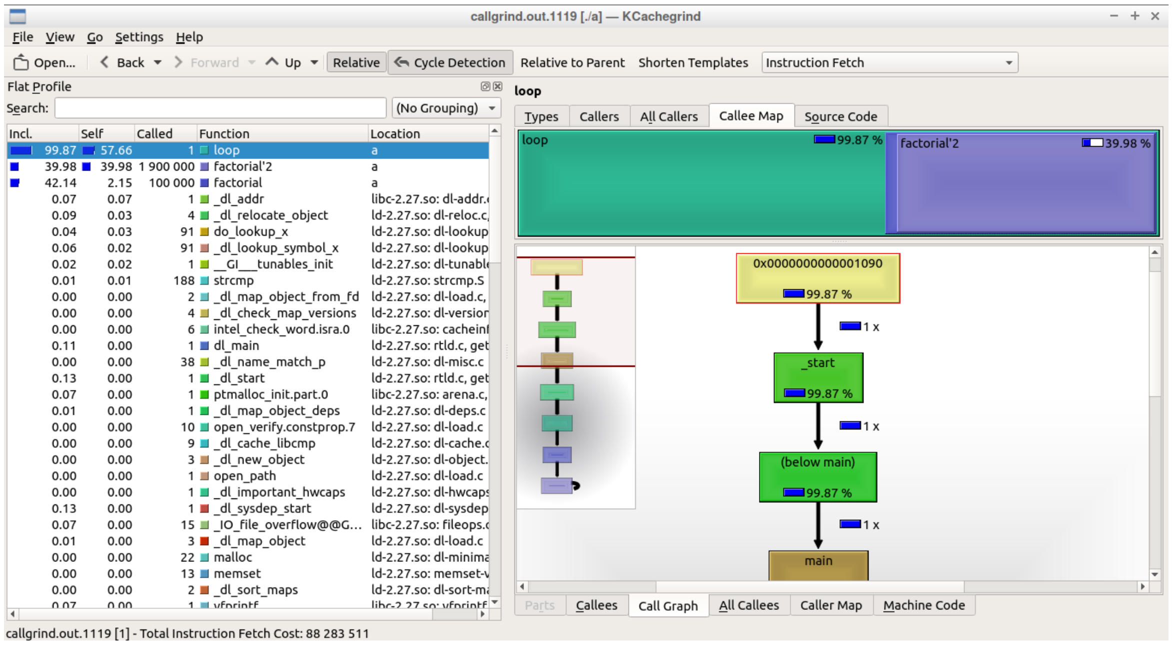
[1] Optimizing HPC Applications with Intel Cluster Tools, Paperback – October 15, 2014 by Alexander Supalov, Andrey Semin, Michael Klemm, ISBN-13: 978-1430264965 ISBN-10: 1430264969 Edition: 1st.

Exercise 02

Another option to understand the profiling data is to use the callgrind command. It is possible to access a different set of information about the code:

```
$ valgrind --tool=callgrind  ./a.o

$ kcachegrind callgrind.out.1119
```


callgrind.out.1119 [./a] — KCachegrind
File View Go Settings Help
Open... Back Forward Up Relative Cycle Detection Relative to Parent Shorten Templates Instruction Fetch
Flat Profile
Search: (No Grouping)
loop
Types Callers All Callers Callee Map Source Code
loop 99.87 %
factorial'2 39.98 %
0x0000000000001090 99.87 %
1 x
_start 99.87 %
1 x
(below main) 99.87 %
1 x
main
Parts Callees Call Graph All Callees Caller Map Machine Code

Incl.	Self	Called	Function	Location
99.87	57.66	1	loop	a
39.98	39.98	1 900 000	factorial'2	a
42.14	2.15	100 000	factorial	a
0.07	0.07	1	_dl_addr	libc-2.27.so: dl-addr.
0.09	0.03	4	_dl_relocate_object	ld-2.27.so: dl-reloc.c,
0.04	0.03	91	do_lookup_x	ld-2.27.so: dl-lookup
0.06	0.02	91	_dl_lookup_symbol_x	ld-2.27.so: dl-lookup
0.02	0.02	1	__GI___tunables_init	ld-2.27.so: dl-tunable
0.01	0.01	188	strcmp	ld-2.27.so: strcmp.S
0.00	0.00	2	_dl_map_object_from_fd	ld-2.27.so: dl-load.c,
0.00	0.00	4	_dl_check_map_versions	ld-2.27.so: dl-version
0.00	0.00	6	intel_check_word.isra.0	libc-2.27.so: cacheinf
0.11	0.00	1	dl_main	ld-2.27.so: rtld.c, get
0.00	0.00	38	_dl_name_match_p	ld-2.27.so: dl-misc.c
0.13	0.00	1	_dl_start	ld-2.27.so: rtld.c, get
0.07	0.00	1	ptmalloc_init.part.0	libc-2.27.so: arena.c,
0.01	0.00	1	_dl_map_object_deps	ld-2.27.so: dl-deps.c
0.00	0.00	10	open_verify.constprop.7	ld-2.27.so: dl-load.c
0.00	0.00	9	_dl_cache_libcmp	ld-2.27.so: dl-cache.c
0.00	0.00	3	_dl_new_object	ld-2.27.so: dl-object.
0.00	0.00	1	open_path	ld-2.27.so: dl-load.c
0.00	0.00	1	_dl_important_hwcaps	ld-2.27.so: dl-hwcaps
0.13	0.00	1	_dl_sysdep_start	ld-2.27.so: dl-sysdep
0.07	0.00	15	_IO_file_overflow@@G…	libc-2.27.so: fileops.
0.01	0.00	3	_dl_map_object	ld-2.27.so: dl-load.c
0.00	0.00	22	malloc	ld-2.27.so: dl-minima
0.00	0.00	13	memset	ld-2.27.so: memset-v
0.00	0.00	2	_dl_sort_maps	ld-2.27.so: dl-sort-ma
0.07	0.00	1	vfprintf	libc-2.27.so: vfprintf

callgrind.out.1119 [1] - Total Instruction Fetch Cost: 88 283 511

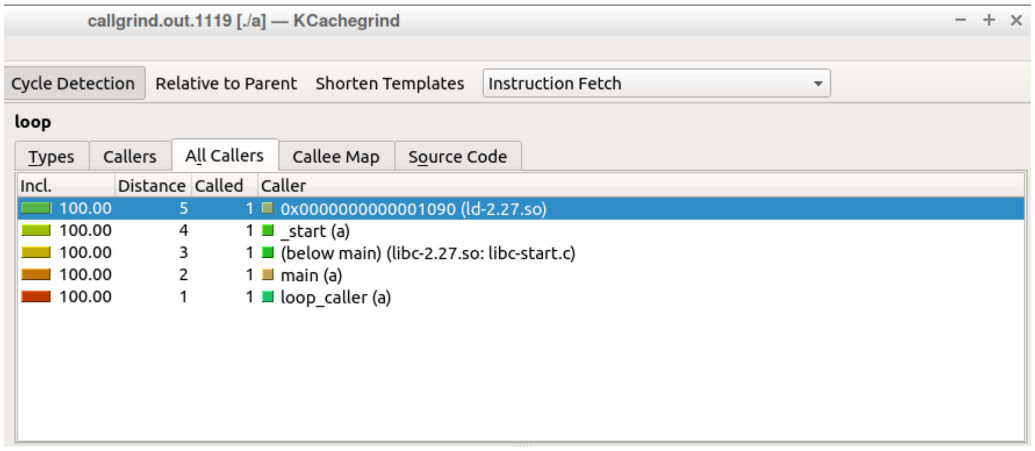
Figure 2 – Kcachegrind Call Graph Viewer.

When analyzing the information displayed by Kcachegrind (see figure 2), in the Function Profile, clicking on header of column "Self", we will get a sorted list of costs for functions, with the function with the highest cost at top. At that point it is possible to analyze the "Called" column, and see how many times determined function was called.

An additional information is a treemap view of the primary event type, up or down the call hierarchy. Each function is represented by a colored rectangle with size approximately proportional to their cost while the active function is running. As regards to the Caller Map, the graph viewer shows the hierarchy of all callers of the currently activated function; for the Callee Map, it shows that of all callees.[2] The figures 3 and 4 show the All Callers for the executable profme and the callgraph, respectively.

[2] The KCachegrind Handbook - https://docs.kde.org/trunk5/en/kdesdk/kcachegrind/kcachegrind.pdf

Exercise 02



Incl.	Distance	Called	Caller
100.00	5	1	0x0000000000001090 (ld-2.27.so)
100.00	4	1	_start (a)
100.00	3	1	(below main) (libc-2.27.so: libc-start.c)
100.00	2	1	main (a)
100.00	1	1	loop_caller (a)

Figure 3 – All callers for the profme executable.

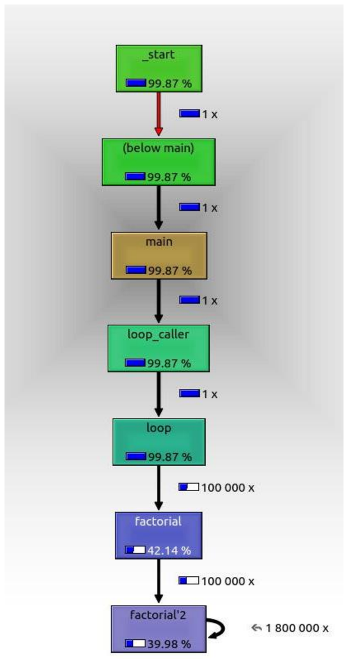
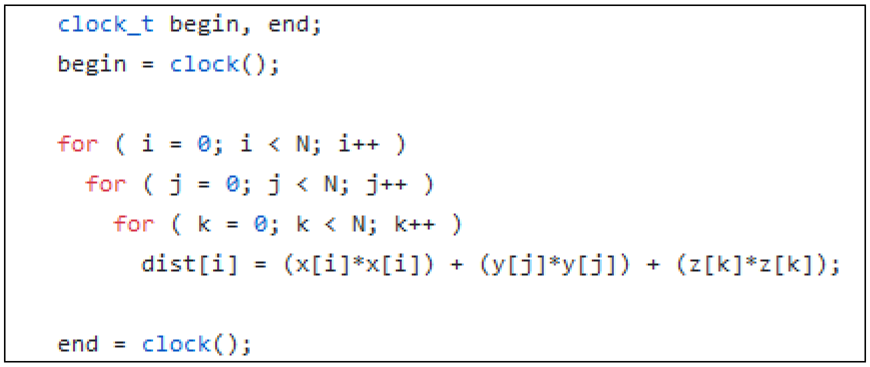


Figure 4 – Call Graph for the profme executable.

Analysis of the Time

The source code "different_clocks.c" was also considered in this work. This code measures the time taken by a for-loop, and executed a number of times (N) defined by the user. Inside de code three different timing functions are considered: clock() getrusage() clock_gettime(). The figures 5, 6, and 7 show a piece of each timing function implemented inside the code.

Exercise 02

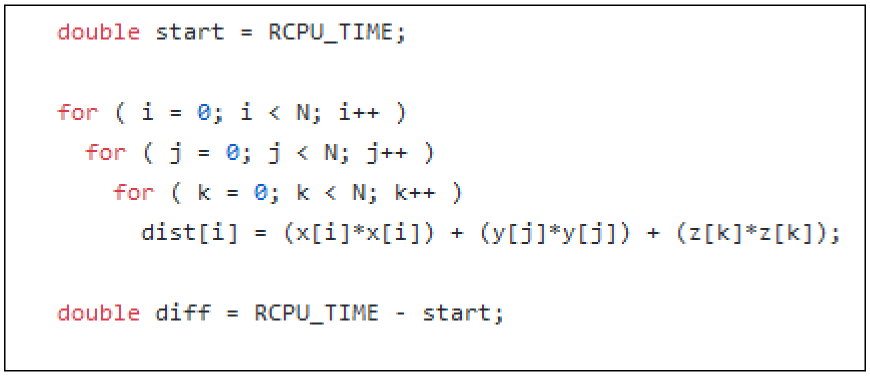


```
clock_t begin, end;
begin = clock();

for ( i = 0; i < N; i++ )
  for ( j = 0; j < N; j++ )
    for ( k = 0; k < N; k++ )
      dist[i] = (x[i]*x[i]) + (y[j]*y[j]) + (z[k]*z[k]);

end = clock();
```

Figure 5 – Clock measurement of the loop-for using the function clock( )

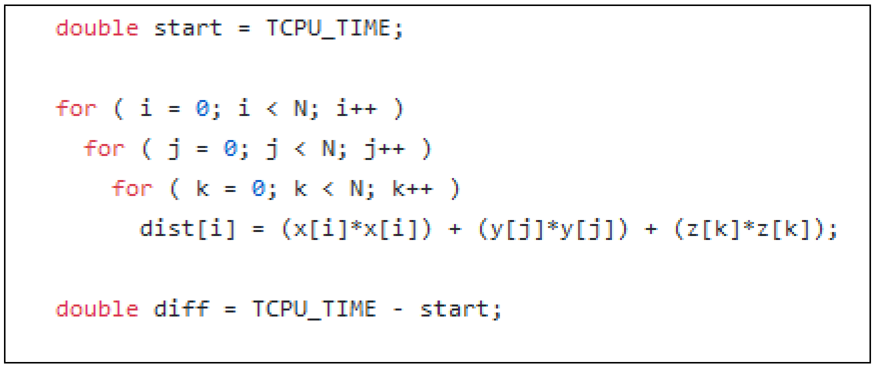


```
double start = RCPU_TIME;

for ( i = 0; i < N; i++ )
  for ( j = 0; j < N; j++ )
    for ( k = 0; k < N; k++ )
      dist[i] = (x[i]*x[i]) + (y[j]*y[j]) + (z[k]*z[k]);

double diff = RCPU_TIME - start;
```

Figure 6 – Resource usage measurement of the loop-for using the function getrusage( )



```
double start = TCPU_TIME;

for ( i = 0; i < N; i++ )
  for ( j = 0; j < N; j++ )
    for ( k = 0; k < N; k++ )
      dist[i] = (x[i]*x[i]) + (y[j]*y[j]) + (z[k]*z[k]);

double diff = TCPU_TIME - start;
```

Figure 7 – Retrieve the time of a specific clock using the function clock_gettime( )

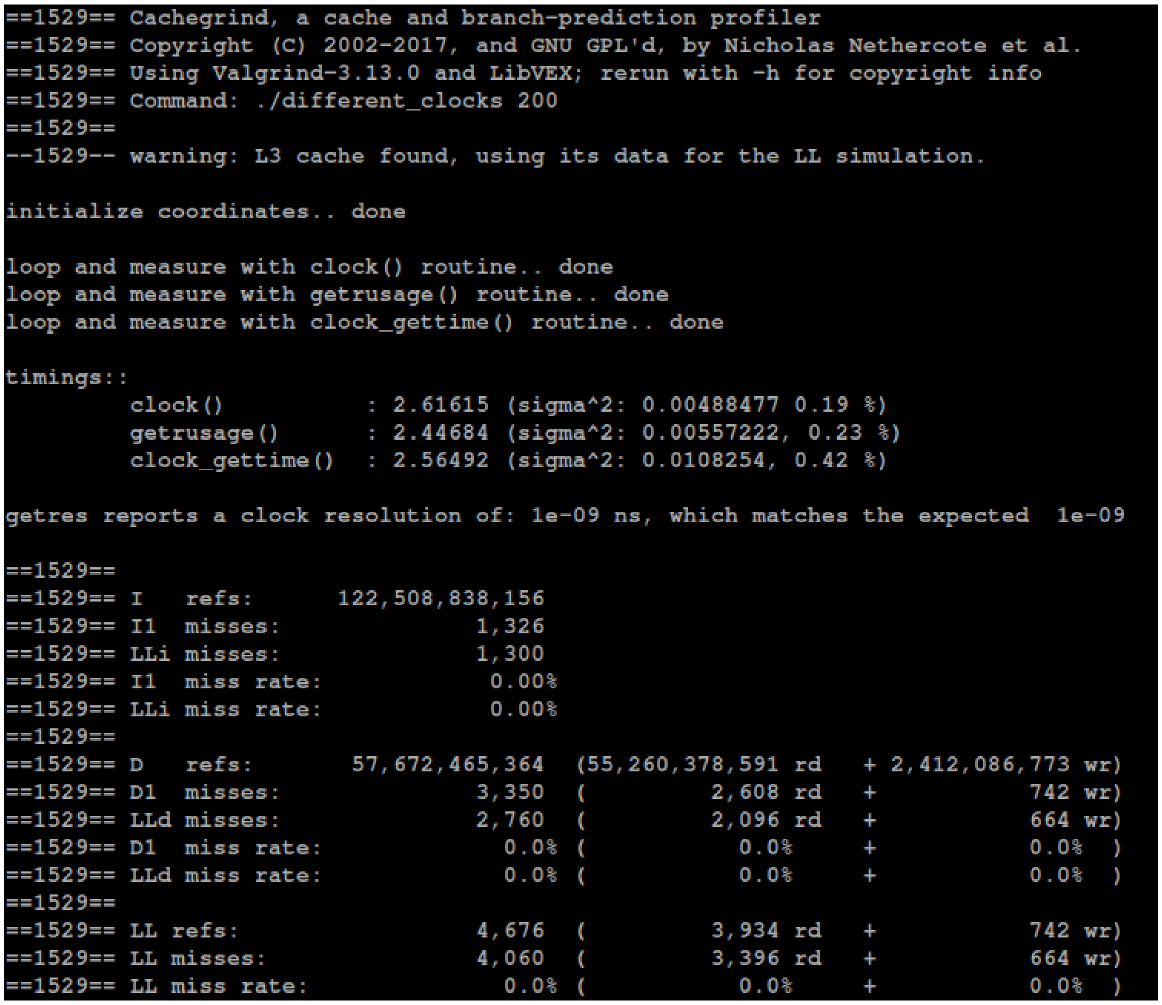
The following commands were used to compile the source code and to execute the valgrind cachegrind option:

```
$ gcc -Wall different_clocks.c -o different_clocks

$ valgrind --tool=cachegrind  ./different_clocks 200
```

Exercise 02

The output of the valgrind cachegrind is showed in the figure 8.



```
==1529== Cachegrind, a cache and branch-prediction profiler
==1529== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==1529== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1529== Command: ./different_clocks 200
==1529==
--1529-- warning: L3 cache found, using its data for the LL simulation.

initialize coordinates.. done

loop and measure with clock() routine.. done
loop and measure with getrusage() routine.. done
loop and measure with clock_gettime() routine.. done

timings::
         clock()          : 2.61615 (sigma^2: 0.00488477 0.19 %)
         getrusage()      : 2.44684 (sigma^2: 0.00557222, 0.23 %)
         clock_gettime()  : 2.56492 (sigma^2: 0.0108254, 0.42 %)

getres reports a clock resolution of: 1e-09 ns, which matches the expected  1e-09

==1529==
==1529== I   refs:      122,508,838,156
==1529== I1  misses:              1,326
==1529== LLi misses:              1,300
==1529== I1  miss rate:            0.00%
==1529== LLi miss rate:            0.00%
==1529==
==1529== D   refs:       57,672,465,364  (55,260,378,591 rd   + 2,412,086,773 wr)
==1529== D1  misses:              3,350  (         2,608 rd   +           742 wr)
==1529== LLd misses:              2,760  (         2,096 rd   +           664 wr)
==1529== D1  miss rate:             0.0% (           0.0%     +           0.0%  )
==1529== LLd miss rate:             0.0% (           0.0%     +           0.0%  )
==1529==
==1529== LL refs:                 4,676  (         3,934 rd   +           742 wr)
==1529== LL misses:               4,060  (         3,396 rd   +           664 wr)
==1529== LL miss rate:              0.0% (           0.0%     +           0.0%  )
```

Figure 8 – Results for the Valgrind cachegrind option

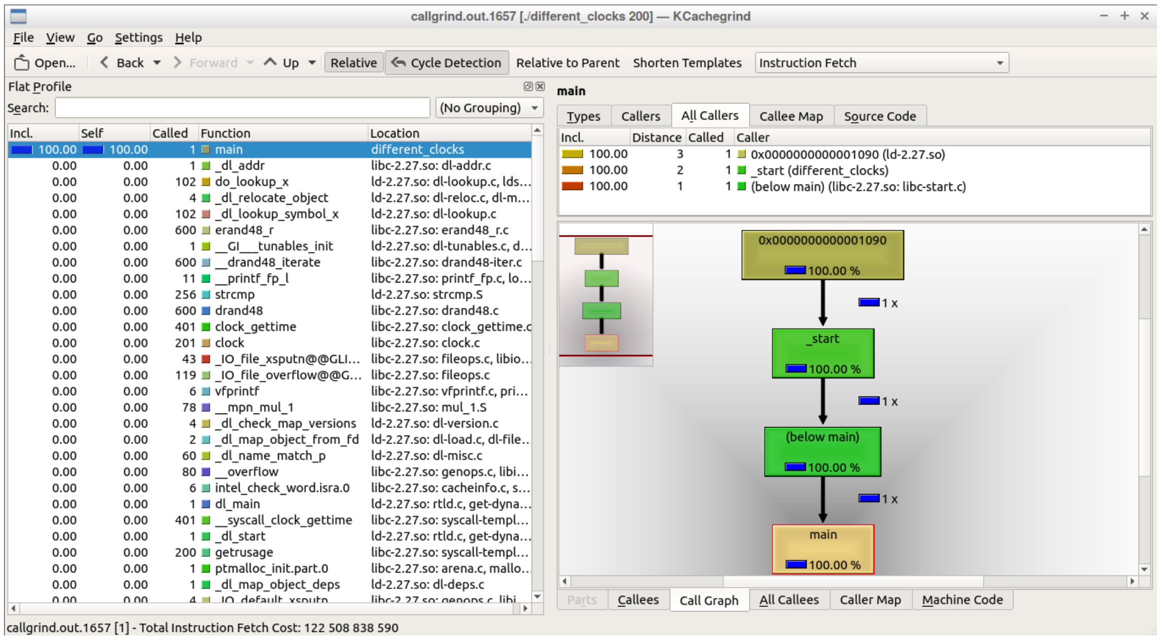
The following command lines were used:

```
$ valgrind --tool=callgrind  ./different_clocks 200

$ kcachegrind callgrind.out.1657
```

Finally, figure 9 shows the kcachegrind for the file callgrind.out.1657

Exercise 02


callgrind.out.1657 [./different_clocks 200] — KCachegrind
File View Go Settings Help
Open... Back Forward Up Relative Cycle Detection Relative to Parent Shorten Templates Instruction Fetch
Flat Profile
Search:
(No Grouping)
main
Types Callers All Callers Callee Map Source Code
0x0000000000001090
100.00 %
1 x
_start
100.00 %
1 x
(below main)
100.00 %
1 x
main
100.00 %
Parts Callees Call Graph All Callees Caller Map Machine Code

Incl.	Self	Called	Function	Location
100.00	100.00	1	main	different_clocks
0.00	0.00	1	_dl_addr	libc-2.27.so: dl-addr.c
0.00	0.00	102	do_lookup_x	ld-2.27.so: dl-lookup.c, lds…
0.00	0.00	4	_dl_relocate_object	ld-2.27.so: dl-reloc.c, dl-m…
0.00	0.00	102	_dl_lookup_symbol_x	ld-2.27.so: dl-lookup.c
0.00	0.00	600	erand48_r	libc-2.27.so: erand48_r.c
0.00	0.00	1	__GI___tunables_init	ld-2.27.so: dl-tunables.c, d…
0.00	0.00	600	__drand48_iterate	libc-2.27.so: drand48-iter.c
0.00	0.00	11	__printf_fp_l	libc-2.27.so: printf_fp.c, lo…
0.00	0.00	256	strcmp	ld-2.27.so: strcmp.S
0.00	0.00	600	drand48	libc-2.27.so: drand48.c
0.00	0.00	401	clock_gettime	libc-2.27.so: clock_gettime.c
0.00	0.00	201	clock	libc-2.27.so: clock.c
0.00	0.00	43	_IO_file_xsputn@@GLI…	libc-2.27.so: fileops.c, libio…
0.00	0.00	119	_IO_file_overflow@@G…	libc-2.27.so: fileops.c
0.00	0.00	6	vfprintf	libc-2.27.so: vfprintf.c, pri…
0.00	0.00	78	__mpn_mul_1	libc-2.27.so: mul_1.S
0.00	0.00	4	_dl_check_map_versions	ld-2.27.so: dl-version.c
0.00	0.00	2	_dl_map_object_from_fd	ld-2.27.so: dl-load.c, dl-file…
0.00	0.00	60	_dl_name_match_p	ld-2.27.so: dl-misc.c
0.00	0.00	80	__overflow	libc-2.27.so: genops.c, libi…
0.00	0.00	6	intel_check_word.isra.0	libc-2.27.so: cacheinfo.c, s…
0.00	0.00	1	dl_main	ld-2.27.so: rtld.c, get-dyna…
0.00	0.00	401	__syscall_clock_gettime	libc-2.27.so: syscall-templ…
0.00	0.00	1	_dl_start	ld-2.27.so: rtld.c, get-dyna…
0.00	0.00	200	getrusage	libc-2.27.so: syscall-templ…
0.00	0.00	1	ptmalloc_init.part.0	libc-2.27.so: arena.c, mallo…
0.00	0.00	1	_dl_map_object_deps	ld-2.27.so: dl-deps.c
0.00	0.00	4	_IO_default_xsputn	libc-2.27.so: genops.c, libi

Incl.	Distance	Called	Caller
100.00	3	1	0x0000000000001090 (ld-2.27.so)
100.00	2	1	_start (different_clocks)
100.00	1	1	(below main) (libc-2.27.so: libc-start.c)

callgrind.out.1657 [1] - Total Instruction Fetch Cost: 122 508 838 590

Figure 9 - Kcachegrind Call Graph Viewer for the file different_clocks.