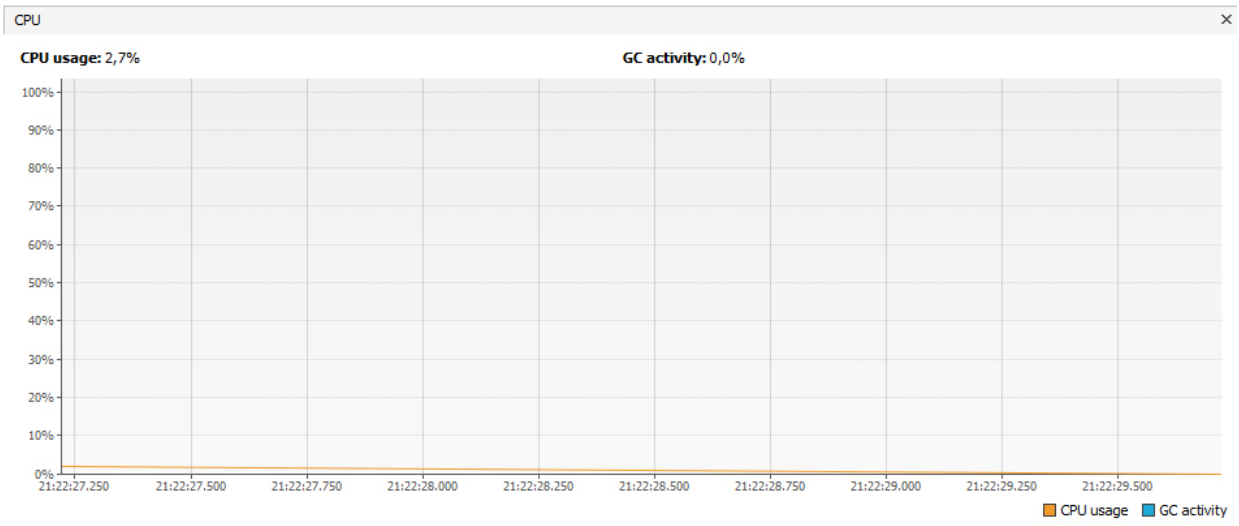
IVS - profiling

OAGUH

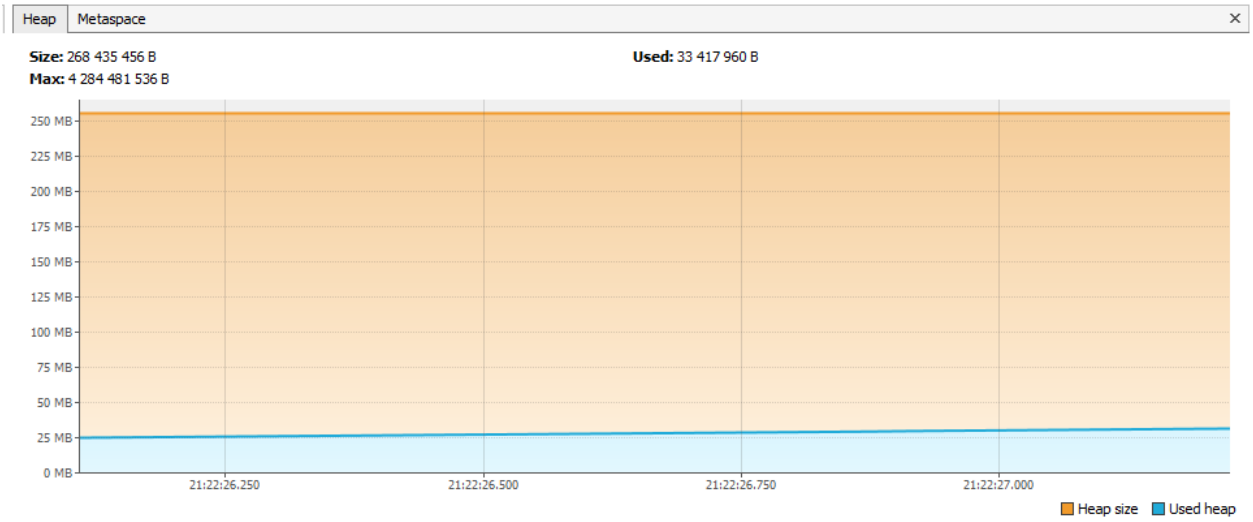
10. dubna 2020

Pro profilování programu na výpočet směrodatné odchylky byl použit profiler `VisualVM`. Jako vstupní data byla použita posloupnost čísel o velikosti 1000. Posloupnost obsahovala kladná, záporná, desetinná čísla, bez desetinných míst a 0.

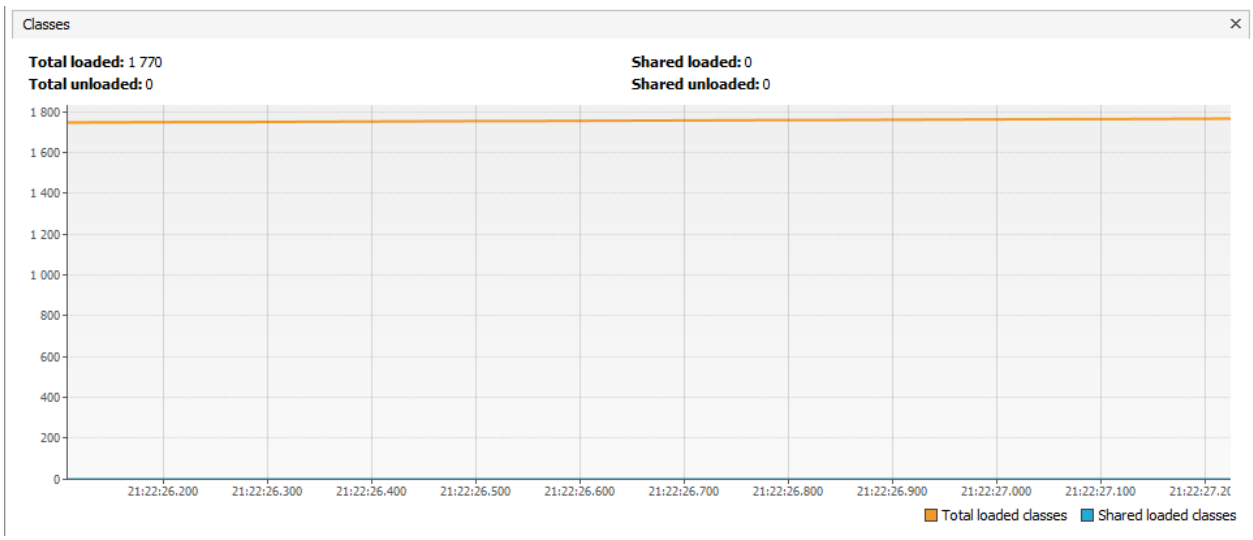
Při běhu aplikace se zatížení procesoru pohybovalo 0,5 – 3 % (testováno na Intel core i5 - 6600K 4,5 GHz). Obsazenost hromady neměla žádné vyrázené skoky v poklesu či zvětšení. Obsazenost začala na 25 MB, lineárně se zvětšovala a skončila na 33 MB. Celkový počet načtených tříd byl 1770 a počet zapnutých vláken 17. viz. obrázky 1, 2, 3, 4



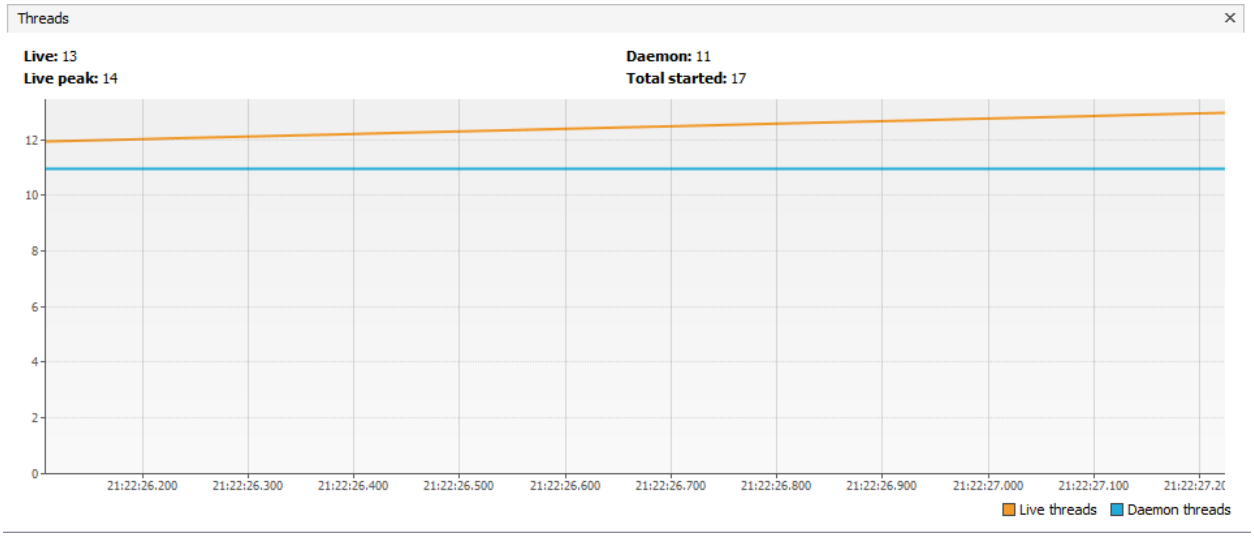
Obrázek 1: Graf CPU



Obrázek 2: Graf hromady



Obrázek 3: Graf tříd



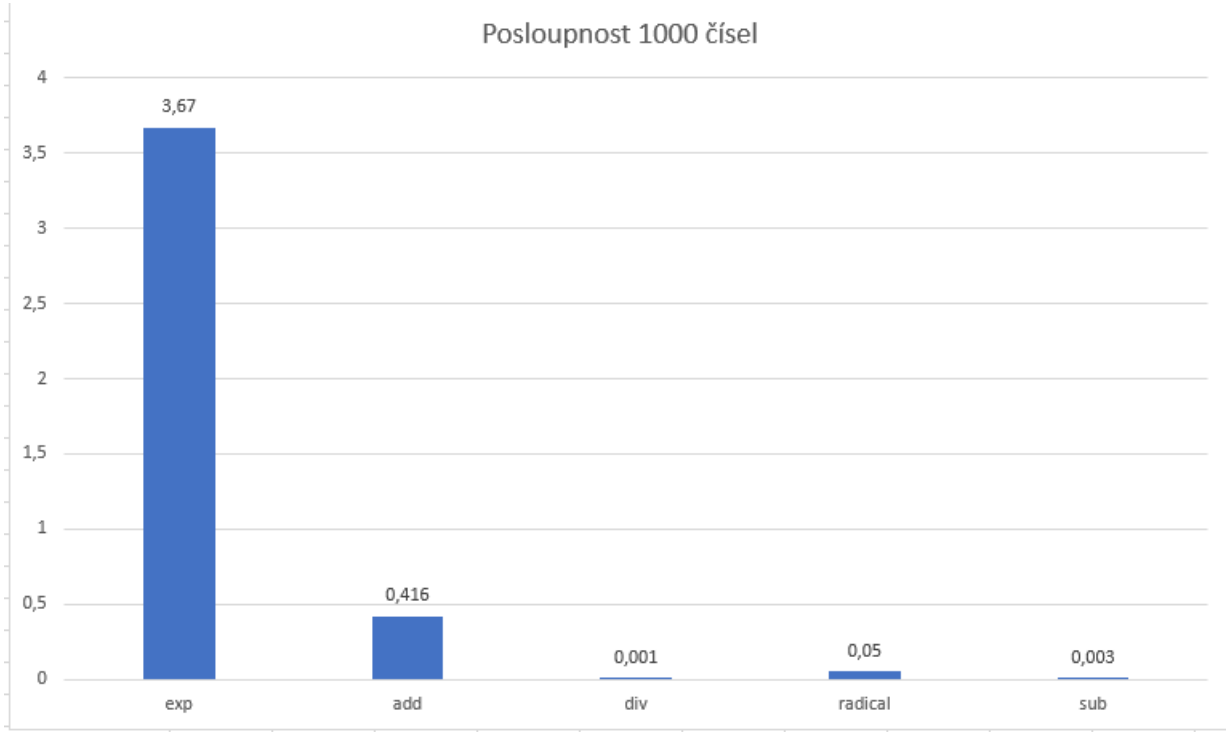
Obrázek 4: Graf vláken

Na snímku 5 vidíme, že nejčastěji se provedla metoda `add` s 2000 zavoláním. S druhým největším počtem byla funkce `exp` s 1001 zavoláním.

Name	Total Time		Total Time (CPU)		Invocations
main	5,63 ms	(100%)	12,3 ms	(100%)	1
CalculatorLibraryProfiling.Deviation.**standardDeviation** (java.util.ArrayList)	5,63 ms	(100%)	12,3 ms	(100%)	1
CalculatorLibraryProfiling.Deviation.**squareArrayLIst** (java.util.ArrayList)	4,68 ms	(83%)	0,0 ms	(0%)	1
CalculatorUtils.Utilities.**exp** (double, int)	3,82 ms	(67,9%)	0,0 ms	(0%)	1 000
Self time	0,851 ms	(15,1%)	0,0 ms	(0%)	1
CalculatorLibraryProfiling.Deviation.**sum** (java.util.ArrayList)	0,431 ms	(7,6%)	14,5 ms	(117,7%)	1
Self time	0,222 ms	(3,9%)	0,0 ms	(0%)	1
CalculatorUtils.Utilities.**add** (double, double)	0,208 ms	(3,7%)	15,0 ms	(122,1%)	1 000
CalculatorLibraryProfiling.Deviation.**average** (java.util.ArrayList)	0,407 ms	(7,2%)	0,0 ms	(0%)	1
CalculatorLibraryProfiling.Deviation.**sum** (java.util.ArrayList)	0,397 ms	(7%)	0,0 ms	(0%)	1
Self time	0,264 ms	(4,7%)	0,0 ms	(0%)	1
CalculatorUtils.Utilities.**add** (double, double)	0,133 ms	(2,4%)	0,0 ms	(0%)	1 000
Self time	0,008 ms	(0,1%)	0,0 ms	(0%)	1
CalculatorUtils.Utilities.**div** (double, double)	0,0 ms	(0%)	0,0 ms	(0%)	1
Self time	0,060 ms	(1,1%)	0,0 ms	(0%)	1
CalculatorUtils.Utilities.**radical** (double, int)	0,052 ms	(0,9%)	0,0 ms	(0%)	1
CalculatorUtils.Utilities.**sub** (double, double)	0,004 ms	(0,1%)	0,0 ms	(0%)	2
CalculatorUtils.Utilities.**exp** (double, int)	0,0 ms	(0%)	0,0 ms	(0%)	1
CalculatorUtils.Utilities.**div** (double, double)	0,0 ms	(0%)	0,0 ms	(0%)	1

Obrázek 5: Počet volání

Na obrázku 6 lze vidět celkový čas strávený v jednotlivých funkcích matematické knihovny. Můžeme si povšimnout, že nejvíc času se strávilo ve funkci `exp` a ve funkci `add`



Obrázek 6: Graf s časy strávených v jednotlivých funkcích matematické knihovny

I přesto, že funkce `exp` měla méně volání než funkce `add` program v ní strávil více času. Úsek kódu kde je metoda `exp` 1000x volána by se dal optimalizovat, tak, že by se nahradila voláním metody `mul` (metoda pro násobení). Funkce by se dala nahradit protože se jedná jen druhou mocninu daného čísla.

Při změně velikosti posloupnosti byl zjištěn velký časový skok mezi posloupností o velikosti 10 a 100 viz. tabulka 1.

Velikost	**Čas**
10	0,497 ms
100	4,25 ms
1000	5,63 ms

Tabulka 1: Časy pro jednotlivé posloupnosti