Grindset - profiling tests

xblazo00

April 2022

1 Profiling information

1.1 First round of tests

Running the original input for the profiling (10, 100 and 1000 numbers) has yielded close to no results due to low sampling rate of gprof.

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	2220	0.00	0.00	mathlib::add(long double, long double)
0.00	0.00	0.00	1113	0.00	0.00	mathlib::power(long double, long long)
0.00	0.00	0.00	6	0.00	0.00	mathlib::div(long double, long double)
0.00	0.00	0.00	6	0.00	0.00	mathlib::mul(long double, long double)
0.00	0.00	0.00	3	0.00	0.00	_GLOBAL__sub_I_main
0.00	0.00	0.00	3	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	3	0.00	0.00	mathlib::sub(long double, long double)
0.00	0.00	0.00	3	0.00	0.00	mathlib::getRoot(long double, long long)
0.00	0.00	0.00	3	0.00	0.00	std::pow(long double, long double)

Here, we can see that the most called function was "mathlib::add", having almost twice the amount of calls as mathlib::power. However, due to low sampling size by gprof, we did not get any readings on the time percentage.

1.2 Second round of tests

To overcome the low sampling rate, we increased the amount of numbers fed to the program (100 thousand, 1 million and 10 million numbers) and got the best results from the 10 million numbers input. It yielded interesting results.

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ns/call	total ns/call	name
33.36	0.11	0.11	20000000	5.50	5.50	mathlib::add(long double, long double)
33.36	0.22	0.11	10000001	11.01	11.01	mathlib::power(long double, long long)
33.36	0.33	0.11				main
0.00	0.33	0.00	2	0.00	0.00	mathlib::div(long double, long double)
0.00	0.33	0.00	2	0.00	0.00	mathlib::mul(long double, long double)
0.00	0.33	0.00	1	0.00	0.00	_GLOBAL__sub_I_main
0.00	0.33	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.33	0.00	1	0.00	0.00	mathlib::sub(long double, long double)
0.00	0.33	0.00	1	0.00	0.00	mathlib::getRoot(long double, long long)
0.00	0.33	0.00	1	0.00	0.00	std::pow(long double, long double)

In this image, we can see that despite being called twice as much as mathlib::power function, mathlib::add was 2 times faster. The impact of the other functions used only once or twice was negligible.

1.3 Conclusion

From the provided data, we have concluded that the program spends the most time in the power and add functions. This means that in order for optimization to be most efficient, these two functions should be targeted.