

IVS - profiling

Lidé u výtahu

April 2020

Obsah

1 Úvod	1
2 Profiling	1
2.1 Jednoduchý profiling	2
2.2 Profiling s opakováním	2
3 Závěr	2
4 Přílohy	3

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
```

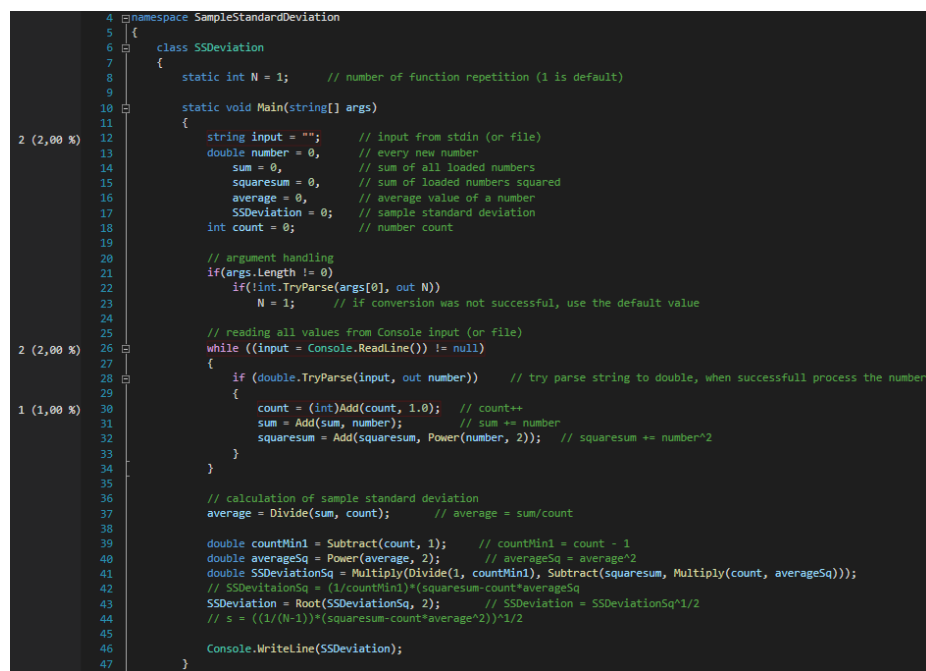
2 Profiling

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, ale z důvodu velikosti jsme je neuložili. Pro rychlejší a pohodlnější zobrazení jsme vytvořili screenshoty funkce `Main` s využitím jednotlivých řádků.

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-*.png. Viz Obrázek 1, kde je výsledek profilingu aplikace se vstupem ze souboru data/data1000.txt.



```
4 namespace SampleStandardDeviation
5 {
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            {
23                if(!int.TryParse(args[0], out N))
24                    N = 1; // if conversion was not successful, use the default value
25            }
26
27            // reading all values from Console input (or file)
28            while ((input = Console.ReadLine()) != null)
29            {
30                if (double.TryParse(input, out number)) // try parse string to double, when successful process the number
31                {
32                    count = (int)Add(count, 1.0); // count++
33                    sum = Add(sum, number); // sum += number
34                    squaresum = Add(squaresum, Power(number, 2)); // squaresum += number^2
35                }
36            }
37
38            // calculation of sample standard deviation
39            average = Divide(sum, count); // average = sum/count
40
41            double countMini = Subtract(count, 1); // countMini = count - 1
42            double averageSq = Power(average, 2); // averageSq = average^2
43            double SSDeviationSq = Multiply(Divide(1, countMini), Subtract(squaresum, Multiply(count, averageSq)));
44            // SSDeviationSq = (1/countMini)*(squaresum-count*averageSq)
45            SSDeviation = Root(SSDeviationSq, 2); // SSDeviation = SSDeviationSq^1/2
46            // s = ((1/(N-1))*(squaresum-count*average^2))^1/2
47            Console.WriteLine(SSDeviation);
48        }
49    }
50 }
```

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.

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

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.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

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

jsme zjistili, které funkce jsou nejvíce náročné a měli by se optimalizovat pro zvýšení výkonu aplikace.

4 Přílohy

Název souboru	Popis
vystup-data10.png	screenshot profilingu s 10 vstupy
vystup-data100.png	screenshot profilingu se 100 vstupy
vystup-data1000.png	screenshot profilingu s 1000 vstupy
vystup-data1000-withN-100.png	screenshot s opakováním 100x
vystup-data1000-withN-1000.png	screenshot s opakováním 1000x
vystup-data1000-withN-10000.png	screenshot s opakováním 10000x