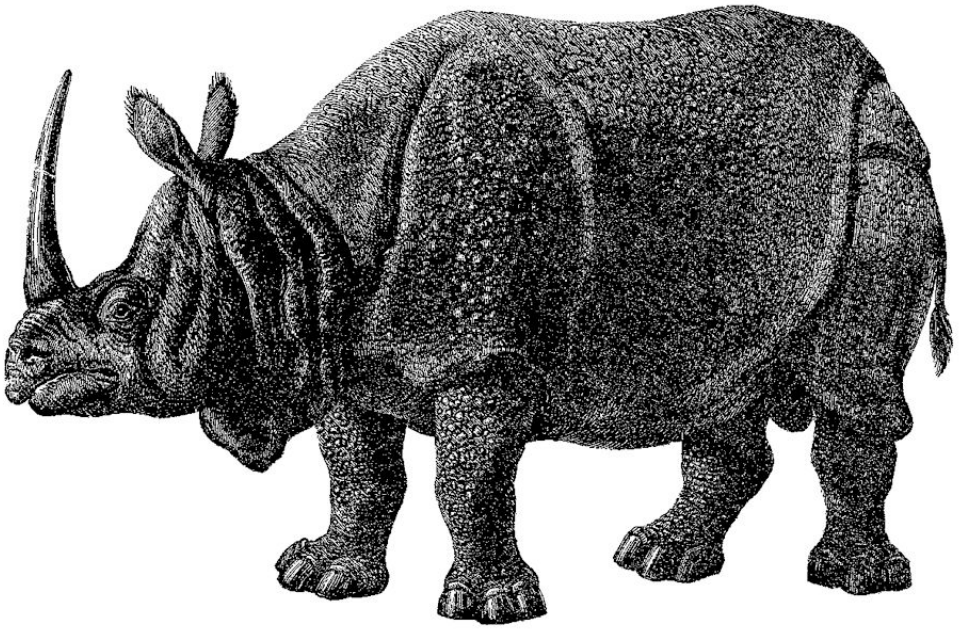
Session 3: O3 is your Lord and Savior



Why Does My Code Suck?

An Insightful Deliverable

Ricard Zarco
& Aleix Lladó

PCA' REILLY

2 Arithmetic Expression Optimizations

1. primers.c program uses the Eratostenes method in order to create a prime list from 1 upto the number you indicate as parameter).

As we told to our teacher, we will use an execution parameter of 5000000 while we do not specify otherwise.

(a) Compile primers with ”-O0” (make primers.0 and do timing, and keep the original output result).

Done.

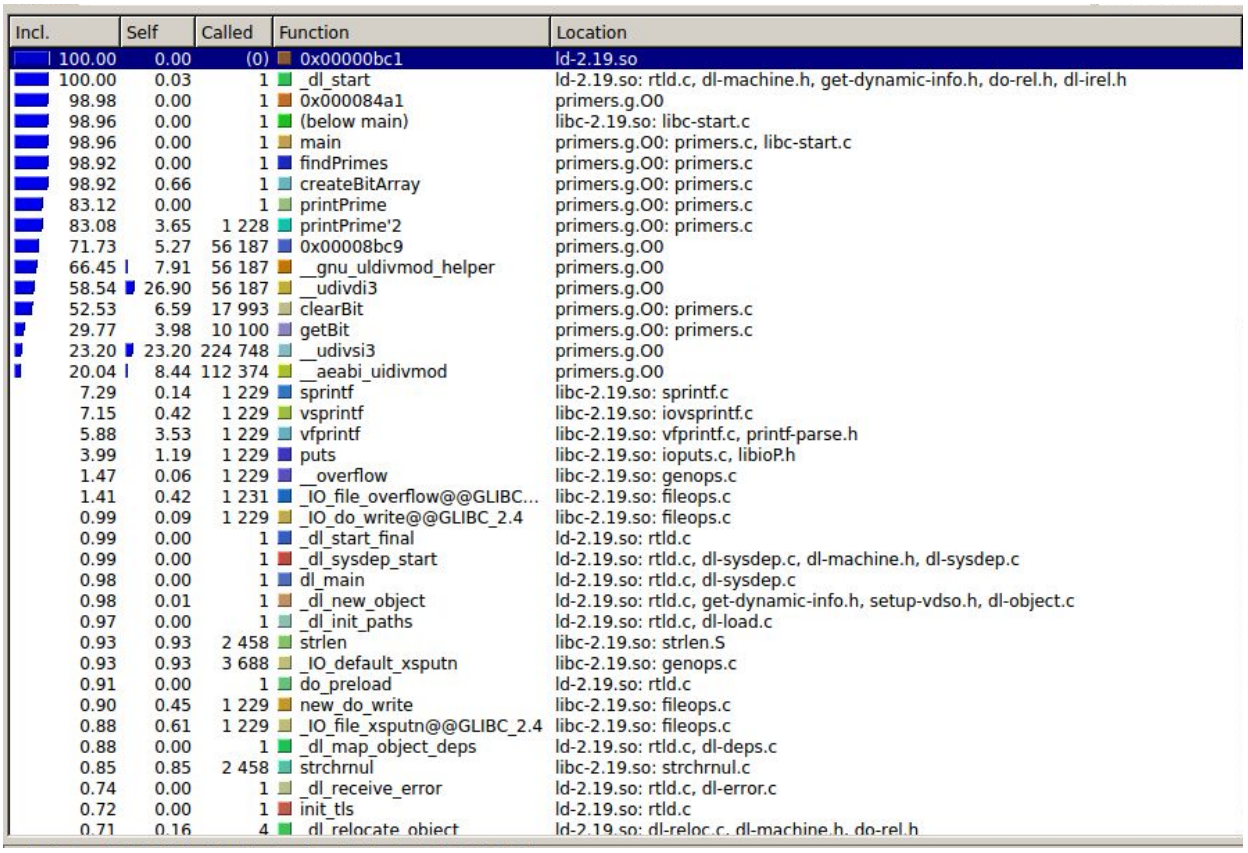
(b) Use profiling (with gprof,valgrind and/or operf, opannotate) to analyze its behavior when compiling with -O0. Note: you may want to use the primers argument to reduce the computation time with valgrind.

- Gprof profiling:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/primers$ gprof ./primers.g.pg.O0
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 22.59      1.78     1.78                             __udivdi3
 19.29      3.30     1.52                             __aeabi_uidiv
 13.32      4.35     1.05        1     1.05     2.10  findPrimes
 12.44      5.33     0.98                             __gnu_uldivmod_helper
  9.90      6.11     0.78                             __aeabi_uldivmod
  9.14      6.83     0.72                             __aeabi_uidivmod
  8.38      7.49     0.66 11528252     0.00     0.00  clearBit
  4.31      7.83     0.34  5002236     0.00     0.00  getBit
  0.63      7.88     0.05   348513     0.00     0.00  printPrime
  0.00      7.88     0.00        1     0.00     0.00  createBitArray
  0.00      7.88     0.00        1     0.00     0.00  freeBitArray
  0.00      7.88     0.00        1     0.00     0.00  setAll
```

- Valgrind: 10000 first numbers



Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x00000bc1	ld-2.19.so
100.00	0.03	1	_dl_start	ld-2.19.so: rtld.c, dl-machine.h, get-dynamic-info.h, do-rel.h, dl-irel.h
98.98	0.00	1	0x000084a1	primers.g.O0
98.96	0.00	1	(below main)	libc-2.19.so: libc-start.c
98.96	0.00	1	main	primers.g.O0: primers.c, libc-start.c
98.92	0.00	1	findPrimes	primers.g.O0: primers.c
98.92	0.66	1	createBitArray	primers.g.O0: primers.c
83.12	0.00	1	printPrime	primers.g.O0: primers.c
83.08	3.65	1 228	printPrime'2	primers.g.O0: primers.c
71.73	5.27	56 187	0x00008bc9	primers.g.O0
66.45	7.91	56 187	__gnu_uldivmod_helper	primers.g.O0
58.54	26.90	56 187	__udivdi3	primers.g.O0
52.53	6.59	17 993	clearBit	primers.g.O0: primers.c
29.77	3.98	10 100	getBit	primers.g.O0: primers.c
23.20	23.20	224 748	__udivsi3	primers.g.O0
20.04	8.44	112 374	__aeabi_uidivmod	primers.g.O0
7.29	0.14	1 229	sprintf	libc-2.19.so: sprintf.c
7.15	0.42	1 229	vsprintf	libc-2.19.so: iovsprintf.c
5.88	3.53	1 229	vfprintf	libc-2.19.so: vfprintf.c, printf-parse.h
3.99	1.19	1 229	puts	libc-2.19.so: ioputs.c, libioP.h
1.47	0.06	1 229	__overflow	libc-2.19.so: genops.c
1.41	0.42	1 231	_IO_file_overflow@@GLIBC...	libc-2.19.so: fileops.c
0.99	0.09	1 229	_IO_do_write@@GLIBC_2.4	libc-2.19.so: fileops.c
0.99	0.00	1	_dl_start_final	ld-2.19.so: rtld.c
0.99	0.00	1	_dl_sysdep_start	ld-2.19.so: rtld.c, dl-sysdep.c, dl-machine.h, dl-sysdep.c
0.98	0.00	1	dl_main	ld-2.19.so: rtld.c, dl-sysdep.c
0.98	0.01	1	_dl_new_object	ld-2.19.so: rtld.c, get-dynamic-info.h, setup-vdso.h, dl-object.c
0.97	0.00	1	_dl_init_paths	ld-2.19.so: rtld.c, dl-load.c
0.93	0.93	2 458	strlen	libc-2.19.so: strlen.S
0.93	0.93	3 688	_IO_default_xsputn	libc-2.19.so: genops.c
0.91	0.00	1	do_preload	ld-2.19.so: rtld.c
0.90	0.45	1 229	new_do_write	libc-2.19.so: fileops.c
0.88	0.61	1 229	_IO_file_xsputn@@GLIBC_2.4	libc-2.19.so: fileops.c
0.88	0.00	1	_dl_map_object_deps	ld-2.19.so: rtld.c, dl-deps.c
0.85	0.85	2 458	strchrnul	libc-2.19.so: strchrnul.c
0.74	0.00	1	_dl_receive_error	ld-2.19.so: rtld.c, dl-error.c
0.72	0.00	1	init_tls	ld-2.19.so: rtld.c
0.71	0.16	4	_dl_relocate_object	ld-2.19.so: dl-reloc.c, dl-machine.h, do-rel.h

- Opannotate: 5000000 first numbers

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/primers$
operf --event=CPU_CYCLES:100000 ./primers.g.O0 5000000

analog@pcZa:~/Laboratori/Sessio3/lab3_session/primers$ opannotate --source primers.g.O0
Using    /home/analog/Laboratori/Sessio3/lab3_session/primers/oprofile_data/samples/    for
session-dir
warning: /no-vmlinux could not be found.
/*
 * Command line: opannotate --source primers.g.O0
 *
 * Interpretation of command line:
 * Output annotated source file with samples
 * Output all files
 *
 * CPU: ARM Cortex-A9, speed 1998 MHz (estimated)
  * Counted CPU_CYCLES events (CPU cycle) with a unit mask of 0x00 (No unit mask) count
100000
 */
/*
            *            Total            samples            for            file            :
"/home/analog/Laboratori/Sessio3/lab3_session/primers/primers.c"
```

```
*
*  12291 10.5323
*/


              :#include <stdio.h>
              :#include <string.h>
              :#include <stdlib.h>
              :#include <assert.h>
              :
              :typedef unsigned long long bignum;
              :
              :typedef struct {
              :    unsigned int *p;        /* pointer to array */
              :    bignum bitsPerByte;     /* 8 on normal systems */
              :    bignum bytesPerInt;     /* sizeof(unsigned int) */
              :    bignum bitsPerInt;      /* for bit arithmetic */
              :    bignum bitsInArray;     /* how many bits in array */
              :    bignum intsInArray;     /* how many uints to give necessary bits */
              :} BITARRAY;
              :
              :void freeBitArray(BITARRAY * ba)
              :{
              :/*    free(ba->p);
              :    free(ba); */
              :}
              :
              :BITARRAY *createBitArray(bignum tits)
              :{
              :    BITARRAY *ba = malloc(sizeof(BITARRAY));
              :    assert(ba != NULL);
              :    ba->bitsPerByte = 8;
              :    ba->bytesPerInt = sizeof(unsigned int);
              :    ba->bitsPerInt = ba->bitsPerByte * ba->bytesPerInt;
              :    ba->bytesPerInt = sizeof(unsigned int);
              :    ba->bitsInArray = bits;
              :    ba->intsInArray = bits / ba->bitsPerInt + 1;
              :    ba->p = malloc(ba->intsInArray * ba->bytesPerInt);
              :    assert(ba->p != NULL);
              :    return ba;
              :}
              :
              :void setBit(BITARRAY * ba, bignum bitSS)
              :{
              :    unsigned int *pInt = ba->p + (bitSS / ba->bitsPerInt);
              :    unsigned int remainder = (bitSS % ba->bitsPerInt);
              :    *pInt |= (1 << remainder);
              :}
              :
              :void clearBit(BITARRAY * ba, bignum bitSS)
  595  0.5099 :{ /* clearBit total:   4359  3.7353 */
 1778  1.5236 :    unsigned int *pInt = ba->p + (bitSS / ba->bitsPerInt);
  647  0.5544 :    unsigned int remainder = (bitSS % ba->bitsPerInt);
  262  0.2245 :    unsigned int mask = 1 << remainder;
```

```
 266  0.2279 :    mask = ~mask;
 453  0.3882 :    *pInt &= mask;
 358  0.3068 :}
             :
             :int getBit(BITARRAY * ba, bignum bitSS)
 290  0.2485 :{ /* getBit total:   2201  1.8861 */
 643  0.5510 :    unsigned int *pInt = ba->p + (bitSS / ba->bitsPerInt);
 262  0.2245 :    unsigned int remainder = (bitSS % ba->bitsPerInt);
 123  0.1054 :    unsigned int ret = *pInt;
 168  0.1440 :    ret &= (1 << remainder);
 357  0.3059 :    return (ret != 0);
 358  0.3068 :}
             :
             :void clearAll(BITARRAY * ba)
             :{
             :    bignum intSS;
             :    for (intSS = 0; intSS <= ba->intsInArray; intSS++) {
             :      *(ba->p + intSS) = 0;
             :    }
             :}
             :
             :void setAll(BITARRAY * ba)
             :{ /* setAll total:     37  0.0317 */
             :    bignum intSS;
  26  0.0223 :    for (intSS = 0; intSS <= ba->intsInArray; intSS++) {
  11  0.0094 :      *(ba->p + intSS) = ~0;
             :    }
             :}
             :
             :void printPrime(bignum bn)
  38  0.0326 :{ /* printPrime total:    430  0.3685 */
             :    static char buf[1000];
  25  0.0214 :    sprintf(buf, "%ull", bn);
 194  0.1662 :    buf[strlen(buf) - 2] = '\0';
   4  0.0034 :    printf("%s\n", buf);
 169  0.1448 :}
             :
             :void findPrimes(bignum topCandidate)
             :{ /* findPrimes total:   5264  4.5108 */
             :    BITARRAY *ba = createBitArray(topCandidate);
             :    assert(ba != NULL);     /* SET ALL BUT 0 AND 1 TO PRIME STATUS */
             :    setAll(ba);
             :    clearBit(ba, 0);
             :    clearBit(ba, 1);        /* MARK ALL THE NON-PRIMES */
             :    bignum thisFactor = 2;
             :    bignum lastSquare = 0;
             :    bignum thisSquare = 0;
             :    while (thisFactor * thisFactor <= topCandidate) {
             :      bignum mark = thisFactor + thisFactor;
2619  2.2443 :      while (mark <= topCandidate) {
 107  0.0917 :          clearBit(ba, mark);
1180  1.0112 :          mark += thisFactor;
             :      }                     /* PRINT THE PROVEN PRIMES SO FAR */
             :      thisSquare = thisFactor * thisFactor;
```

```
671  0.5750 :      for (; lastSquare < thisSquare; lastSquare++) {
620  0.5313 :          if (getBit(ba, lastSquare))
 49  0.0420 :              printPrime(lastSquare);
             :      }                     /* SET thisFactor TO NEXT PRIME */
             :      thisFactor++;
             :      while (getBit(ba, thisFactor) == 0)
   1 8.6e-04 :          thisFactor++;
             :      assert(thisFactor <= topCandidate);
             :    }                       /* PRINT THE REMAINING PRIMES */
  12  0.0103 :    for (; lastSquare <= topCandidate; lastSquare++) {
   5  0.0043 :      if (getBit(ba, lastSquare))
             :          printPrime(lastSquare);
             :    }
             :    freeBitArray(ba);
             :}
             :
             :void test()
             :{
             :    int ss;
             :    BITARRAY *ba = createBitArray(77);
             :    clearAll(ba);           /*setAll(ba); */
             :    setBit(ba, 0);
             :    setBit(ba, 64);
             :    setBit(ba, 10);
             :    clearBit(ba, 10);
             :    clearBit(ba, 0);
             :    setBit(ba, 64);
             :    setBit(ba, 10);
             :    setBit(ba, 0);
             :    for (ss = 0; ss < 78; ss++) {
             :      if (getBit(ba, ss) != 0) {
             :          printf("%d", ss);
             :          printf("=%d ON!!!", (getBit(ba, ss)));
             :          printf("\n");
             :      }
             :    }
             :    printf("First int is %ull.\n", *(ba->p));
             :}
             :
             :int main(int argc, char *argv[])
             :{
             :    bignum topCandidate;
             :    if (argc > 1)
             :      topCandidate = atoll(argv[1]);      /*       test();  */
             :    else topCandidate = 10000000;
             :    findPrimes(topCandidate);
             :    return 0;
             :}
```

(c) Which are the most time consuming functions? Look at their most time consuming lines to figure out which are the expensive operations. Is there any way to reduce/avoid those costy operations?

GetBit, *clearBit* and *setBit* have pretty heavy times, because there are divisions and modulus that aren't handled natively by the processor. We can reduce the impact of those operations by changing them for equivalent operations of less overall latency.

(d) Copy the original code to primers_opt.c. Optimize this copy of primers based on the analysis done in the previous question.

Our optimizations are mainly focused on the division and modulus operations used in some of the most called functions used in the code (mainly the ones marked above). We have changed them for equivalent operations of less latency.

We have chosen this approach because we can see in gprof that a lot of the total time is spent in variants of the function *__uidiv3*, which are used to compute division and modulus, operations not natively supported via hardware.

You can check it out inside the file *primers_opt.c*.

(e) Compile it using make primers opt.g and check that the results of the optimized version you have done are the same than the original one.

We have created two separated outputs, one for the original version of the code and another one for the optimized one (both with execution parameters of 5000000).

Using diff, we have seen that no differences were found between the two.

(f) Which speedup have you obtained with your optimized code compared to the original primers when compiling both of them with "-O0"?

Using */usr/bin/time* (and redirecting the output to a file so we don't waste lots of time spent printing on the screen), we have obtained these times:

- Original:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/primers$ /usr/bin/time ./primers.O0 5000000 > auxOut5000000.out

7.83user 0.02system 0:07.85elapsed 99%CPU (0avgtext+0avgdata 1476maxresident)k
0inputs+0outputs (0major+212minor)pagefaults 0swaps
```

- Optimized:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/primers$ /usr/bin/time ./primers_opt.O0 5000000 > /dev/null

2.11user 0.02system 0:02.14elapsed 99%CPU (0avgtext+0avgdata 1568maxresident)k
0inputs+0outputs (0major+213minor)pagefaults 0swaps
```

Using elapsed time, we have the following speedup:

$$Speedup = \frac{OldTime}{NewTime} = \frac{7.85}{2.14} = 3.67$$

(g) Now, compile the original and the optimized versions of primers with ”-O3” and do timing of both of them.

- Original:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/primers$ /usr/bin/time ./primers.O3 5000000 > /dev/null
0.93user 0.00system 0:00.93elapsed 99%CPU (0avgtext+0avgdata 1432maxresident)k
0inputs+0outputs (0major+211minor)pagefaults 0swaps
```

- Optimized:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/primers$    /usr/bin/time    ./primers_opt.O3 5000000 > /dev/null
0.96user 0.00system 0:00.96elapsed 99%CPU (0avgtext+0avgdata 1476maxresident)k
0inputs+0outputs (0major+214minor)pagefaults 0swaps
```

(h) Which speedup have you obtained compared to the original primers when compiling with ”-O3” both programs, original and optimized? Do profiling again and look a the output of objdump -d in order to explain the speedup obtained.

- Speedup between O0 and O3 versions of the original code (note that this speedup is also the approximate value of the speedup between the original O0 code and the optimized O3 version):

$$Speedup = \frac{OldTime}{NewTime} = \frac{7.85}{0.93} = 8.44$$

- Speedup between O0 and O3 versions of the optimized code:

$$Speedup = \frac{OldTime}{NewTime} = \frac{2.14}{0.96} = 2.23$$

We will first do profiling with gprof of the O3 version of the original code:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/primers$ gprof -l ./primers.g.pg.O3
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 20.02      0.08     0.08                             clearBit (primers.c:51 @ 8a02)
 11.26      0.13     0.05                             clearBit (primers.c:49 @ 89f4)
 10.01      0.17     0.04                             findPrimes (primers.c:99 @ 89fe)
  7.51      0.20     0.03                             clearBit (primers.c:47 @ 89e4)
  7.51      0.23     0.03                             findPrimes (primers.c:104 @ 8a14)
  6.26      0.25     0.03                             clearBit (primers.c:48 @ 89e8)
  6.26      0.28     0.03                             findPrimes (primers.c:101 @ 89f2)
  5.00      0.30     0.02                             findPrimes (primers.c:99 @ 89d0)
```

```
5.00      0.32     0.02                    findPrimes (primers.c:99 @ 89f8)
3.75      0.33     0.02                    findPrimes (primers.c:101 @ 89ec)
2.50      0.34     0.01                    findPrimes (primers.c:105 @ 8a44)
2.50      0.35     0.01                    findPrimes (primers.c:104 @ 8a8a)
2.50      0.36     0.01                    getBit (primers.c:57 @ 8a34)
2.50      0.37     0.01                    getBit (primers.c:58 @ 8a40)
2.50      0.38     0.01                    printPrime (primers.c:83 @ 8a66)
1.25      0.39     0.01                    getBit (primers.c:56 @ 8a32)
1.25      0.39     0.01                    getBit (primers.c:56 @ 8a38)
1.25      0.40     0.01                    getBit (primers.c:56 @ 8af8)
1.25      0.40     0.01                    getBit (primers.c:59 @ 8afc)
```

We can already see that the most notable difference is that there isn't a single call to the division and modulus functions that we had before, as they had a sizable impact on the performance of the execution.

If we use gprof without any flag, it only shows the use of the function *findPrimes()*.

We can further check this claim looking into the assembly code of the original O3 executable, where we can see that there isn't any call to the *__divdi3* functions (or similarly expensive variants) inside *getBit*, *setBit* and *clearBit.*

The *objdump* will be given inside a file in the deliverable.

3 Memoization and Buffering

2. trigon.c program performs several calls to write and the trigonometric routines sin() and cos().

(a) Study the source code of the program.

//Ayyyyy macarena

(b) Compile the program with make trigon.pg3. Profile and analyze the program behavior. Explain the results.

We will first try to time it with /usr/bin/time, with no parameters. It prints metric tons of garbage on the screen (it's actually trying to directly print *double* data types), but once it's finished, we have this result:

```
1.37user 11.50system 0:14.90elapsed 86%CPU (0avgtext+0avgdata 1208maxresident)k
0inputs+0outputs (0major+67minor)pagefaults 0swaps
```

We can already see that the time spent in system functions is pretty high, with a CPU usage way below 98%. This is due to the two calls to *write()* that we have inside the code, located inside of a nested loop. The problem is intensified by the fact that it also prints directly on the screen, causing lots of I/O and CPU "dead" time.

We can mitigate the problem redirecting the output to */dev/null*, as it is shown here:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ /usr/bin/time ./trigon.pg3 >
/dev/null
1.31user 1.09system 0:02.41elapsed 99%CPU (0avgtext+0avgdata 1084maxresident)k
0inputs+0outputs (0major+66minor)pagefaults 0swaps
```

As we can see, the elapsed time is greatly reduced by not having to print to the screen, pointing out that the biggest factor to have in mind when trying to optimize this code is the I/O usage.

Let's further analyze the program with gprof (we are profiling the version with no output redirection):

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ gprof -l ./trigon.pg3
Flat profile:

Each sample counts as 0.01 seconds.
no time accumulated
```

We have no profiling information with gprof because the real heavy-lifting is done via dynamically linked libraries. Gprof can't get information out of those libraries, but it can be done using static compilation. We'll see it in the next section.

(c) Re-compile the program with make trigon.pg3s. Profile and analyze the program behavior. Explain the results.

- Timing with no output redirection:

```
1.59user 11.53system 0:15.09elapsed 86%CPU (0avgtext+0avgdata 460maxresident)k
0inputs+0outputs (0major+120minor)pagefaults 0swaps
```

- With output redirection:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ /usr/bin/time ./trigon.pg3s >
/dev/null
1.28user 1.16system 0:02.44elapsed 99%CPU (0avgtext+0avgdata 484maxresident)k
0inputs+0outputs (0major+122minor)pagefaults 0swaps
```

We can't find any significative difference between both versions of the executable when looking at the execution times. Let's check gprof profiling (we're checking a non-redirected output version):

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ gprof ./trigon.pg3s
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 92.26     12.63    12.63                             write
  2.19     12.93     0.30                             cos
  1.64     13.16     0.23                             sinl
  1.02     13.30     0.14                             feraiseexcept
  0.69     13.39     0.10                             sincos
  0.47     13.46     0.07                             fesetenv
  0.44     13.52     0.06                             bsloww2
  0.33     13.56     0.05                             feupdateenv
  0.29     13.60     0.04                             csloww
  0.18     13.63     0.03                             fesetround
  0.11     13.64     0.02                             __dvd
  0.11     13.66     0.02                             csloww1
  0.11     13.67     0.02                             feholdexcept
  0.07     13.68     0.01                             __write_nocancel
  0.04     13.69     0.01                             __mpcos
  0.04     13.69     0.01                             frame_dummy
  0.00     13.69     0.00        1     0.00     0.00  main
```

Now we're cooking. Having an executable compiled with -static lets us make a profiling of the libraries used that we couldn't see before. As we have theorized before, we can see that almost all the work was done by external libraries (especially that *write()* that writes into the standard output).

Let's see the redirected output (to */dev/null*) version to further prove this point:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ gprof ./trigon.pg3s
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 66.03      1.55     1.55                             write
  7.48      1.72     0.18                             cos
  7.48      1.90     0.18                             sinl
  4.27      2.00     0.10                             bsloww2
  3.42      2.08     0.08                             csloww
  3.42      2.16     0.08                             feraiseexcept
  2.78      2.22     0.07                             fesetenv
  1.50      2.26     0.04                             sincos
  1.28      2.29     0.03                             fesetround
  0.64      2.30     0.02                             __dvd
  0.43      2.31     0.01                             csloww1
  0.43      2.32     0.01                             feholdexcept
  0.43      2.33     0.01                             feupdateenv
  0.21      2.34     0.01                             __write_nocancel
  0.21      2.34     0.01                             frame_dummy
  0.00      2.34     0.00        1     0.00     0.00  main
```

At this point it's safe to assume that I/O is the main culprit of the horrible times we got for the execution of this program.

(d) Figure out how many system calls are done in the program, and which is the elapsed time of the program. Note that strace may take a while to find out this information.

We have already seen the elapsed time of the program in multiple configurations in the previous sections of this deliverable. Check those out.

As for the number of system calls done, we will use strace. Here's the output:

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 99.68    0.500795           0   2004042      4041 write
  0.32    0.001586           0      9951           rt_sigreturn
  0.00    0.000000           0         6           read
  0.00    0.000000           0        52        48 open
  0.00    0.000000           0         4           close
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         4           lseek
  0.00    0.000000           0         4         4 access
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         2           setitimer
  0.00    0.000000           0         1           uname
  0.00    0.000000           0         6           mprotect
```

```
  0.00    0.000000           0         1           writev
  0.00    0.000000           0         2           rt_sigaction
  0.00    0.000000           0         9           mmap2
  0.00    0.000000           0        16        16 stat64
  0.00    0.000000           0         3           fstat64
  0.00    0.000000           0         1           set_tls
------ ----------- ----------- --------- --------- ----------------
100.00    0.502381              2014109      4109 total
```

We can see that we have 2 million system calls to *write*, which does lots of I/O operations.

(e) Based on the previous analysis, modify the program in order to avoid so many calls and reduce the overall elapsed time.

The main focus of the optimized code is avoiding *write* calls, so we store all the outputs of an iteration of the outer loop and we print them all at once, getting a drastic reduction in the number of system calls done while also reducing the execution time.

We go from *2*n*PUNTS* calls to *n* calls to *write*. The results are obvious when we check how many system calls are done (*strace -c ./trigon.pg3s*).

```
% time     seconds  usecs/call     calls    errors syscall
------- ----------- ----------- --------- --------- ----------------
100.00    0.044159          44      1003         2 write
  0.00    0.000000           0         1           open
  0.00    0.000000           0         1           close
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         4           brk
  0.00    0.000000           0         1           readlink
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         2           setitimer
  0.00    0.000000           0         1           uname
  0.00    0.000000           0         2           writev
  0.00    0.000000           0       169           rt_sigreturn
  0.00    0.000000           0         2           rt_sigaction
  0.00    0.000000           0         1           mmap2
  0.00    0.000000           0         1           set_tls
------ ----------- ----------- --------- --------- ----------------
100.00    0.044159                  1191         3 total
```

You can check the code in the deliverable (*trigon_opt1.c*).

(f) Compute the speed-up of your optimized version compared to the original version.

Let's first get the elapsed time of the new version (with its output not redirected):

```
0.67user 1.19system 0:02.85elapsed 65%CPU (0avgtext+0avgdata 476maxresident)k
0inputs+0outputs (0major+124minor)pagefaults 0swaps
```

Let's see the speedup (no output redirection):

$$Speedup = \frac{OldTime}{NewTime} = \frac{15.09}{2.85} = 5.29$$

We see that the %CPU is extremely low. We tried redirecting the output to a file or */dev/null,* and we have seen that the %CPU increases drastically:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ /usr/bin/time ./trigon.pg3s >
/dev/null
0.91user 0.01system 0:00.92elapsed 100%CPU (0avgtext+0avgdata 472maxresident)k
0inputs+0outputs (0major+123minor)pagefaults 0swaps
```

This tells us that we spend a lot of time printing to the screen. There's nothing we can do about that, so the best thing we can do to avoid this is to just redirect the output.

With this, we can calculate the speedup when we redirect the output:

$$Speedup = \frac{OldTime}{NewTime} = \frac{2.44}{0.92} = 2.65$$

(g) Profile the new code. How much CPU time is devoted to trigonometric computations? Which is the maximum speedup that we can achieve if we improve the corresponding execution time?

We will profile the code with gprof two times, one with the output left untouched and another one redirecting it to a file. We will mark the portions directly related to trigonometric functions.

Note: We don't know if some of the functions there are related to trigonometric computations. If there are more, we just have to add their % to the subtraction done in the speedup calculation.

Here we have the new profiling done with gprof (with no output redirection):

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ gprof ./trigon.pg3s
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 41.92      0.70     0.70                             write
 17.96      1.00     0.30                             cos
 10.78      1.18     0.18                             sinl
  7.19      1.30     0.12                             bsloww2
  5.39      1.39     0.09                             feraiseexcept
  4.19      1.46     0.07                             csloww
  3.59      1.52     0.06                             fesetround
  3.29      1.58     0.06                             fesetenv
  1.80      1.61     0.03                             __dubsin
  1.80      1.64     0.03                             sincos
```

```
  1.50      1.66     0.03                              feholdexcept
  0.30      1.67     0.01                              __dvd
  0.30      1.67     0.01                              feupdateenv
  0.00      1.67     0.00        1     0.00     0.00  main
```

Write calls still have a good chunk of execution time. Let's see another profiling with the output redirected to a file (*./trigon.pg3s > trigon_opt1.out*):

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ gprof ./trigon.pg3s
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 24.49      0.24     0.24                             cos
 18.37      0.42     0.18                             sinl
 13.78      0.56     0.14                             fesetenv
  9.18      0.65     0.09                             write
  8.16      0.73     0.08                             csloww
  7.14      0.80     0.07                             bsloww2
  4.08      0.84     0.04                             feraiseexcept
  4.08      0.88     0.04                             sincos
  3.57      0.91     0.04                             feholdexcept
  2.04      0.93     0.02                             fesetround
  2.04      0.95     0.02                             feupdateenv
  1.02      0.96     0.01                             __dubsin
  0.51      0.97     0.01                             __dvd
  0.51      0.97     0.01                             __mpcos
  0.51      0.98     0.01                             random
  0.51      0.98     0.01                             setstate
  0.00      0.98     0.00        1     0.00     0.00  main
```

We see an improvement. We could probably improve it further by redirecting the output to */dev/null*. Let's see the theoretical maximum speedups we can achieve with both non-redirection and redirection of the output:

- No redirection:

$$MaxSpeedup = \frac{100}{100 - (17.96+10.78+1.8)} = 1.44$$

- Redirection to a file:

$$MaxSpeedup = \frac{100}{100 - (24.49 + 18.37 + 4.08)} = 1.88$$

(h) Modify the program in order to avoid the computation repetitions and check that the output of the new version is the same as the original code.

The focus of this optimization will be avoiding repeating the computation of *sin(d)* and *cos(d)* every iteration using memoization. You have the code in the deliverable (*trigon_opt2.c).*

(i) Profile the new version of the program and compare it with the profile of the original code.

As always, we profile it with non-redirected and redirected outputs.

- Not redirected:

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.00      0.70     0.70                             write
  0.00      0.70     0.00        1     0.00     0.00  main
```

- Redirected to a file:

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.00      0.12     0.12                             write
  0.00      0.12     0.00        1     0.00     0.00  main
```

- Redirected to */dev/null*:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.00      0.01     0.01                             write
  0.00      0.01     0.00        1     0.00     0.00  main
```

If you check the profiling of the original code done in previous sections for both versions, you can clearly see that we have reduced the time for trigonometric calculations so much that they don't even appear in the profiling of the optimized versions, as they are only calculated once for one iteration of the outer loop and reused for all the other iterations.

(j) Which overall speed-up have you obtained?

Let's get the times for the non-redirected and redirected outputs:

- Not redirected:

```
0.04user 0.52system 0:02.90elapsed 19%CPU (0avgtext+0avgdata 476maxresident)k
0inputs+0outputs (0major+124minor)pagefaults 0swaps
```

- Redirected:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/trigon$ /usr/bin/time ./trigon.pg3s >
/dev/null
0.04user 0.00system 0:00.04elapsed 88%CPU (0avgtext+0avgdata 468maxresident)k
0inputs+0outputs (0major+124minor)pagefaults 0swaps
```

Speedups with their corresponding counterparts of the original code:

- Not redirected:

$$Speedup = \frac{OldTime}{NewTime} = \frac{15.09}{2.9} = 5.20$$

- Redirected:

$$Speedup = \frac{OldTime}{NewTime} = \frac{2.44}{0.04} = 61$$

4 Routine Specialization

3. pi.c program compute the first 10000 decimals of the π number.

(a) What is the speedup of the pi program compiled with O3 compared to the pi compiled with O0?

-O0:

```
60.97user 0.27system 1:01.29elapsed 99%CPU (0avgtext+0avgdata 864maxresident)k
0inputs+0outputs (0major+67minor)pagefaults 0swaps
```

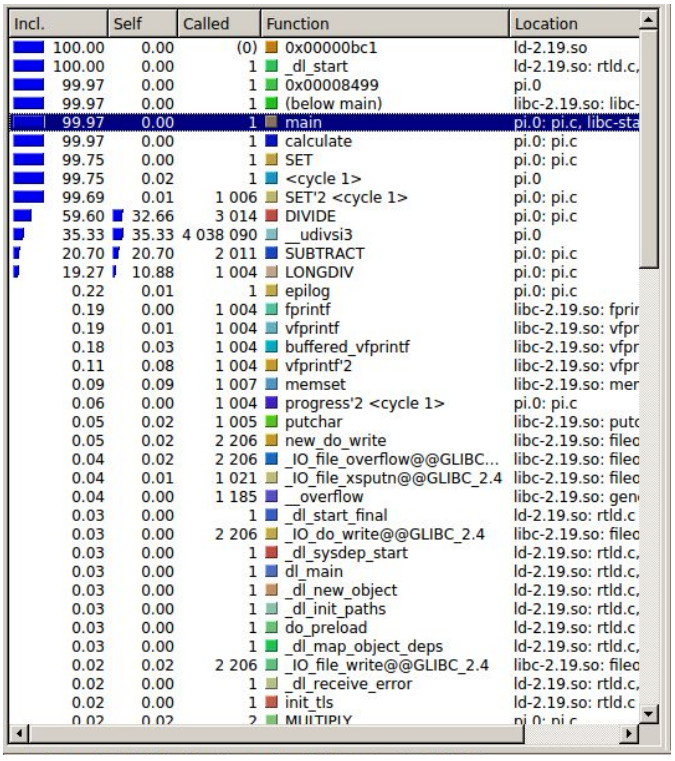
-O3:

```
18.76user 0.17system 0:18.93elapsed 99%CPU (0avgtext+0avgdata 908maxresident)k
0inputs+0outputs (0major+67minor)pagefaults 0swaps
```

$$Speedup = \frac{OldTime}{NewTime} = \frac{61.29}{18.93} = 3.24$$

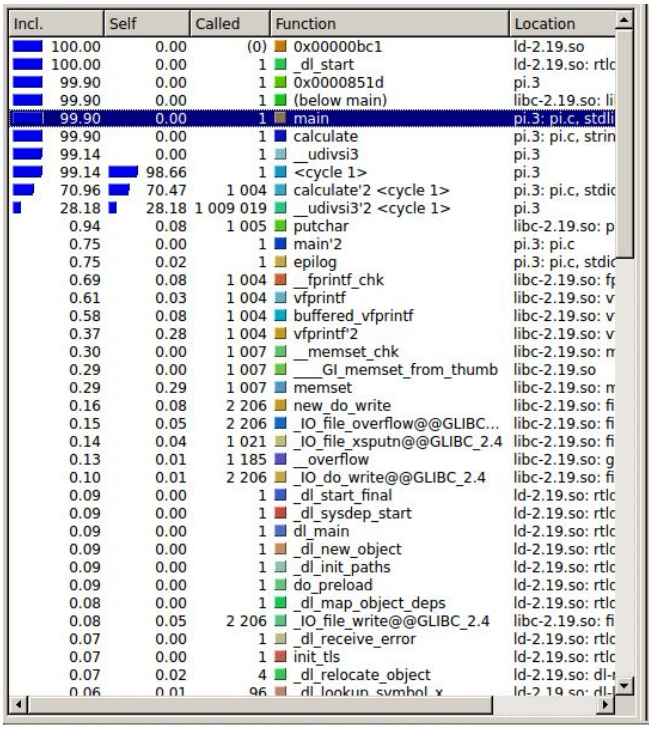
(b) Profile both pi binaries (compiled with O3 and with O0) using valgrind and callgrind annotate. Open the assembler code to understand the differences (objdump -d) and/or look for the places where ”divide instructions” are executed using callgrind annotate (Note: use 1000 as argument of pi.c when using valgrind). Could you justify these differences?

- Valgrind for O0:



Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x00000bc1	ld-2.19.so
100.00	0.00	1	_dl_start	ld-2.19.so: rtld.c,
99.97	0.00	1	0x00008499	pi.0
99.97	0.00	1	(below main)	libc-2.19.so: libc-
99.97	0.00	1	main	pi.0: pi.c, libc-sta
99.97	0.00	1	calculate	pi.0: pi.c
99.75	0.00	1	SET	pi.0: pi.c
99.75	0.02	1	<cycle 1>	pi.0
99.69	0.01	1 006	SET'2 <cycle 1>	pi.0: pi.c
59.60	32.66	3 014	DIVIDE	pi.0: pi.c
35.33	35.33	4 038 090	__udivsi3	pi.0
20.70	20.70	2 011	SUBTRACT	pi.0: pi.c
19.27	10.88	1 004	LONGDIV	pi.0: pi.c
0.22	0.01	1	epilog	pi.0: pi.c
0.19	0.00	1 004	fprintf	libc-2.19.so: fprir
0.19	0.01	1 004	vfprintf	libc-2.19.so: vfpr
0.18	0.03	1 004	buffered_vfprintf	libc-2.19.so: vfpr
0.11	0.08	1 004	vfprintf'2	libc-2.19.so: vfpr
0.09	0.09	1 007	memset	libc-2.19.so: mer
0.06	0.00	1 004	progress'2 <cycle 1>	pi.0: pi.c
0.05	0.02	1 005	putchar	libc-2.19.so: putc
0.05	0.02	2 206	new_do_write	libc-2.19.so: filec
0.04	0.02	2 206	_IO_file_overflow@@GLIBC...	libc-2.19.so: filec
0.04	0.01	1 021	_IO_file_xsputn@@GLIBC_2.4	libc-2.19.so: filec
0.04	0.00	1 185	__overflow	libc-2.19.so: gen
0.03	0.00	1	_dl_start_final	ld-2.19.so: rtld.c
0.03	0.00	2 206	_IO_do_write@@GLIBC_2.4	libc-2.19.so: filec
0.03	0.00	1	_dl_sysdep_start	ld-2.19.so: rtld.c,
0.03	0.00	1	dl_main	ld-2.19.so: rtld.c,
0.03	0.00	1	_dl_new_object	ld-2.19.so: rtld.c,
0.03	0.00	1	_dl_init_paths	ld-2.19.so: rtld.c,
0.03	0.00	1	do_preload	ld-2.19.so: rtld.c
0.03	0.00	1	_dl_map_object_deps	ld-2.19.so: rtld.c,
0.02	0.02	2 206	_IO_file_write@@GLIBC_2.4	libc-2.19.so: filec
0.02	0.00	1	_dl_receive_error	ld-2.19.so: rtld.c,
0.02	0.00	1	init_tls	ld-2.19.so: rtld.c
0.02	0.02	2	MULTIPLY	pi.0: pi.c

- Valgrind for O3:



Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x00000bc1	ld-2.19.so
100.00	0.00	1	_dl_start	ld-2.19.so: rtlc
99.90	0.00	1	0x0000851d	pi.3
99.90	0.00	1	(below main)	libc-2.19.so: li
99.90	0.00	1	main	pi.3: pi.c, stdli
99.90	0.00	1	calculate	pi.3: pi.c, strin
99.14	0.00	1	__udivsi3	pi.3
99.14	98.66	1	<cycle 1>	pi.3
70.96	70.47	1 004	calculate'2 <cycle 1>	pi.3: pi.c, stdic
28.18	28.18	1 009 019	__udivsi3'2 <cycle 1>	pi.3
0.94	0.08	1 005	putchar	libc-2.19.so: p
0.75	0.00	1	main'2	pi.3: pi.c
0.75	0.02	1	epilog	pi.3: pi.c, stdic
0.69	0.08	1 004	__fprintf_chk	libc-2.19.so: fp
0.61	0.03	1 004	vfprintf	libc-2.19.so: v
0.58	0.08	1 004	buffered_vfprintf	libc-2.19.so: v
0.37	0.28	1 004	vfprintf'2	libc-2.19.so: v
0.30	0.00	1 007	__memset_chk	libc-2.19.so: m
0.29	0.00	1 007	___GI_memset_from_thumb	libc-2.19.so
0.29	0.29	1 007	memset	libc-2.19.so: m
0.16	0.08	2 206	new_do_write	libc-2.19.so: fi
0.15	0.05	2 206	_IO_file_overflow@@GLIBC...	libc-2.19.so: fi
0.14	0.04	1 021	_IO_file_xsputn@@GLIBC_2.4	libc-2.19.so: fi
0.13	0.01	1 185	__overflow	libc-2.19.so: g
0.10	0.01	2 206	_IO_do_write@@GLIBC_2.4	libc-2.19.so: fi
0.09	0.00	1	_dl_start_final	ld-2.19.so: rtlc
0.09	0.00	1	_dl_sysdep_start	ld-2.19.so: rtlc
0.09	0.00	1	dl_main	ld-2.19.so: rtlc
0.09	0.00	1	_dl_new_object	ld-2.19.so: rtlc
0.09	0.00	1	_dl_init_paths	ld-2.19.so: rtlc
0.09	0.00	1	do_preload	ld-2.19.so: rtlc
0.08	0.00	1	_dl_map_object_deps	ld-2.19.so: rtlc
0.08	0.05	2 206	_IO_file_write@@GLIBC_2.4	libc-2.19.so: fi
0.07	0.00	1	_dl_receive_error	ld-2.19.so: rtlc
0.07	0.00	1	init_tls	ld-2.19.so: rtlc
0.07	0.02	4	_dl_relocate_object	ld-2.19.so: dl-r
0.06	0.01	96	_dl_lookup_symbol_x	ld-2.19.so: dl-l

Having a look inside both *callgrind_annotate,* we can see that the version compiled with O0 has spent an estimation of 131,163,455 cycles inside *__uidivsi3* (division pseudo-instruction) located into the *DIVIDE* and *LONGDIV* functions of the code, where they compute a division.

If we check the O3 *callgrind_annotate,* we see that the *__uidivsi3* “instructions” have disappeared from the *DIVIDE* function and that there only remains 31,122,626 cycles spent on *__udivsi3'2* inside the *LONGDIV* function.

You can check the *callgrind_annotate* output of the two versions inside the deliverable (*callgrind_annotate.O0* and *callgrind_annotate.O3*).

(c) It seems that the compiler has done a good work but there is still work to do. Based on the previous profiling, look at the source code of the program, and in particular, the source code of routine DIVIDE. Then, propose an optimization that can help to reduce the cost of each specific call to DIVIDE in order to reduce/avoid long latency operations. Specialize the code of pi using memoization. Some hints:

• For each DIVIDE by a particular value, x[k], r, and u in the code will always have the same range of values (maybe different range among them)

• Those values in the range are for a limited number of input values

For this code, we used memorization and routine specialization.

We have seen that the possible values of *n* are very limited: 5, 25 and 239. Of those values, only 25 and 239 are important, as they are used in the *DIVIDE* calls located inside a loop (with a good chunk of iterations), while the call with 5 as a parameter is only called once.

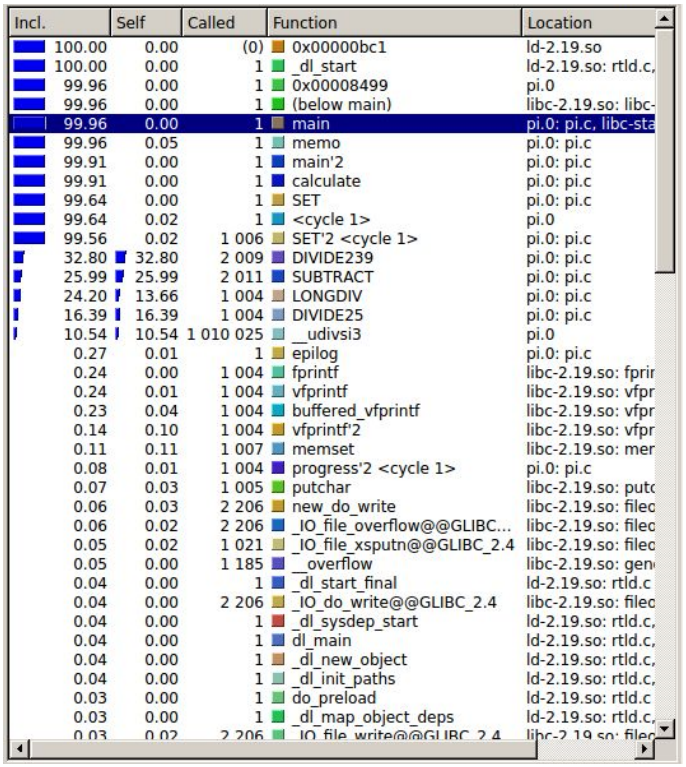
Knowing that, we created two new functions (*DIVIDE25* and *DIVIDE239)* that do not need n as a parameter, as it is already known. We also know that k[x] has a limited range of values ({0,1,...,9}). Knowing all of this, we can precompute all the possible values needed for those functions (as they largely depend on k[x] and n) and store them elsewhere, making use of the memoization technique.

You can check the code inside the deliverable (*pi_opt.c*).

(d) Profile the new version of the program and compare it with the profile of the original code.

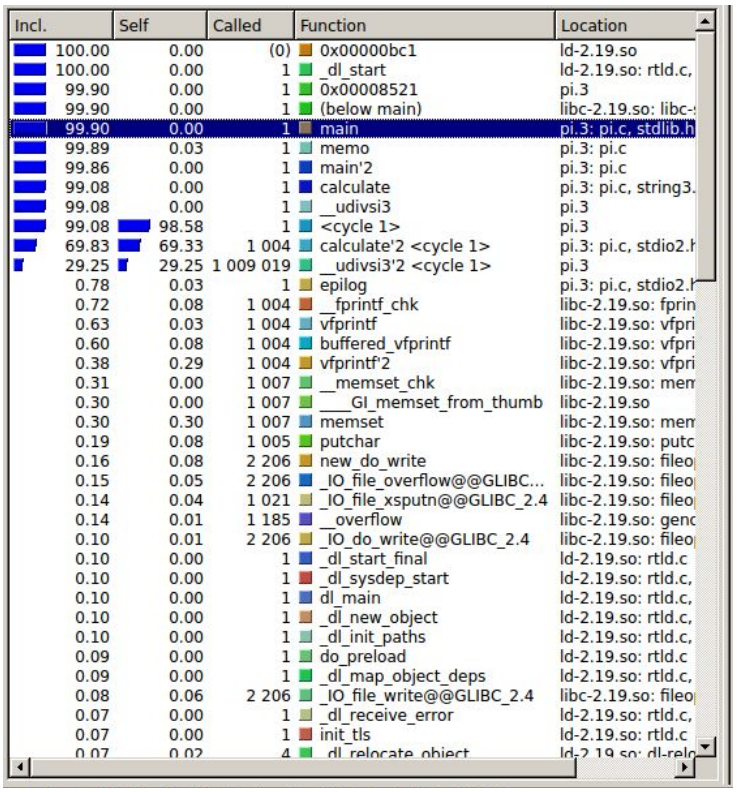
We will use Valgrind with the same input parameters as before.

- Valgrind for O0:



Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x00000bc1	ld-2.19.so
100.00	0.00	1	_dl_start	ld-2.19.so: rtld.c,
99.96	0.00	1	0x00008499	pi.0
99.96	0.00	1	(below main)	libc-2.19.so: libc-
99.96	0.00	1	main	pi.0: pi.c, libc-sta
99.96	0.05	1	memo	pi.0: pi.c
99.91	0.00	1	main'2	pi.0: pi.c
99.91	0.00	1	calculate	pi.0: pi.c
99.64	0.00	1	SET	pi.0: pi.c
99.64	0.02	1	<cycle 1>	pi.0
99.56	0.02	1 006	SET'2 <cycle 1>	pi.0: pi.c
32.80	32.80	2 009	DIVIDE239	pi.0: pi.c
25.99	25.99	2 011	SUBTRACT	pi.0: pi.c
24.20	13.66	1 004	LONGDIV	pi.0: pi.c
16.39	16.39	1 004	DIVIDE25	pi.0: pi.c
10.54	10.54	1 010 025	__udivsi3	pi.0
0.27	0.01	1	epilog	pi.0: pi.c
0.24	0.00	1 004	fprintf	libc-2.19.so: fprir
0.24	0.01	1 004	vfprintf	libc-2.19.so: vfpr
0.23	0.04	1 004	buffered_vfprintf	libc-2.19.so: vfpr
0.14	0.10	1 004	vfprintf'2	libc-2.19.so: vfpr
0.11	0.11	1 007	memset	libc-2.19.so: mer
0.08	0.01	1 004	progress'2 <cycle 1>	pi.0: pi.c
0.07	0.03	1 005	putchar	libc-2.19.so: putc
0.06	0.03	2 206	new_do_write	libc-2.19.so: filec
0.06	0.02	2 206	_IO_file_overflow@@GLIBC...	libc-2.19.so: filec
0.05	0.02	1 021	_IO_file_xsputn@@GLIBC_2.4	libc-2.19.so: filec
0.05	0.00	1 185	__overflow	libc-2.19.so: gen
0.04	0.00	1	_dl_start_final	ld-2.19.so: rtld.c
0.04	0.00	2 206	_IO_do_write@@GLIBC_2.4	libc-2.19.so: filec
0.04	0.00	1	_dl_sysdep_start	ld-2.19.so: rtld.c,
0.04	0.00	1	dl_main	ld-2.19.so: rtld.c,
0.04	0.00	1	_dl_new_object	ld-2.19.so: rtld.c,
0.04	0.00	1	_dl_init_paths	ld-2.19.so: rtld.c,
0.03	0.00	1	do_preload	ld-2.19.so: rtld.c
0.03	0.00	1	_dl_map_object_deps	ld-2.19.so: rtld.c,
0.03	0.02	2 206	_IO_file_write@@GLIBC_2.4	libc-2.19.so: filec

-Valgrind for O3:



Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x00000bc1	ld-2.19.so
100.00	0.00	1	_dl_start	ld-2.19.so: rtld.c,
99.90	0.00	1	0x00008521	pi.3
99.90	0.00	1	(below main)	libc-2.19.so: libc-
99.90	0.00	1	main	pi.3: pi.c, stdlib.h
99.89	0.03	1	memo	pi.3: pi.c
99.86	0.00	1	main'2	pi.3: pi.c
99.08	0.00	1	calculate	pi.3: pi.c, string3.
99.08	0.00	1	__udivsi3	pi.3
99.08	98.58	1	<cycle 1>	pi.3
69.83	69.33	1 004	calculate'2 <cycle 1>	pi.3: pi.c, stdio2.h
29.25	29.25	1 009 019	__udivsi3'2 <cycle 1>	pi.3
0.78	0.03	1	epilog	pi.3: pi.c, stdio2.h
0.72	0.08	1 004	__fprintf_chk	libc-2.19.so: fprin
0.63	0.03	1 004	vfprintf	libc-2.19.so: vfpri
0.60	0.08	1 004	buffered_vfprintf	libc-2.19.so: vfpri
0.38	0.29	1 004	vfprintf'2	libc-2.19.so: vfpri
0.31	0.00	1 007	__memset_chk	libc-2.19.so: mem
0.30	0.00	1 007	___GI_memset_from_thumb	libc-2.19.so
0.30	0.30	1 007	memset	libc-2.19.so: mem
0.19	0.08	1 005	putchar	libc-2.19.so: putc
0.16	0.08	2 206	new_do_write	libc-2.19.so: fileo
0.15	0.05	2 206	_IO_file_overflow@@GLIBC...	libc-2.19.so: fileo
0.14	0.04	1 021	_IO_file_xsputn@@GLIBC_2.4	libc-2.19.so: fileo
0.14	0.01	1 185	__overflow	libc-2.19.so: genc
0.10	0.01	2 206	_IO_do_write@@GLIBC_2.4	libc-2.19.so: fileo
0.10	0.00	1	_dl_start_final	ld-2.19.so: rtld.c
0.10	0.00	1	_dl_sysdep_start	ld-2.19.so: rtld.c,
0.10	0.00	1	dl_main	ld-2.19.so: rtld.c,
0.10	0.00	1	_dl_new_object	ld-2.19.so: rtld.c,
0.10	0.00	1	_dl_init_paths	ld-2.19.so: rtld.c,
0.09	0.00	1	do_preload	ld-2.19.so: rtld.c
0.09	0.00	1	_dl_map_object_deps	ld-2.19.so: rtld.c,
0.08	0.06	2 206	_IO_file_write@@GLIBC_2.4	libc-2.19.so: fileo
0.07	0.00	1	_dl_receive_error	ld-2.19.so: rtld.c,
0.07	0.00	1	init_tls	ld-2.19.so: rtld.c
0.07	0.02	4	_dl_relocate_object	ld-2.19.so: dl-relo

Comparing the two O0 versions, we can see that the function *DIVIDE* disappears from the profiling, giving birth to our new *DIVIDE25* and *DIVIDE239* functions. We can't even see the remaining original function because, as we said, it's only used once in all of the execution.

__uidivsi3 "instructions" have also been cut down considerably, as they are only done once and the results get stored. In the O3 version the differences are not as evident, as the compiler optimizes it already as much as it can.

We also included the *callgrind_annotate* of these new two versions inside the deliverable (*callgrind_annotate.O0_opt* and *callgrind_annotate.O3_opt*).

(e) Which speed-up have you obtained?

Let's get the times for both O0 and O3 versions:

- Optimized O0:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/pi$ /usr/bin/time ./pi.0 > /dev/null
53.09user 0.00system 0:53.13elapsed 99%CPU (0avgtext+0avgdata 804maxresident)k
0inputs+0outputs (0major+70minor)pagefaults 0swaps
```

- Optimized O3:

```
analog@pcZa:~/Laboratori/Sessio3/lab3_session/pi$ /usr/bin/time ./pi.3 > /dev/null
17.43user 0.01system 0:17.44elapsed 99%CPU (0avgtext+0avgdata 1012maxresident)k
0inputs+0outputs (0major+72minor)pagefaults 0swaps
```

Now let’s compute the speedups with the new times:

- Speedup for O0:

$$Speedup = \frac{OldTime}{NewTime} = \frac{61.29}{53.13} = 1.15$$

- Speedup for O3:

$$Speedup = \frac{OldTime}{NewTime} = \frac{18.93}{17.44} = 1.09$$