IVS - profiling

Lidé u výtahu

April 2020

Obsah

1 Úvod

Pro výpočet výběrové směrodatné odchylky naše skupina zvolila vývoj konzolové aplikace s názvem SampleStandardDeviation.exe. Tato aplikace používá třídu IVSMath s matematickými operacemi. Ze stdin načte libovolný počet čísel a na stdout vypíše výběrovou směrodatnou odchylku. Nepovinný argument N určí, kolikrát se má každá funkce třídy IVSMath spustit.

Příklady spuštění:

```
$ ./SampleStandardDeviation.exe < /data/data10.txt
$ ./SampleStandardDeviation.exe 1000 < /data/data10.txt
```

2 Profiling

Ve Visual Studiu 2019 jsme naši aplikaci profilovali pomocí Performance profileru, který jako výstup vytváří soubor s příponou .diagsession. Tento soubor obsahuje veškerá data zjištěná při profilingu a dá se otevřít přímo ve Visual Studiu. Pro rychlejší a pohodlnější zobrazení jsme vytvořili i screenshoty funkce Main, které ale zdaleka nezobrazují vše potřebné.

Soubory *.diagsession mimo jiné obsahují tabulku funkcí s jednotkami CPU, které daná funkce spotřebovala za běhu aplikace. Zobrazuje funkce, které spotřebují alespoň jednu jednotku CPU, ostatní nezahrnuje. To stejné platí pro zobrazení náročnosti jednotlivých řádků.

2.1 Jednoduchý profiling

V první fázi jsme profilovali aplikaci bez argumentu N. Každá funkce se tedy vykonala pouze jednou. Výsledky jsou v souborech vystup-*.(diagsession|png). Viz Obrázek 1, kde je výsledek profilingu aplikace se vstupem ze souboru data-/data1000.txt.

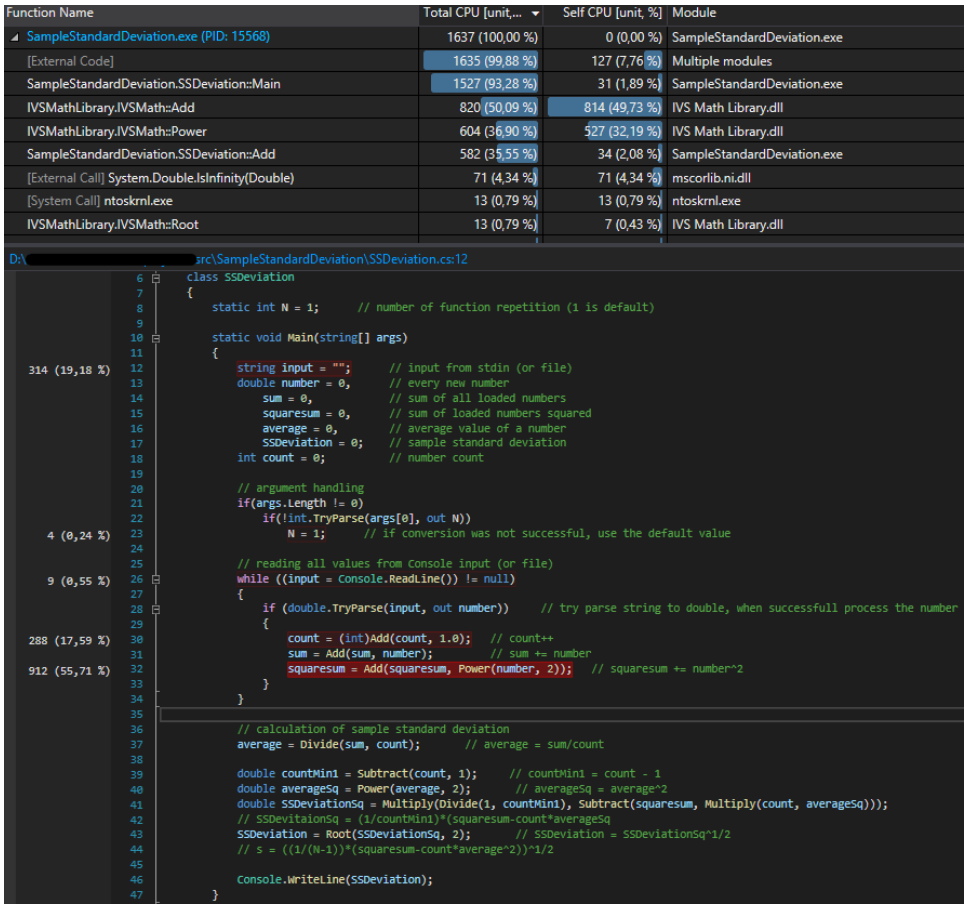
```
                namespace SampleStandardDeviation
                {
                    class SSDeviation
                    {
                        static int N = 1;      // number of function repetition (1 is default)

                        static void Main(string[] args)
                        {
2 (2,00 %)              string input = "";      // input from stdin (or file)
                        double number = 0,      // every new number
                            sum = 0,            // sum of all loaded numbers
                            squaresum = 0,      // sum of loaded numbers squared
                            average = 0,        // average value of a number
                            SSDeviation = 0;    // sample standard deviation
                        int count = 0;          // number count

                        // argument handling
                        if(args.Length != 0)
                            if(!int.TryParse(args[0], out N))
                                N = 1;      // if conversion was not successful, use the default value

                        // reading all values from Console input (or file)
2 (2,00 %)              while ((input = Console.ReadLine()) != null)
                        {
                            if (double.TryParse(input, out number))     // try parse string to double, when successfull process the number
                            {
1 (1,00 %)                      count = (int)Add(count, 1.0);   // count++
                                sum = Add(sum, number);         // sum += number
                                squaresum = Add(squaresum, Power(number, 2));   // squaresum += number^2
                            }
                        }

                        // calculation of sample standard deviation
                        average = Divide(sum, count);       // average = sum/count

                        double countMin1 = Subtract(count, 1);      // countMin1 = count - 1
                        double averageSq = Power(average, 2);       // averageSq = average^2
                        double SSDeviationSq = Multiply(Divide(1, countMin1), Subtract(squaresum, Multiply(count, averageSq)));
                        // SSDevitaionSq = (1/countMin1)*(squaresum-count*averageSq
                        SSDeviation = Root(SSDeviationSq, 2);       // SSDeviation = SSDeviationSq^1/2
                        // s = ((1/(N-1))*(squaresum-count*average^2))^1/2

                        Console.WriteLine(SSDeviation);
                    }
```

Obrázek 1: Výstup jednoduchého profileru

2.2 Profiling s opakováním

Jednoduchý profiling nám přinesl zajímavá ale veskrze ne moc použitelná data pro případnou optimalizaci. Proto jsme profilovali s argumentem N = 100, 1000 a 10000. Každá funkce se při každém volání zopakuje Nkrát. Výstup profileru s opakováním nám ukáže mnohem přesněji, kde je potřeba optimalizovat.

Viz Obrázek 2, kde je výsledek profilingu aplikace se vstupem ze souboru data/data1000.txt a argumentem N=10000. Na tomto výstupu je vidět, že nejvíce času strávil program ve funkcích IVSMath.Add a IVSMath.Power. Na tyto funkce bychom se tedy měli soustředit při případné optimalizaci.



Function Name	Total CPU [unit,...	Self CPU [unit, %]	Module
SampleStandardDeviation.exe (PID: 15568)	1637 (100,00 %)	0 (0,00 %)	SampleStandardDeviation.exe
[External Code]	1635 (99,88 %)	127 (7,76 %)	Multiple modules
SampleStandardDeviation.SSDeviation::Main	1527 (93,28 %)	31 (1,89 %)	SampleStandardDeviation.exe
IVSMathLibrary.IVSMath::Add	820 (50,09 %)	814 (49,73 %)	IVS Math Library.dll
IVSMathLibrary.IVSMath::Power	604 (36,90 %)	527 (32,19 %)	IVS Math Library.dll
SampleStandardDeviation.SSDeviation::Add	582 (35,55 %)	34 (2,08 %)	SampleStandardDeviation.exe
[External Call] System.Double.IsInfinity(Double)	71 (4,34 %)	71 (4,34 %)	mscorlib.ni.dll
[System Call] ntoskrnl.exe	13 (0,79 %)	13 (0,79 %)	ntoskrnl.exe
IVSMathLibrary.IVSMath::Root	13 (0,79 %)	7 (0,43 %)	IVS Math Library.dll


```
6   class SSDeviation
7   {
8       static int N = 1;      // number of function repetition (1 is default)
9
10      static void Main(string[] args)
11      {
12          string input = "";      // input from stdin (or file)
13          double number = 0,      // every new number
14              sum = 0,            // sum of all loaded numbers
15              squaresum = 0,      // sum of loaded numbers squared
16              average = 0,        // average value of a number
17              SSDeviation = 0;    // sample standard deviation
18          int count = 0;          // number count
19
20          // argument handling
21          if(args.Length != 0)
22              if(!int.TryParse(args[0], out N))
23                  N = 1;      // if conversion was not successful, use the default value
24
25          // reading all values from Console input (or file)
26          while ((input = Console.ReadLine()) != null)
27          {
28              if (double.TryParse(input, out number))     // try parse string to double, when successfull process the number
29              {
30                  count = (int)Add(count, 1.0);   // count++
31                  sum = Add(sum, number);         // sum += number
32                  squaresum = Add(squaresum, Power(number, 2));   // squaresum += number^2
33              }
34          }
35
36          // calculation of sample standard deviation
37          average = Divide(sum, count);       // average = sum/count
38
39          double countMin1 = Subtract(count, 1);     // countMin1 = count - 1
40          double averageSq = Power(average, 2);       // averageSq = average^2
41          double SSDeviationSq = Multiply(Divide(1, countMin1), Subtract(squaresum, Multiply(count, averageSq)));
42          // SSDevitaionSq = (1/countMin1)*(squaresum-count*averageSq
43          SSDeviation = Root(SSDeviationSq, 2);           // SSDeviation = SSDeviationSq^1/2
44          // s = ((1/(N-1))*(squaresum-count*average^2))^1/2
45
46          Console.WriteLine(SSDeviation);
47      }
```

Obrázek 2: Výstup profileru s opakováním

3 Závěr

Obě metody profilingu jsou v praxi přínosné. V našem případě bylo k dosažení cíle potřeba rozšířit původní zadání. Pomocí metody profilingu s opakováním jsme zjistili, které funkce jsou nejvíce náročné a měli by se optimalizovat pro zvýšení výkonu aplikace.

4 Přílohy

Název souboru	Popis
vystup-data10.diagsession vystup-data10.png	výstup profilingu s 10 vstupy screenshot s 10 vstupy
vystup-data100.diagsession vystup-data10.png	výstup profilingu se 100 vstupy screenshot se 100 vstupy
vystup-data1000.diagsession vystup-data10.png	výstup profilingu s 1000 vstupů screenshot s 1000 vstupů
vystup-data1000-withN-100.diagsession vystup-data1000-withN-100.png	výstup profilingu s opakováním, kde N=100 screenshot s opakováním 100x
vystup-data1000-withN-1000.diagsession vystup-data1000-withN-1000.png	výstup profilingu s opakováním, kde N=1000 screenshot s opakováním 1000x
vystup-data1000-withN-10000.diagsession vystup-data1000-withN-10000.png	výstup profilingu s opakováním, kde N=10000 screenshot s opakováním 10000x