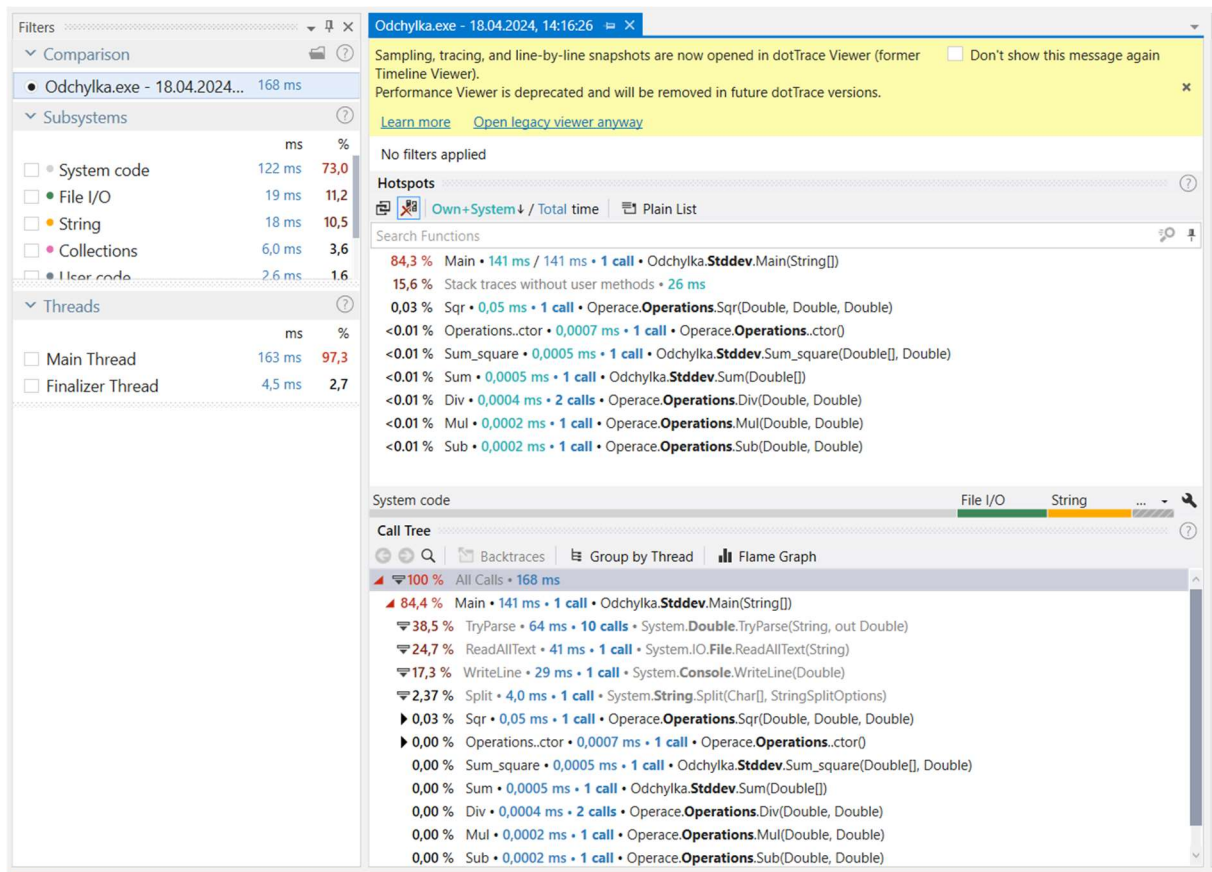


IVS - profiling

Tým – Lorem ipsum
18.04.2024

Obsahem, je výstup profileru pro 10, 1000, 1000000 čísel. Kde lze pozorovat počty volání a čas strávený voláním těchto metod.

Pro 10 čísel:



Pro 1000 čísel:

The screenshot displays the Visual Studio Performance Profiler interface for the application `Odchylka.exe`, dated 18.04.2024 at 14:19:49. The interface is divided into several panes:

- Filters:** Shows the selected application `Odchylka.exe - 18.04.2024...` with a total time of 193 ms.
- Subsystems:** A table listing various subsystems and their execution times and percentages.
- Threads:** A table listing the threads of the application.
- Hotspots:** A list of the most time-consuming operations.
- Call Tree:** A hierarchical view of the method calls during execution.

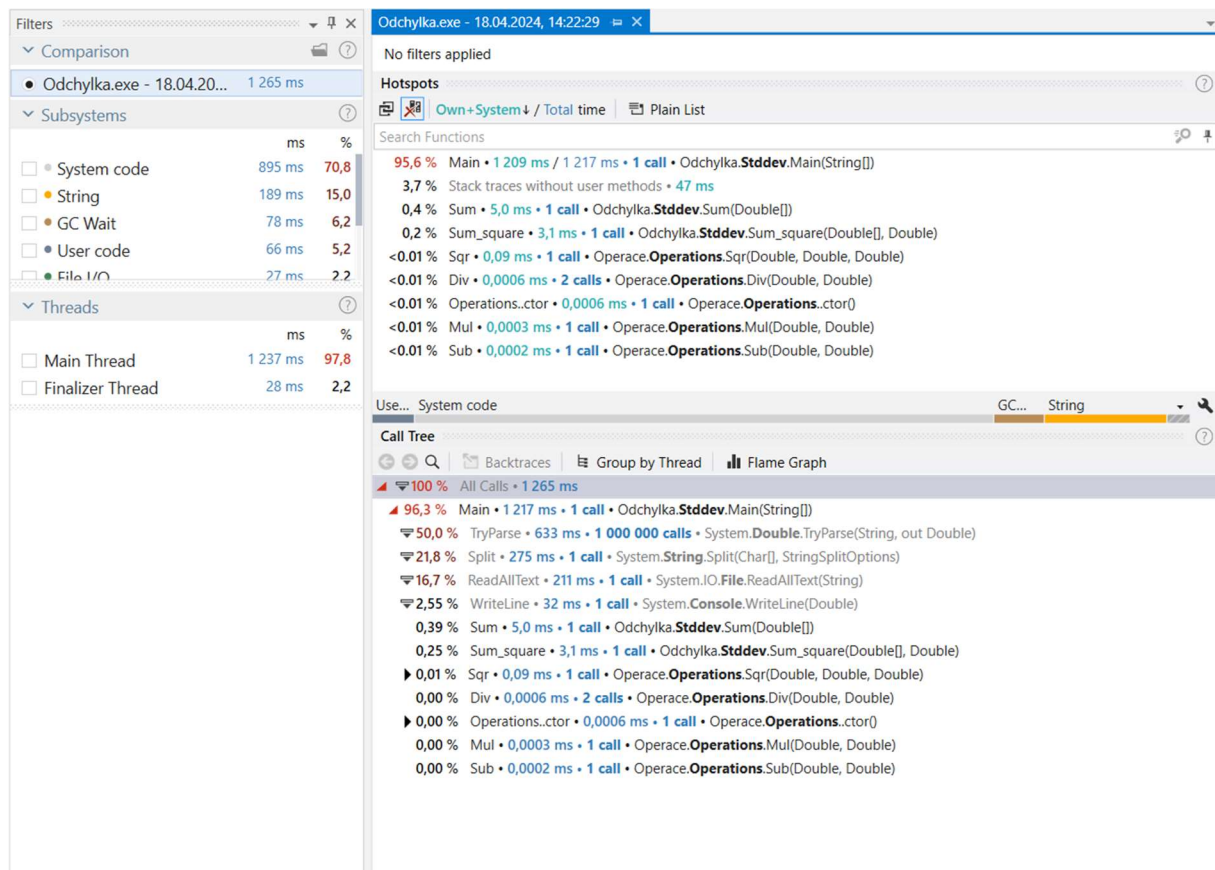
Subsystem	ms	%
System code	146	75,9
String	19	9,8
File I/O	18	9,2
Collections	8,7	4,5
User code	1,0	0,5

Thread	ms	%
Main Thread	188	97,6
Finalizer Thread	4,6	2,4

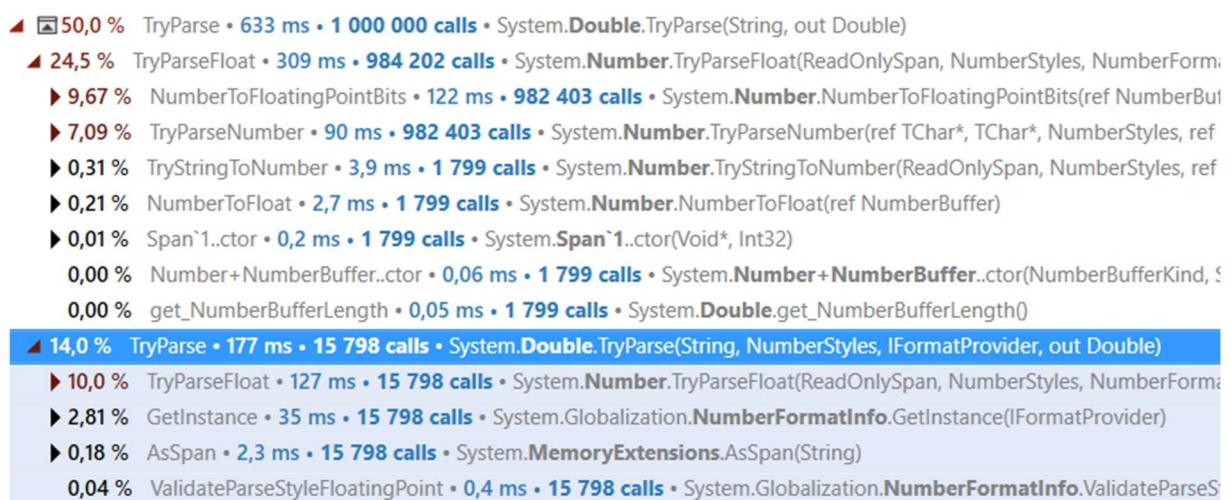
Hotspot	ms	%
Main	165	85,7
Stack traces without user methods	28	14,3
Sqr	0,06	0,03
Sum	0,004	<0,01
Sum_square	0,003	<0,01
Operations..ctor	0,0007	<0,01
Div	0,0003	<0,01
Sub	0,0002	<0,01
Mul	0,0002	<0,01

Call	ms	%
Main	165	85,7
TryParse	72	37,3
ReadAllText	57	29,8
WriteLine	26	13,4
Split	9,2	4,79
Sqr	0,06	0,03
Sum	0,004	0,00
Sum_square	0,003	0,00
Operations..ctor	0,0007	0,00
Div	0,0003	0,00
Sub	0,0002	0,00
Mul	0,0002	0,00

Pro 1000000 čísel:



Je vidět, že program tráví nejvíce času převedením obsahu na double.



Možnou optimalizací by bylo umožnit uživateli zadávat pouze celá čísla, tím bychom se vyhnuli built-in funkcím jako TryParse a došlo by ke zrychlení programu.

Problém nastává, když vložíme např. 300 000 čísel s hodnotami 10^6 a více.

The screenshot displays the Visual Studio performance profiler interface. The top-left pane shows 'Subsystems' with a table of execution times and percentages. The top-right pane shows 'Hotspots' with a table of function calls. The bottom-left pane shows 'Threads' with a table of thread execution times. The bottom-right pane shows the 'Call Tree' with a hierarchical view of function calls. The rightmost pane shows the source code of the program.

Subsystem	ms	%
User code	198 669	52,0
System code	183 157	47,9
GC Wait	266	0,07

Thread	ms	%
Main Thread	381 825	99,9
Finalizer Thread	266	0,07

Function	ms	%
Sqr	198 669	52,0
Abs	183 157	47,9
[Garbage collection]	266	0,07

Function	ms	%
Sqr	198 669	52,0
Abs	183 157	47,9
[Garbage collection]	266	0,07

```
81 {
82     num1 *= baseNum;
83 }
84 return num1;
85 }
86
87 /**
88  * Funkce obecné odmocniny s přirozenými exponenty
89  *
90  * @param num1 První operand pro operaci - základ
91  * @param num2 Druhý operand pro operaci - exponent
92  * @return Vrací výsledek operace
93  */
94 public double Sqr(double num1, double num2, double epsilon)
95 {
96     double guess = num2 / num1; // Začínáme od 1/n vstupn
97     while (Math.Abs(num2 - Math.Pow(guess, num1)) > epsilon)
98     {
99         guess = ((num1 - 1) * guess + num2 / Math.Pow(guess, num1 - 1)) / num1;
100     }
101     return guess;
102 }
103
104 /**
105  * Funkce faktoriálu
106  */
```

Z čehož plyne, že optimalizací by bylo použití jiného výpočtu n -té odmocniny.