PCA-FIB
Laboratori 2
Ricard Zarco Badia
Aleix Lladó Nieto

Índex

1 Tools to Measure

1.1 Accounting Tools

1. This exercise should be done in the FIB computer. Objective: see how the I/O and the execution environment may affect the performance of the application. Compile popul.c program with gcc and answer the followings questions:

(a) Run the program and do timing with the GNU time command, redirecting the output to a file in your FIB account: under /home/... and under /dades/...
(Note: at the FIB machines, your /home/... account disk is mounted by NFS, and your /dades/... account disk is mounted by CIFS)

//home
Desktop/lab2_session> time ./popul > ~/Desktop/out.o
2.012u 0.012s 0:06.29 32.1% 0+0k 0+97664io 0pf+0w

//dades
Desktop/lab2_session> time ./popul > /home2/users/alumnes/1172424/dades/out.o
1.863u 0.075s 0:06.18 31.2% 0+0k 0+97664io 0pf+0w

(b) Repeat the experiment redirecting the output to a file located at /tmp.

//tmp
Desktop/lab2_session> time ./popul > /tmp/out.o
1.893u 0.004s 0:01.90 99.4% 0+0k 0+0io 0pf+0w

(c) Repeat the experiment redirecting the output to /dev/null.

//dev/null
Desktop/lab2_session> time ./popul > /dev/null
1.886u 0.003s 0:01.89 99.4% 0+0k 0+0io 0pf+0w

(d) Do you know why you are observing some differences? (elapsed time, %CPU, user time,...)

Els canvis observats són sobretot en el “% CPU” i en el elapsed time.
Això es degut a que la carpeta /home i /dades estan al·locades a un servidor extern i la carpeta /dev i /tmp estan al·locades al propi PC.

(e) What can you say regarding to the measures you have done once you have seen the results? Which is the best experimental setup for your future experiments?

Hem vist tant fent l'execució al home i a dades perdem molt de temps ja que no estem utilitzant la CPU durant el temps total perquè perds temps input/output.
El millor mètode és una execució local redireccionant la sortida a /tmp o a /dev.

2. Copy lab2 session.tar.gz files to the SD card. For instance, you may want to create a folder lab2 under /home/analog/ and copy the lab files there. Once you are login in the Zedboard platform, go to the directory where you copy the files and compile pi.c program with gcc and O0 optimization level. Then, run the program, do timing with the GNU time command. Finally, try to understand the obtained result: what is %CPU?. Remember to save the output result of the original pi.c program to compare its result to future optimizations of the program. Note that you may want to repeat several times the measure in order to be sure that the timing results are similar.

-O0:

real 1m1.332s
user 1m1.000s
sys 0m0.320s

analog@pcZa:~/Laboratori/Sessio2/lab2_session$ /usr/bin/time ./pi.O0 > pi.O0.out
61.18user 0.31system 1:01.51elapsed 99%CPU (0avgtext+0avgdata 828maxresident)k
0inputs+0outputs (0major+65minor)pagefaults 0swaps

3. Now, compile the pi.c program using gcc and O0, O1, O2 and O3 optimization level flags:

(a) Run and check that pi obtained with O0-3 optimization levels obtain the same result.

No hi ha diferència en els outputs dels fitxers.

-O1:

analog@pcZa:~/Laboratori/Sessio2/lab2_session$ /usr/bin/time ./pi.O1 > pi.01.out
30.36user 0.32system 0:30.69elapsed 99%CPU (0avgtext+0avgdata 928maxresident)k
0inputs+0outputs (0major+68minor)pagefaults 0swaps

-O2:

analog@pcZa:~/Laboratori/Sessio2/lab2_session$ /usr/bin/time ./pi.O2 > pi.02.out
18.91user 0.31system 0:19.23elapsed 99%CPU (0avgtext+0avgdata 900maxresident)k
0inputs+0outputs (0major+68minor)pagefaults 0swaps

-O3:

analog@pcZa:~/Laboratori/Sessio2/lab2_session$ /usr/bin/time ./pi.O3 > pi.03.out
19.01user 0.29system 0:19.31elapsed 99%CPU (0avgtext+0avgdata 796maxresident)k

0inputs+0outputs (0major+65minor)pagefaults 0swaps

(b) Compute the speed-up of user time + system time of the program compiled using O3 compared to the program compiled with O0.

Speed up (user + sys) = 61.49/19.3 = 3.186

(c) Compute the speed-up of elapsed time of the program compiled using O3 compared to the program compiled with O0.

Speed up (elapsed) = 61.51/19.23 = 3.199

1.2 Profiling tools

1.2.1 Profiling with gprof and Valgrind

4. Compile the pi.c program with gcc, O0 optimization level, gprof profiling option -pg for the gprof experiments, and the debug option (-g), with and without static link (-static). Using gprof and Valgrind answer the following questions. Note: Look at the pid number of the output of valgrind to know which is the callgrind.out.pid file that you should use in callgrind annotate.
Advise: run pi.c with 1000 argument, otherwise it will take too long.

(a) Which is the most invoked routine by the program?
(b) Which is the most CPU time consuming routine?
(c) Which is the most CPU time consuming source code line?
(d) Does it appear the system mode execution time in the gprof output? and in the Valgrind output?

GPROF O0 NO_STATIC

analog@pcZa:~/Laboratori/Sessio2/lab2_session$ gprof -b ./pi.O0.pg.g.no_static

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
45.00	0.27	0.27				__aeabi_uidiv
28.33	0.44	0.17	2011	0.08	0.08	SUBTRACT
16.67	0.54	0.10	3014	0.03	0.03	DIVIDE
10.00	0.60	0.06	1004	0.06	0.06	LONGDIV
0.00	0.60	0.00	1007	0.00	0.00	SET
0.00	0.60	0.00	1005	0.00	0.00	progress
0.00	0.60	0.00	2	0.00	0.00	MULTIPLY
0.00	0.60	0.00	1	0.00	330.00	calculate
0.00	0.60	0.00	1	0.00	0.00	epilog

a) La rutina que més s'invoca és la DIVIDE.
b) La rutina de llibreria que més CPU time gasta és __aeabi_uidiv.
La rutina d'usuari que més CPU time gasta és SUBTRACT.

c) La línia de rutina que més CPU time gasta és la següent amb un 16.67%:
16.67 0.37 0.10 SUBTRACT (pi.c:91 @ 890c)

d) No es pot saber el mode d'execució del sistema ja que les llibreries no s'han compilat amb la informació gprof.

GPROF O0 STATIC

analog@pcZa:~/Laboratori/Sessio2/lab2_session$ gprof -b ./pi.O0.pg.g.static
Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
32.84	0.22	0.22				__udivsi3
25.37	0.39	0.17	3014	0.06	0.06	DIVIDE
23.88	0.55	0.16	2011	0.08	0.08	SUBTRACT
10.45	0.62	0.07	1004	0.07	0.07	LONGDIV
5.97	0.66	0.04				write
1.49	0.67	0.01				_IO_file_write
0.00	0.67	0.00	1007	0.00	0.00	SET
0.00	0.67	0.00	1005	0.00	0.00	progress
0.00	0.67	0.00	2	0.00	0.00	MULTIPLY
0.00	0.67	0.00	1	0.00	400.00	calculate
0.00	0.67	0.00	1	0.00	0.00	epilog
0.00	0.67	0.00	1	0.00	400.00	main

a) La rutina que més s'invoca és la DIVIDE.
b) La rutina de llibreria que més CPU time gasta és __aeabi_uidiv.
La rutina d'usuari que més CPU time gasta és DIVIDE.

c) La línia de rutina que més CPU time gasta és la següent amb un 11.94%:
11.94 0.30 0.08 SUBTRACT (pi.c:91 @ 8c78)

d) No es pot saber el mode d'execució del sistema ja que les llibreries no s'han compilat amb la informació gprof.

"- compte amb la corrent estàtica, no toqueu la placa.
- què m'has dit, que si compilo amb --static no toqui la placa?"

VALGRIND O0 no_static

a) La rutina que més cops s'invoca de llibreria és __udivsi3 amb 4038090 vegades.
La rutina que d'usuari que més cops s'invoca és DIVIDE amb 3014 vegades.
b) La rutina de llibreria que més CPU time gasta és __aeabi_uidiv amb un 35.33%.
La rutina d'usuari que més CPU time gasta és DIVIDE amb un 32.66%.
c) La línia de rutina d'usuari que més CPU time gasta és DIVIDE 19.99%. (ULL)
d) Sí, però és una estimació.

VALGRIND O0 static

a) La rutina que més cops s'invoca de llibreria és __udivsi3 amb 4038092 vegades.
   La rutina que d'usuari que més cops s'invoca és DIVIDE amb 3014 vegades.
b) La rutina de llibreria que més CPU time gasta és __aeabi_uidiv amb un 35.35%.
   La rutina d'usuari que més CPU time gasta és DIVIDE amb un 32.67%.
c) La línia de rutina d'usuari que més CPU time gasta és DIVIDE 19.99%. (ULL)
d) Sí, però és una estimació.

5. Repeat the previous experiment compiling now with O3 optimization level:
(a) Which differences you can observe looking at the flat profile (significant changes on the routine weights, routines that disappear/appear,...)?
(b) Do you know why there are those differences? Hints:
• It may be useful for you to look at the assembler code generated using O0 and O3 optimization levels (Hint: look for the bl ARM assembler instruction).
• Also, observe the information given by gprof and Valgrind at source code line level.

GPROF O3 NO_STATIC

analog@pcZa:~/Laboratori/Sessio2/lab2_session$ gprof -b ./pi.O3.pg.g.no_static
Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
70.66	0.12	0.12				calculate
29.44	0.17	0.05				__aeabi_uidiv

a) Han desaparegut totes les rutines menys el calculate.
b) Degut al nivell d'optimització O3, que activa la desplegament de bucles i intercalació de funcions automàtiques.

GPROF O3 STATIC

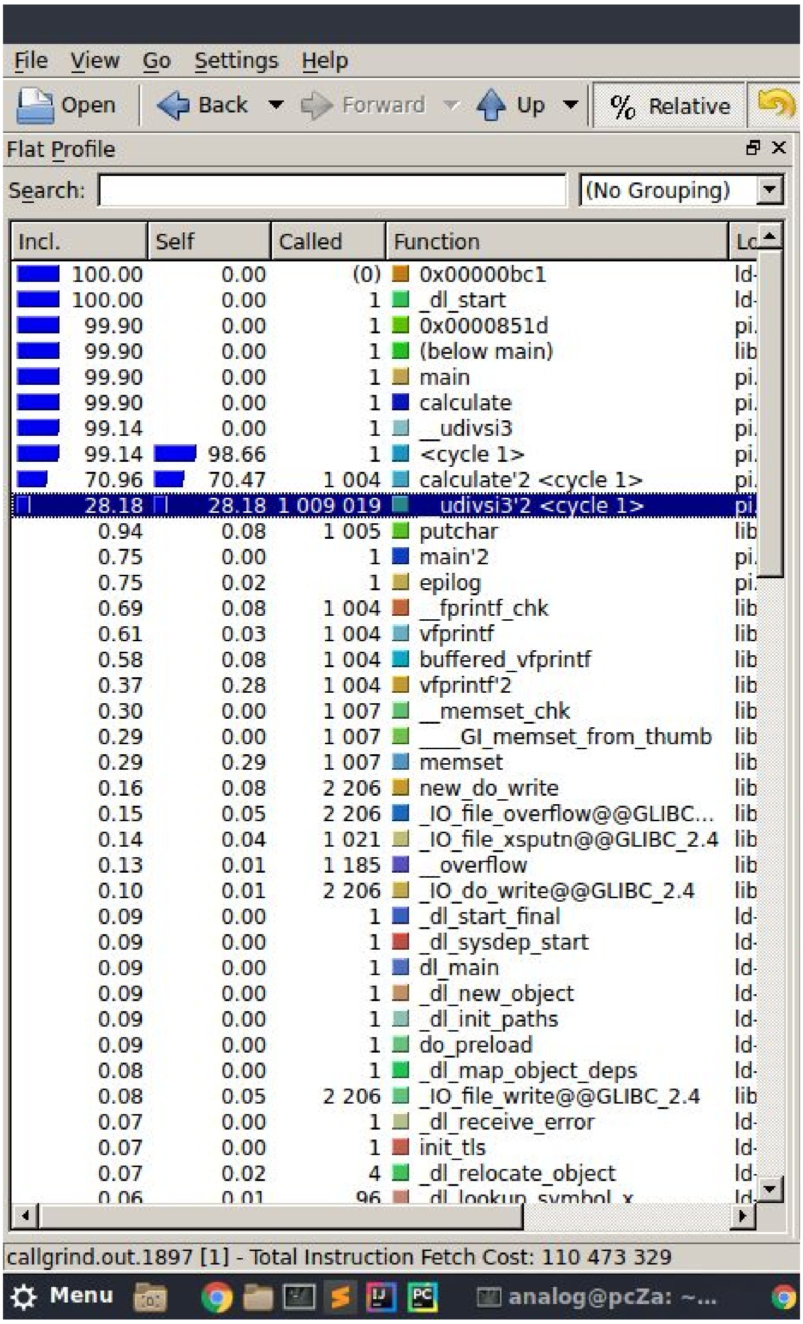
analog@pcZa:~/Laboratori/Sessio2/lab2_session$ gprof -b ./pi.O3.pg.g.static
Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
65.00	0.13	0.13				calculate
25.00	0.18	0.05				__udivsi3
10.00	0.20	0.02				write
0.00	0.20	0.00	1	0.00	0.00	main

a) Han desaparegut totes les rutines menys el calculate i ha aparegut el write i el main.
b) Degut al nivell d’optimització O3, que activa la desplegament de bucles i intercalació de funcions automàtiques.

VALGRIND O3 NO_STATIC

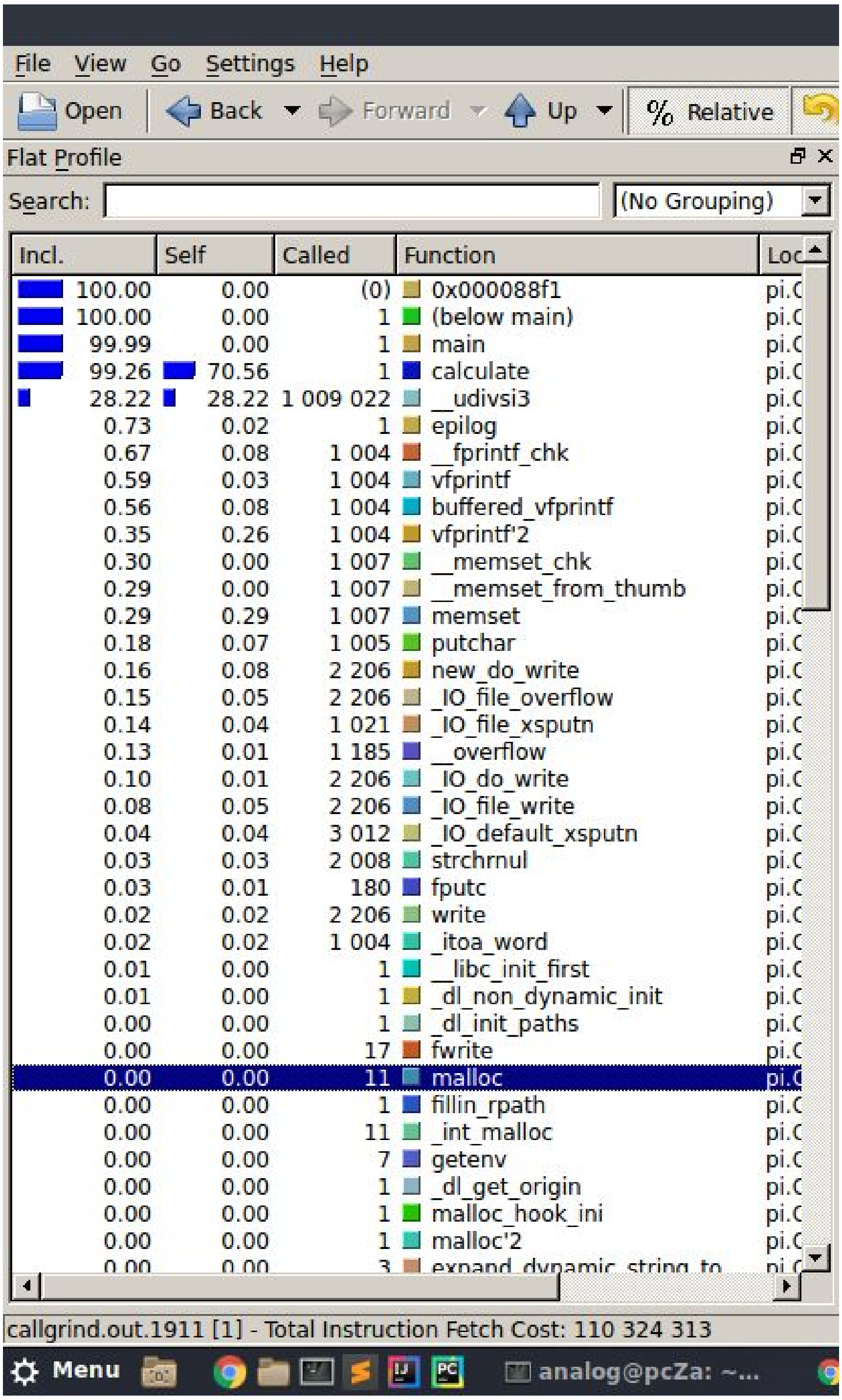


Incl.	Self	Called	Function
100.00	0.00	(0)	0x00000bc1
100.00	0.00	1	_dl_start
99.90	0.00	1	0x0000851d
99.90	0.00	1	(below main)
99.90	0.00	1	main
99.90	0.00	1	calculate
99.14	0.00	1	__udivsi3
99.14	98.66	1	<cycle 1>
70.96	70.47	1 004	calculate'2 <cycle 1>
28.18	28.18	1 009 019	__udivsi3'2 <cycle 1>
0.94	0.08	1 005	putchar
0.75	0.00	1	main'2
0.75	0.02	1	epilog
0.69	0.08	1 004	__fprintf_chk
0.61	0.03	1 004	vfprintf
0.58	0.08	1 004	buffered_vfprintf
0.37	0.28	1 004	vfprintf'2
0.30	0.00	1 007	__memset_chk
0.29	0.00	1 007	___GI_memset_from_thumb
0.29	0.29	1 007	memset
0.16	0.08	2 206	new_do_write
0.15	0.05	2 206	_IO_file_overflow@@GLIBC...
0.14	0.04	1 021	_IO_file_xsputn@@GLIBC_2.4
0.13	0.01	1 185	__overflow
0.10	0.01	2 206	_IO_do_write@@GLIBC_2.4
0.09	0.00	1	_dl_start_final
0.09	0.00	1	_dl_sysdep_start
0.09	0.00	1	dl_main
0.09	0.00	1	_dl_new_object
0.09	0.00	1	_dl_init_paths
0.09	0.00	1	do_preload
0.08	0.00	1	_dl_map_object_deps
0.08	0.05	2 206	_IO_file_write@@GLIBC_2.4
0.07	0.00	1	_dl_receive_error
0.07	0.00	1	init_tls
0.07	0.02	4	_dl_relocate_object
0.06	0.01	96	_dl_lookup_symbol_x

a) Han desaparegut totes les rutines menys el calculate.

b) Degut al nivell d’optimització O3, que activa la desplegament de bucles i intercalació de funcions automàtiques.

VALGRIND O3 STATIC



Incl.	Self	Called	Function	Loc
100.00	0.00	(0)	0x000088f1	pi.c
100.00	0.00	1	(below main)	pi.c
99.99	0.00	1	main	pi.c
99.26	70.56	1	calculate	pi.c
28.22	28.22	1 009 022	__udivsi3	pi.c
0.73	0.02	1	epilog	pi.c
0.67	0.08	1 004	__fprintf_chk	pi.c
0.59	0.03	1 004	vfprintf	pi.c
0.56	0.08	1 004	buffered_vfprintf	pi.c
0.35	0.26	1 004	vfprintf'2	pi.c
0.30	0.00	1 007	__memset_chk	pi.c
0.29	0.00	1 007	__memset_from_thumb	pi.c
0.29	0.29	1 007	memset	pi.c
0.18	0.07	1 005	putchar	pi.c
0.16	0.08	2 206	new_do_write	pi.c
0.15	0.05	2 206	_IO_file_overflow	pi.c
0.14	0.04	1 021	_IO_file_xsputn	pi.c
0.13	0.01	1 185	__overflow	pi.c
0.10	0.01	2 206	_IO_do_write	pi.c
0.08	0.05	2 206	_IO_file_write	pi.c
0.04	0.04	3 012	_IO_default_xsputn	pi.c
0.03	0.03	2 008	strchrnul	pi.c
0.03	0.01	180	fputc	pi.c
0.02	0.02	2 206	write	pi.c
0.02	0.02	1 004	_itoa_word	pi.c
0.01	0.00	1	__libc_init_first	pi.c
0.01	0.00	1	_dl_non_dynamic_init	pi.c
0.00	0.00	1	_dl_init_paths	pi.c
0.00	0.00	17	fwrite	pi.c
0.00	0.00	11	malloc	pi.c
0.00	0.00	1	fillin_rpath	pi.c
0.00	0.00	11	_int_malloc	pi.c
0.00	0.00	7	getenv	pi.c
0.00	0.00	1	_dl_get_origin	pi.c
0.00	0.00	1	malloc_hook_ini	pi.c
0.00	0.00	1	malloc'2	pi.c
0.00	0.00	3	expand_dynamic_string_to	pi.c

a) Han desaparegut totes les rutines menys el calculate i ha aparegut el write i el main.

b) Degut al nivell d'optimització O3, que activa la desplegament de bucles i intercalació de funcions automàtiques.

1.2.2 Profiling with oprofile

6. Compile your pi.c program using gcc and O0 optimization level and the debug option. Perform two oprofile of the pi.c program varying the counter value that indicates the frequency of the sample of the CPU CYCLES event (first counter that appears in the output of ophelp command). Use values 750000 and 100000: frequency is 1/counter. Compare the results obtained with opreport -l.

(a) Why do you think that there are those differences in the samples column?

```
 CPU_CYCLES =100000
analog@pcZa:~/Laboratori/Sessio2/lab2_session$ opreport -l
Using /home/analog/Laboratori/Sessio2/lab2_session/oprofile_data/samples/ for samples
directory.
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
samples  %        image name           symbol name
136877   31.8189  pi.O0.g.no_static    __aeabi_uidiv
117135   27.2296  pi.O0.g.no_static    SUBTRACT
114716   26.6673  pi.O0.g.no_static    DIVIDE
40716     9.4650  pi.O0.g.no_static    LONGDIV
14294     3.3228  pi.O0.g.no_static    .divsi3_skip_div0_test
5167      1.2011  no-vmlinux           /no-vmlinux
544       0.1265  pi.O0.g.no_static    __divsi3
169       0.0393  libc-2.19.so         memset
142       0.0330  libc-2.19.so         vfprintf
50        0.0116  libc-2.19.so         putchar
45        0.0105  libc-2.19.so         _IO_file_write@@GLIBC_2.4
42        0.0098  libc-2.19.so         new_do_write
37        0.0086  libc-2.19.so         write
35        0.0081  libc-2.19.so         buffered_vfprintf
32        0.0074  libc-2.19.so         _IO_file_overflow@@GLIBC_2.4
29        0.0067  pi.O0.g.no_static    calculate
21        0.0049  libc-2.19.so         _IO_do_write@@GLIBC_2.4
19        0.0044  libc-2.19.so         _IO_file_xsputn@@GLIBC_2.4
16        0.0037  libc-2.19.so         _IO_default_xsputn
14        0.0033  pi.O0.g.no_static    epilog
13        0.0030  libc-2.19.so         __overflow
11        0.0026  libc-2.19.so         strchrnul
11        0.0026  pi.O0.g.no_static    MULTIPLY
10        0.0023  libc-2.19.so         fprintf
9         0.0021  pi.O0.g.no_static    progress
7         0.0016  libc-2.19.so         fputc
6         0.0014  libc-2.19.so         _itoa_word
```

```
5         0.0012  pi.O0.g.no_static       SET
1        2.3e-04  ld-2.19.so              check_match.12909
1        2.3e-04  ld-2.19.so              do_lookup_x
1        2.3e-04  libc-2.19.so            fwrite
```

CPU_CYCLES = 750000

```
analog@pcZa:~/Laboratori/Sessio2/lab2_session$ opreport -l
Using /home/analog/Laboratori/Sessio2/lab2_session/oprofile_data/samples/ for samples directory.
CPU: ARM Cortex-A9, speed 1998 MHz (estimated)
Counted CPU_CYCLES events (CPU cycle) with a unit mask of 0x00 (No unit mask) count 750000
samples  %        image name               symbol name
17701    31.8696  pi.O0.g.no_static       __aeabi_uidiv
15110    27.2046  pi.O0.g.no_static       SUBTRACT
14739    26.5367  pi.O0.g.no_static       DIVIDE
5190      9.3443  pi.O0.g.no_static       LONGDIV
1858      3.3452  pi.O0.g.no_static       .divsi3_skip_div0_test
804       1.4476  no-vmlinux               /no-vmlinux
67        0.1206  pi.O0.g.no_static       __divsi3
27        0.0486  libc-2.19.so            memset
7         0.0126  libc-2.19.so            buffered_vfprintf
7         0.0126  libc-2.19.so            vfprintf
7         0.0126  libc-2.19.so            write
3         0.0054  libc-2.19.so            new_do_write
3         0.0054  libc-2.19.so            putchar
3         0.0054  libc-2.19.so            strchrnul
2         0.0036  libc-2.19.so            _IO_default_xsputn
2         0.0036  libc-2.19.so            _IO_file_xsputn@@GLIBC_2.4
2         0.0036  libc-2.19.so            fputc
2         0.0036  pi.O0.g.no_static       calculate
2         0.0036  pi.O0.g.no_static       progress
1         0.0018  libc-2.19.so            _IO_do_write@@GLIBC_2.4
1         0.0018  libc-2.19.so            _IO_file_overflow@@GLIBC_2.4
1         0.0018  libc-2.19.so            __overflow
1         0.0018  libc-2.19.so            fprintf
1         0.0018  pi.O0.g.no_static       MULTIPLY
1         0.0018  pi.O0.g.no_static       SET
```

Hi ha molta diferència en els samples ja que hem canviat la freqüència d’interrupció i en el segon cas, s’interromp molt menys que en el primer cas.

7. Compile your pi.c program using gcc and O3 optimization level and the debug option. Perform a oprofile of the pi.c program that indicates the frequency of the sample of the CPU CYCLES event is 1/100000. Compare the results obtained with this profiling to the profiling obtained in previous exercise with the same frequency. Use opreport and opannotate.

(a) What are the main differences? Why?

```
analog@pcZa:~/Laboratori/Sessio2/lab2_session$ opreport -l
Using /home/analog/Laboratori/Sessio2/lab2_session/oprofile_data/samples/ for samples
directory.
warning: /no-vmlinux could not be found.
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
samples  %        image name           symbol name
98241    70.4828  pi.O3.g.no_static    calculate
21903    15.7143  pi.O3.g.no_static    __aeabi_uidiv
12689     9.1037  pi.O3.g.no_static    .divsi3_skip_div0_test
5247      3.7644  no-vmlinux           /no-vmlinux
448       0.3214  pi.O3.g.no_static    __divsi3
308       0.2210  libc-2.19.so         memset
149       0.1069  libc-2.19.so         vfprintf
49        0.0352  libc-2.19.so         _IO_file_write@@GLIBC_2.4
49        0.0352  libc-2.19.so         putchar
48        0.0344  libc-2.19.so         new_do_write
37        0.0265  libc-2.19.so         _IO_file_overflow@@GLIBC_2.4
34        0.0244  libc-2.19.so         write
28        0.0201  libc-2.19.so         _IO_file_xsputn@@GLIBC_2.4
27        0.0194  libc-2.19.so         buffered_vfprintf
25        0.0179  libc-2.19.so         __fprintf_chk
20        0.0143  pi.O3.g.no_static    epilog
16        0.0115  libc-2.19.so         _IO_default_xsputn
16        0.0115  libc-2.19.so         _IO_do_write@@GLIBC_2.4
15        0.0108  libc-2.19.so         strchrnul
11        0.0079  libc-2.19.so         __overflow
7         0.0050  libc-2.19.so         __memset_chk
6         0.0043  libc-2.19.so         ____GI_memset_from_thumb
5         0.0036  libc-2.19.so         _itoa_word
2         0.0014  libc-2.19.so         fwrite
1         7.2e-04 ld-2.19.so           _dl_check_map_versions
1         7.2e-04 ld-2.19.so           _dl_relocate_object
1         7.2e-04 ld-2.19.so           dl_main
```

Degut a les optimitzacions, han desaparegut les operacions bàsiques i s'han agrupat en una única funció calculate. Hi ha aproximadament 110000 samples de diferència amb la funció de llibreria __aeabi_uidiv de O0 amb O3.

1.2.3 Instrumenting with system calls and PAPI

8. The pi times.c program uses the system call times() in order to show the execution time (decomposed in user mode and system mode) for each call to calculate().

(a) Observe the differences between pi.c and pi times.c programs and how the system call times() is called. Indicate if the time shown by the program is CPU time or elapsed time. Can the system call times() provide both CPU and elapsed time?

TIMES():
Timing amb crida times: user 61.029999 segons, system: 0.090000 segons
El temps que mostra la rutina times() és el CPU time.
No podem mostrar el elapsed time ja que només mostra tasques cridades pel nostre procés (i fills).

(b) Modify pi times.c program so that getrusage() system call is used instead of times() system call. Indicate if getrusage() can provide both CPU and elapsed time? Do you have less or more precision compared to "times" results? Considerations:

• struct timeval struct uses to be defined at /usr/include/bits/time.h file.

• In order to obtain time format 1.035 seconds, the tv sec field of the struct timeval struct will have value 1 and tv usec field will have value 35000 (0.035 seconds are 35000 microseconds).

GETRUSAGE():
CPU time: 61.040000 sec user, 0.140000 sec system
Tampoc podem saber el Elapsed time. Tenim més precisió en aquest cas.

(c) Modify pi times.c program so that PAPI is used in order to provide CPU CYCLES and the CPU time of calculate function. Note that papi avail provides information about the frequency of the processor.

analog@pcZa:~/Laboratori/Sessio2/lab2_session$ papi_avail
Available events and hardware information.

PAPI Version   : 5.3.0.0
Vendor string and code   : ARM (7)
Model string and code   : ARMv7 Processor rev 0 (v7l) (0)

CPU Revision : 0.000000
CPU Max Megahertz : 666
CPU Min Megahertz : 666
Hdw Threads per core : 1
Cores per Socket : 2
Sockets : 1
CPUs per Node : 2
Total CPUs : 2
Running in a VM : no
Number Hardware Counters : 2
Max Multiplex Counters : 64

Output given: 41035187584 cycles, 61 s
Version with more precision: 40979766127 cycles, 61531180 us

2 Automatization and data managment tools

9. Create an script that automatizes the execution of the program pi.c for NMIN up to NMAX (with NSTEP step) number of decimals.

The script has the following arguments:
• Executable program of the original (no optimized) pi.c.
• Executable program of the possible optimized pi.c (to be done in next sessions).
• NMIN and NMAX : minimum and maximum number of decimals.
• NSTEP value: loop step.
• NEXEC value: number of executions to do average of execution time.

First, the script should check the correct result for each execution of the optimized program, comparing its results to the original program results. If there is any difference the script should give a message ”No correct results for N=Value” and stop the execution of the script. If everything is fine, it should run the optimized program and generate a text file with the following format for each line:

number_of_decimals elapsed_time

For instance: 500 1.3454 1000 2.1234 1500 3.9834 ...

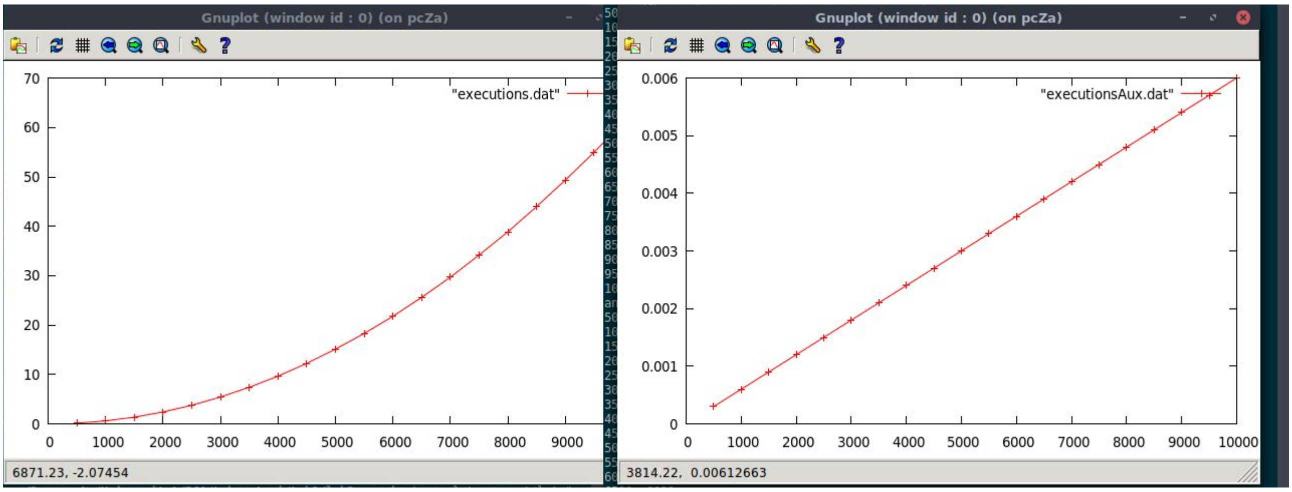
Where elapsed time is the average of the elapsed time of the NEXEC executions done.

Script annexat al document.

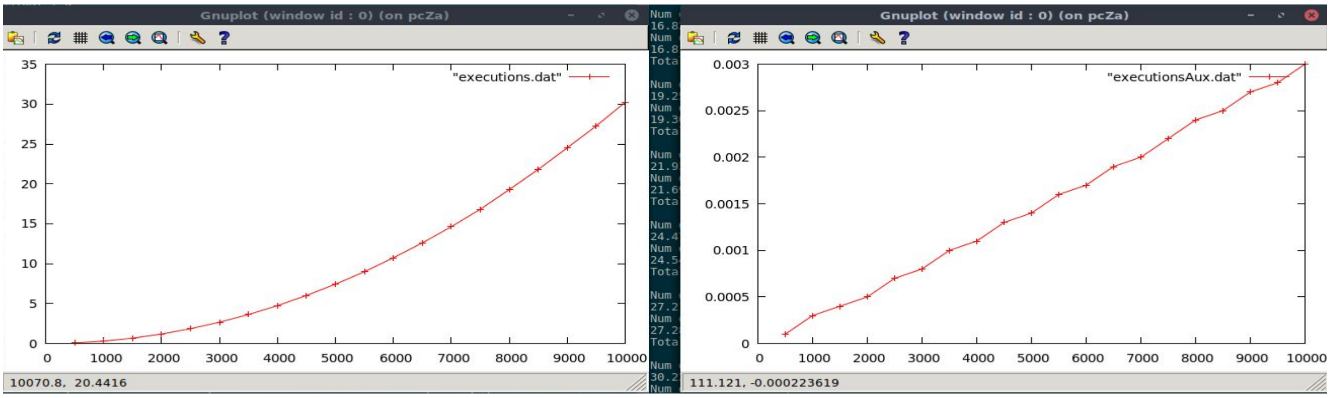
(a) One figure that shows elapsed time (Y axis) function of the number of decimals computed (X axis) for the original pi.c compiled with ”-O0”, ”-O1”, and ”-O3 -march=native” compiler options. You can create a figure for each case or all cases in the same figure.

(b) Another figure that shows the elapsed time/number of decimals computed (Y axis) function of the number of decimals computed (X axis) for the pi.c program for ”-O0”, ”-O1”, and ”-O3 -march=native” compiler options. You can create a figure for each case or all cases in the same figure.

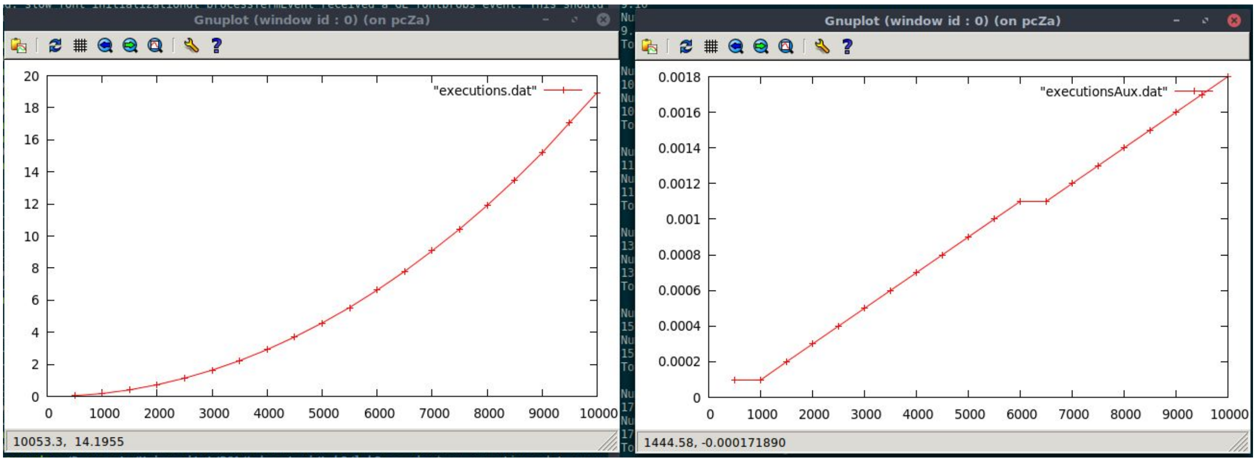
Execució Script O0. A l’esquerra time per steps. A la dreta time/steps per steps



Execució Script O1. A l’esquerra time per steps. A la dreta time/steps per steps



Execució Script O3. A l'esquerra time per steps. A la dreta time/steps per steps



(c) Explain the execution differences and the shape of the figures.

L'execució sobre O3 triga 3 cops menys aproximadament que l'execució sobre O0. El creixement del temps no és linial (més aviat és exponencial) degut al creixament del nombre de steps.

(d) Prepare an script (similar to a regression test) to run your script with two examples to test it:

i. First test a correct program: The original executable program should be the pi.c compiled with -O0. The optimized executable program should be the pi.c compiled with -O3.NEXEC value should be three. The rest of parameters can be: NMIN= 500, NMAX=1000 and NSTEP= 500.

Script annexat al document.

ii. Second test an incorrect program: The original executable program should be the pi.c compiled with -O0. The optimized executable program should be the popul.c compiled with -O3 (YES! popul.c, it should be incorrect :). NEXEC value should be three. The rest of parameters can be: NMIN= 500, NMAX= 1000 and NSTEP= 500.

Script annexat al document.