VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta informačných technologií

Praktické aspekty vývoje software 2017/2018

Projekt č. 2

Profiling – správa

Úvod

Účelom tohto dokumentu je analyzovať výstup profileru pri profilovaní programu na výpočet smerodajnej odchýlky a navrhnúť možný postup pri ďalšej optimalizácii programu. Na profiling bol využitý nástroj *DotTrace* od *JetBrains*.

Výstupy profileru

```
■ 97,33 % Main • 1 838 ms • 1 call • StandartDeviation.Program.Main(String[])
▼ 92,44 % Read • 1 746 ms • 1 call • System.Console.Read
                                     WriteLine • 79 ms • 2 calls • System.Console.WriteLine(String)

      4 ≡ 0,17 %
      GetRandomNumbers • 3 ms • 1 call • StandartDeviation.Program.GetRandomNumbers(Int32)

      ▼ 0,03 %
      Add • 1 ms • 10 calls • System.Collections.Generic.List*1.Add(T)

                    0,00 % Random..ctor • 0 ms • 1 call • System.Random..ctor(Int3)
                    0.00 % List`1..cctor • 0 ms • 1 call • System.Collections.Generic.List`1..cctor
                     0,00 % InternalSample • 0 ms • 10 calls • System.Random.InternalSample
        ■ 0,16 % GetStandartDeviation • 3 ms • 1 call • StandartDeviation.Program.GetStandartDeviation(List)
■ 0,02 % GetArithmeticMean • 0 ms • 1 call • StandartDeviation.Program.GetArithmeticMean(List)
                            0,00 % add • 0 ms • 10 calls • Calculator.MatLib.add(Double, Double)
                            0,00 % divide • 0 ms • 1 call • Calculator.MatLib.divide(Double, Double)
                  $\begin{array}{c} \equiv 0,00 \% \text{ divide } \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ call culator.MatLib.divide(Double, Double)} \\
\begin{array}{c} \equiv 0,00 \% \text{ square } \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ standartDeviation.Program.GetSumSqr(List)} \\
\equiv 0,00 \% \text{ square } \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ calculator.MatLib.square(Double)} \\
\equiv 0,00 \% \text{ square } \cdot 0 \text{ ms} \cdot 1 \text{ calls} \cdot \text{ Calculator.MatLib.add(Double, Double)} \\
\equiv 0,00 \% \text{ square } \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ Calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ square } \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ Calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ square } \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ Calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ sqrt} \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ Calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ sqrt} \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ sqrt} \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ sqrt} \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ sqrt} \cdot 0 \text{ ms} \cdot 1 \text{ call } \cdot 0 \text{ ms} \text{ call } \cdot 0 \text{ ms} \text{ call } \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ sqrt} \cdot 0 \text{ ms} \cdot 1 \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ sqrt} \cdot 0 \text{ ms} \cdot 1 \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \% \text{ sqrt} \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \Rightarrow 0 \text{ ms} \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \Rightarrow 0 \text{ ms} \text{ calculator.MatLib.sqrt(Double)} \\
\equiv 0,00 \Rightarrow 0 \text{ ms} \text{ calculator.MatLib.sqrt(Doubl
                      0,00 % subtract • 0 ms • 1 call • Calculator.MatLib.subtract(Double, Double)
0,00 % multiply • 0 ms • 1 call • Calculator.MatLib.multiply(Double, Double)
0,00 % divide • 0 ms • 1 call • Calculator.MatLib.divide(Double, Double)
■ 0,00 % square • 0 ms • 1 call • Calculator.MatLib.square(Double)
■ 0,14 % WriteLine • 3 ms • 1 call • System.Console.WriteLine(String, Object)
■ 1,51 % CreateSecurityIdentity • 28 ms • 1 call • System.Security.Policy.PEFileEvidenceFactory.CreateSecurityIdentity(SafePEFileHandle, Evidence)
▼ 0,83 % SetupDomain • 16 ms • 1 call • System.AppDomain.SetupDomain(Boolean, String, String, String[], String[])
▼ 0,19 % InitializeDomainSecurity • 4 ms • 1 call • System.AppDomain.InitializeDomainSecurity(Evidence, Evidence, Boolean, IntPtr, Boolean)
▼ 0,08 % InitializeSwitches • 1 ms • 1 call • System.CompatibilitySwitches.InitializeSwitches
   3,07 % UpgradeSecurityIdentity • 1 ms • 1 call • System.Security.Policy.AssemblyEvidenceFactory.UpgradeSecurityIdentity(Evidence, RuntimeAssembly)
0,00 % 4 functions hidden • 0 ms total • 4 calls total
```

Obrázek 1: Profiling pre 10 prvkov

```
Main Thread • 1 40
       96.55 % Main • 1 357 ms • 1 call • Standart Deviation, Program, Main (String[])
    ▼ 89,88 % Read • 1 263 ms • 1 call • System.Console.Read
▼ 5,73 % WriteLine • 81 ms • 2 calls • System.Console.WriteLine(String)
    ▲ 3 ms • 1 call • StandartDeviation.Program.GetRandomNumbers(Int32)
       ► ■ 0,04 % Add • 1 ms • 100 calls • System.Collections.Generic.List`1.Add(T) 0,00 % InternalSample • 0 ms • 100 calls • System.Random.InternalSample
          0,00 % Random..ctor • 0 ms • 1 call • System.Random..ctor(Int32)
0,00 % List`1..cctor • 0 ms • 1 call • System.Collections.Generic.List`1..cctor
    ■ 0,22 % GetStandartDeviation • 3 ms • 1 call • StandartDeviation.Program.GetStandartDeviation(List)
■ 0,03 % GetSumSqr • 0 ms • 1 call • StandartDeviation.Program.GetSumSqr(List)
■ 0,00 % square • 0 ms • 100 calls • Calculator.MatLib.square(Double)
              ■ 0,00 % add • 0 ms • 100 calls • Calculator.MatLib.add(Double, Double)
          ■ 0,03 % GetArithmeticMean • 0 ms • 1 call • StandartDeviation.Program.GetArithmeticMean(List)
              ■ 0,00 % add • 0 ms • 100 calls • Calculator.MatLib.add(Double, Double)
           0,00 % divide • 0 ms • 1 call • Calculator.MatLib.divide(Double, Double)
0,00 % sqrt • 0 ms • 1 call • Calculator.MatLib.sqrt(Double)
           © 0,00 % squt on s o 1 call o Calculator.MatLib.multiply(Double, Double)

© 0,00 % subtract o 0 ms o 1 call o Calculator.MatLib.subtract(Double, Double)

© 0,00 % divide o 0 ms o 1 call o Calculator.MatLib.divide(Double, Double)
              0,00 % square • 0 ms • 1 call • Calculator.MatLib.square(Double)
▼ 0,19 % WriteLine • 3 ms • 1 call • System.Console.WriteLine(String, Object)
▼ 2,00 % CreateSecurityIdentity • 28 ms • 1 call • System.Security.Policy.PEFIleEvidenceFactory.CreateSecurityIdentity(SafePEFileHandle, Evidence)
■ 1,05 % SetupDomain • 15 ms • 1 call • System.AppDomain.SetupDomain(Boolean, String), String[])  
■ 0,24 % InitializeDomainSecurity • 3 ms • 1 call • System.AppDomain.InitializeDomainSecurity(Evidence, Evidence, Boolean, IntPtr, Boolean)
🔻 0,09 % UpgradeSecurityIdentity • 1 ms • 1 call • System.Security.Policy.AssemblyEvidenceFactory.UpgradeSecurityIdentity(Evidence, RuntimeAssembly)
  7 0,07 % InitializeSwitches • 1 ms • 1 call • System.CompatibilitySwitches.InitializeSwitches
0,00 % 4 functions hidden • 0 ms total • 4 calls total
```

Obrázek 2: Profiling pre 100 prvkov

```
| Main Thread • 3 530 ms | 3 478 ms • 1 call • StandartDeviation.Program.Main(String[]) | 98,52 % | Main • 3 478 ms • 1 call • System.Console.Read | 7 2,29 % | WriteLine • 81 ms • 2 calls • System.Console.WriteLine(String) | 95,86 % | Read • 3 384 ms • 1 call • StandartDeviation.Program.GetRandomNumbers(Int32) | 90,10 % | GetRandomNumbers • 4 ms • 1 call • StandartDeviation.Program.GetStandartDeviation(List) | 90,00 % | GetStandartDeviation • 3 ms • 1 call • StandartDeviation.Program.GetArithmeticMean(List) | 90,00 % | GetArithmeticMean • 1 ms • 1 call • StandartDeviation.Program.GetArithmeticMean(List) | 90,00 % | doi/die • 0 ms • 1 call • Calculator.MatLib.ddiv(de(Double, Double) | 90,00 % | doi/die • 0 ms • 1 call • Calculator.MatLib.ddiv(de(Double, Double) | 90,00 % | doi/die • 0 ms • 1 call • StandartDeviation.Program.GetSumSqr(List) | 90,00 % | subtract • 0 ms • 1 call • Calculator.MatLib.sdu(Double, Double) | 90,00 % | subtract • 0 ms • 1 call • Calculator.MatLib.sdu(Double, Double) | 90,00 % | subtract • 0 ms • 1 call • Calculator.MatLib.square(Double) | 90,00 % | subtract • 0 ms • 1 call • Calculator.MatLib.square(Double) | 90,00 % | sqrt • 0 ms • 1 call • Calculator.MatLib.square(Double) | 90,00 % | sqrt • 0 ms • 1 call • Calculator.MatLib.square(Double) | 90,00 % | sqrt • 0 ms • 1 call • Calculator.MatLib.square(Double) | 90,00 % | sqrt • 0 ms • 1 call • Calculator.MatLib.square(Double) | 90,00 % | sqrt • 0 ms • 1 call • System.Security.Policy.PEFIleEvidenceFactory.CreateSecurityIdentity(SafePEFileHandle, Evidence) | 90,00 % | sqrt • 0 ms • 1 call • System.Security.Policy.PEFIleEvidenceFactory.CreateSecurityIdentity(SafePEFileHandle, Evidence) | 90,00 % | 10 kitializeDomainSecurity • 3 ms • 1 call • System.AppDomain.SetupDomain.Geoclean, String, String[], String[]) | 90,00 % | 10 kitializeDomainSecurity • 1 ms • 1 call • System.AppDomain.InitializeDomainSecurity.Evidence, Evidence, Boolean, IntPrr, Boolean) | 90,00 % | 10 kitializeSwitches • 1 ms • 1 call • System.CompatibilitySwitches.InitializeSwi
```

Obrázek 3: Profiling pre 1000 prvkov

Záver:

Z výstupu profilera je zrejmé, že najčastejšie volanými funkciami sú funkcie **add** (sčítanie) a **square** (druhá mocnina) z nami vytvorenej matematickej knižnice.

Pre N prvkov je funkcia **add** volaná pri výpočte smerodajnej odchýlky dokopy 2*N $kr\acute{a}t$: N- $kr\acute{a}t$ pri výpočte aritmetického priemeru funkciou GetArithmeticMean a N- $kr\acute{a}t$ pri výpočte sumy druhých mocnín všetkých prvkov funkciou GetSumSqr.

Funkcia **square** je pre N prvkov volaná pri výpočte smerodajnej odchýlky práve N+1 krát: N-krát pri výpočte sumy druhých mocnín všetkých prvkov funkciou GetSumSqrt a raz v tele funkcie GetStandartDeviation na umocnenie aritmetického priemeru.

Zvyšné matematické funkcie **substract** (odčítanie), **multiply** (násobenie) a **sqrt** (druhá odmocnina) sú pri výpočte volané iba *1-krát* a funkcia **divide** (delenie) *2-krát* bez ohľadu na to, koľko prvkov má postupnosť. Z tohto dôvodu nie je potrebné tieto funkcie ďalej optimalizovať.

Pri ďalšej optimalizácii výpočtu smerodajnej odchýlky by sme sa predovšetkým zamerali na zrýchlenie funkcií **add** a **square**.