

# Report k profileru - součást týmovému projektu do předmětu IVS

**Jméno:** Dominik Hofman

**Login:** xhofma11

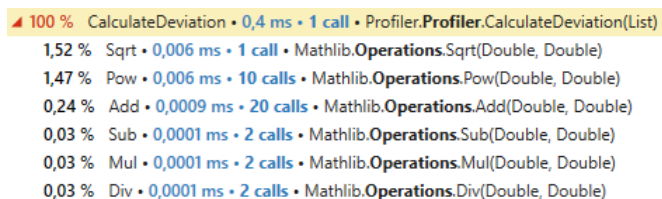
**Tým:** xhofma11

**Datum:** 23. dubna 2024

## 1 Úvod

Program prošel podrobným profilováním za použití výkonného nástroje **DotTrace**, který umožňuje hlubokou analýzu běhu programu. Typ profilace byl specificky nastaven na **Tracing**, což je metoda sledování spuštění a ukončení jednotlivých metod v programu. Tato forma profilace umožňuje detailní pohled na to, jak jednotlivé části kódu interagují a jaký vliv mají na celkový výkon aplikace. Analyzovány jsou nejen časy spuštění a ukončení funkcí, ale také všechny volání metod včetně jejich parametrů a návratových hodnot. Tímto způsobem získáváme komplexní povědomí o chování programu a identifikujeme možné úzká hrdla nebo neefektivní části kódu. Tato data nám poskytují cenné informace pro optimalizaci výkonu aplikace a zlepšení uživatelské zkušenosti.

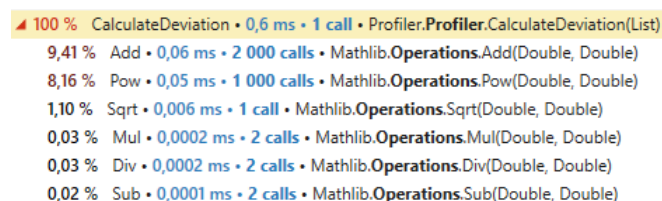
## 2 Výsledky profilace



▲ 100 % CalculateDeviation • 0,4 ms • 1 call • Profiler.Profiler.CalculateDeviation(List)

1,52 %	Sqrt	• 0,006 ms • 1 call	• Mathlib.Operations.Sqrt(Double, Double)
1,47 %	Pow	• 0,006 ms • 10 calls	• Mathlib.Operations.Pow(Double, Double)
0,24 %	Add	• 0,0009 ms • 20 calls	• Mathlib.Operations.Add(Double, Double)
0,03 %	Sub	• 0,0001 ms • 2 calls	• Mathlib.Operations.Sub(Double, Double)
0,03 %	Mul	• 0,0001 ms • 2 calls	• Mathlib.Operations.Mul(Double, Double)
0,03 %	Div	• 0,0001 ms • 2 calls	• Mathlib.Operations.Div(Double, Double)

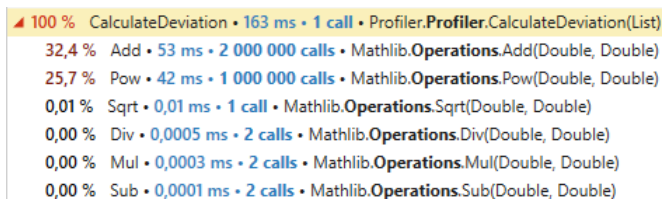
Obrázek 1: Profilace se vstupy 10 číselných hodnot



▲ 100 % CalculateDeviation • 0,6 ms • 1 call • Profiler.Profiler.CalculateDeviation(List)

9,41 %	Add	• 0,06 ms • 2 000 calls	• Mathlib.Operations.Add(Double, Double)
8,16 %	Pow	• 0,05 ms • 1 000 calls	• Mathlib.Operations.Pow(Double, Double)
1,10 %	Sqrt	• 0,006 ms • 1 call	• Mathlib.Operations.Sqrt(Double, Double)
0,03 %	Mul	• 0,0002 ms • 2 calls	• Mathlib.Operations.Mul(Double, Double)
0,03 %	Div	• 0,0002 ms • 2 calls	• Mathlib.Operations.Div(Double, Double)
0,02 %	Sub	• 0,0001 ms • 2 calls	• Mathlib.Operations.Sub(Double, Double)

Obrázek 2: Profilace se vstupy 1 000 číselných hodnot



▲ 100 % CalculateDeviation • 163 ms • 1 call • Profiler.Profiler.CalculateDeviation(List)

32,4 %	Add	• 53 ms • 2 000 000 calls	• Mathlib.Operations.Add(Double, Double)
25,7 %	Pow	• 42 ms • 1 000 000 calls	• Mathlib.Operations.Pow(Double, Double)
0,01 %	Sqrt	• 0,01 ms • 1 call	• Mathlib.Operations.Sqrt(Double, Double)
0,00 %	Div	• 0,0005 ms • 2 calls	• Mathlib.Operations.Div(Double, Double)
0,00 %	Mul	• 0,0003 ms • 2 calls	• Mathlib.Operations.Mul(Double, Double)
0,00 %	Sub	• 0,0001 ms • 2 calls	• Mathlib.Operations.Sub(Double, Double)

Obrázek 3: Profilace se vstupy 1 000 000 číselných hodnot

### 3 Závěr

Pokud se podíváme na výsledek profilace se vstupy 10 číselných hodnot, zdá se, že největším problémem bude mocnina a odmocnina. Pro tak malé vstupy, kdy výpočet trvá necelou milisekundu, je to však zanedbatelné. Více informací lze zjistit až z větších vstupů, tedy 1000 a 1000 000 číselných hodnot, kdy lze vidět, že největší podíl na výkonu funkce **CalculateDeviation** mají sčítání a mocnění.

Když se ovšem podíváme blíže na počet volání těchto matematických operací, můžeme vypořádat, že mocnění bylo voláno méně o jednu polovinu než sčítání a přesto její vykonání trvá skoro stejně jako u sčítání.

Při optimalizaci je nutno se tedy zaměřit zejména na funkci mocnění, u které je předpoklad, že s velkými daty bude její výpočet trvat déle.

Níže je v bodech sepsáno, na co se při optimalizaci zejména zaměřit v kontextu implementace:

- Algoritmus umocňování
- Vhodné datové typy
- Ošetření okrajových podmínek
- Efektivní práce s pamětí
- Paralelizace a distribuce úloh