Giulia Franco
matricola SM3500370
Year 2018/2019
Exercise 2, High Performance Computing course.

Profiling

Profiling means finding the pathological spots (*hot spots*) in the code in order to optimize it. We assume a quite easy program for the exercise: a cpp code with the implementation of the inverse of a matrix. The profiling procedure is done using gprof and perf tools.

gprof output

time(%)	comulative (s)	self (s)	calls	self s/call	total s/call	name
99.93	3.82	3.82	102	0.04	0.04	determinant
0.26	3.83	0.01				main
0.13	3.83	0.01	1	0.01	3.78	cofactor
0.00	3.83	0.00	1	0.00	0.04	transpose

It's possible to notice that the majority of the time it's spent in the *determinant* function. This isn't surprising since the condition for the existence of the inverse of a matrix is $Det(M) \neq 0$, for an M random matrix.
We can also profile the code using perf for both event based profiling (*perf stat*) and sample based profiling (*perf record*). The following tables report the results.

perf report output

time(%)	calling function	function called
85,17%	inverse	determinant
10,58%	inverse	mcount_internal
3,47%	inverse	_mcount
0,48%	inverse	mcount@plt
0,28%	inverse	[k] 0xffffffff8104315a
0,01%	inverse	profil_counter
0,01%	inverse	ieee754_pow

Samples: 17K of event 'cycles', Event count (approx.): 13249931193

perf stat output

4434,980037 task-clock	1,000 CPUs utilized
87 context-switches	0,020 K/sec
2 cpu-migrations	0,000 K/sec
172 page-faults	0,039 K/sec
13258602515 cycles	2,990 GHz
3110296868 stalled-cycles-frontend	23,46% frontend cycles idle
376870025 stalled-cycles-backend	62,84% backend cycles idle
30724215449 instructions	2,32 insns per cycle
	0,10 stalled cycles per insn
3886260866 branches	876,275 M/sec
12339895 branch-misses	0,32% of all branches
4,434061291 seconds time elapsed	

Finally it's possible to represent the dynamics of the execution using a calltree graph. In order to obtain that we can use *gproftodot*.

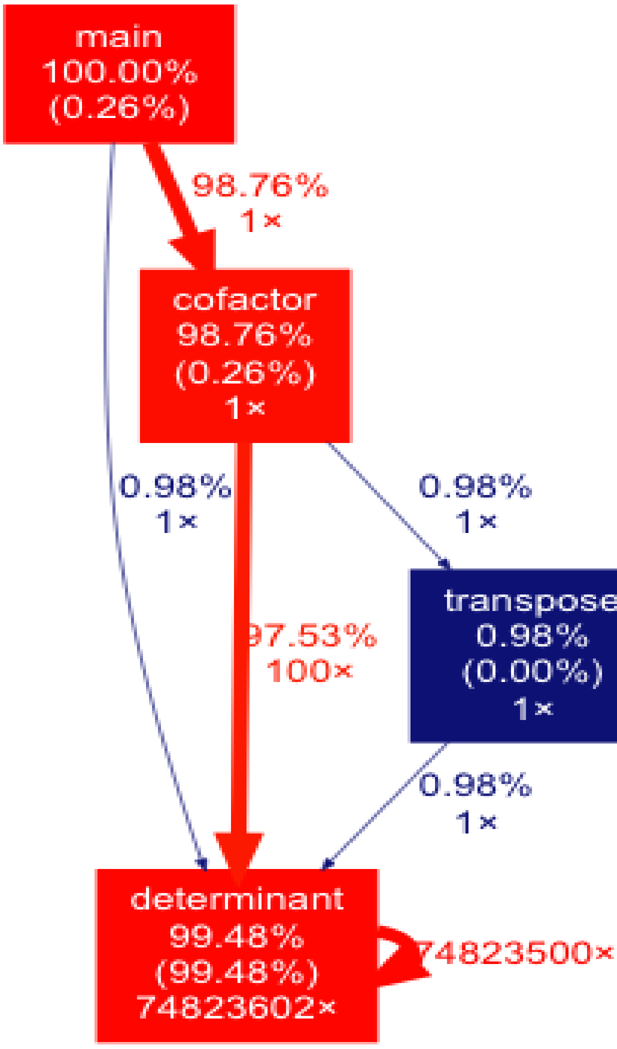

Figura 1: Call tree using gprof

From the analysis we can conclude that the most efficient way to improve the performance, if necessary, of the code is to optimize the function *determinant*, perhaps using loop unrolling.