Protokol k profileru

IVS 2. projekt – CodePainCodeGain

Průběh měření:

Pro měření jsme vytvořily 3 separátní skripty pro porovnání výstupu profileru u různých optimalizacích. Profilování bylo provedeno v pythonu pomocí modulu cProfile výstup byl seřazen dle interního času vykonávání funkcí.

Kód a skripty:

Názvy skriptů a jejich význam:

- ***standard_deviation.py:*** Obsahuje prvotní a nejvíce intuitivní implementaci
- ***np_standard_deviation.py:*** Optimalizace čtení inputu a použití některých funkcí z knihovny numpy
- ***check_std_correctness.py:*** Načtení hodnot do stdin pomocí numpy, samostatný výpočet směrodatné odchylky funkcí stdev z modulu statistics.
- ***generate_nums.py:*** náhodné generování čísel pro výpočet směrodatné odchylky

Výstupy z profileru:

Ukázka je pro velikost vstupu o 10 000 000 číslech

```
python3 ../standard_deviation.py < test4-10to7.txt
577367520.9076962
Sat Apr 20 01:00:06 2024    profiling_stats

         40045586 function calls in 11.263 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    3.658    3.658    4.956    4.956 standard_deviation.py:9(read_stdin)
        1    2.805    2.805    4.204    4.204 standard_deviation.py:32(sum_of_squared_differences)
 20000000    1.344    0.000    1.344    0.000 calculatormathlib.py:24(add)
        1    1.266    1.266    1.920    1.920 standard_deviation.py:23(sum)
 10000000    0.710    0.000    0.710    0.000 calculatormathlib.py:75(pow2)
        1    0.632    0.632    0.632    0.632 {method 'split' of 'str' objects}
 10000000    0.459    0.000    0.459    0.000 {method 'append' of 'list' objects}
        1    0.182    0.182   11.263   11.263 standard_deviation.py:53(main)
        1    0.103    0.103    0.103    0.103 {method 'strip' of 'str' objects}
    22785    0.097    0.000    0.097    0.000 {built-in method _codecs.utf_8_decode}
    22785    0.008    0.000    0.105    0.000 codecs.py:319(decode)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    6.125    6.125 standard_deviation.py:40(standard_deviation)
        1    0.000    0.000   11.263   11.263 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 calculatormathlib.py:102(sqrt)
        2    0.000    0.000    0.000    0.000 calculatormathlib.py:60(div)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000   11.263   11.263 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 calculatormathlib.py:36(sub)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

```
python3 ../np_standard_deviation.py < test4-10to7.txt
577367520.9076962
Sat Apr 20 01:00:16 2024    profiling_stats

         20000025 function calls in 8.494 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    3.464    3.464    5.331    5.331
np_standard_deviation.py:20(sum_of_squared_differences)
        1    2.145    2.145    2.145    2.145 {built-in method numpy.array}
 10000000    0.940    0.000    0.940    0.000 calculatormathlib.py:75(pow2)
 10000000    0.927    0.000    0.927    0.000 calculatormathlib.py:24(add)
        1    0.635    0.635    0.635    0.635 {method 'split' of 'str' objects}
        1    0.177    0.177    3.146    3.146 np_standard_deviation.py:9(read_stdin)
        1    0.103    0.103    0.103    0.103 {built-in method _codecs.utf_8_decode}
        1    0.086    0.086    0.189    0.189 {method 'read' of '_io.TextIOWrapper' objects}
        1    0.009    0.009    8.493    8.493 np_standard_deviation.py:40(main)
        1    0.007    0.007    0.007    0.007 {method 'reduce' of 'numpy.ufunc' objects}
        1    0.001    0.001    8.494    8.494 <string>:1(<module>)
        1    0.000    0.000    8.494    8.494 {built-in method builtins.exec}
        1    0.000    0.000    5.338    5.338 np_standard_deviation.py:28(standard_deviation)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.103    0.103 codecs.py:319(decode)
        1    0.000    0.000    0.007    0.007 fromnumeric.py:2177(sum)
        1    0.000    0.000    0.007    0.007 fromnumeric.py:71(_wrapreduction)
        2    0.000    0.000    0.000    0.000 calculatormathlib.py:60(div)
        1    0.000    0.000    0.000    0.000 calculatormathlib.py:102(sqrt)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.isinstance}
        1    0.000    0.000    0.000    0.000 fromnumeric.py:72(<dictcomp>)
        1    0.000    0.000    0.000    0.000 calculatormathlib.py:36(sub)
        1    0.000    0.000    0.000    0.000 fromnumeric.py:2172(_sum_dispatcher)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.000    0.000    0.000    0.000 {method 'items' of 'dict' objects}
```

```
python3 check_std_correctness.py < test4-10to7.txt
577367520.9077069
Sat Apr 20 01:27:44 2024    profiling_stats

         60046703 function calls (60046701 primitive calls) in 21.735 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    5.881    5.881   16.500   16.500 statistics.py:697(_ss)
        1    3.748    3.748    5.056    5.056 check_std_correctness.py:13(read_stdin)
 20000000    3.619    0.000    3.619    0.000 {method 'as_integer_ratio' of 'float' objects}
        1    3.391    3.391    7.395    7.395 statistics.py:150(_sum)
 20000000    2.872    0.000    6.491    0.000 statistics.py:240(_exact_ratio)
 10000000    0.737    0.000    0.737    0.000 {method 'get' of 'dict' objects}
        1    0.640    0.640    0.640    0.640 {method 'split' of 'str' objects}
 10000000    0.462    0.000    0.462    0.000 {method 'append' of 'list' objects}
        1    0.180    0.180   21.735   21.735 check_std_correctness.py:24(main)
        1    0.102    0.102    0.102    0.102 {method 'strip' of 'str' objects}
    22785    0.096    0.000    0.096    0.000 {built-in method _codecs.utf_8_decode}
    22785    0.008    0.000    0.103    0.000 codecs.py:319(decode)
      142    0.000    0.000    0.000    0.000 fractions.py:62(__new__)
       70    0.000    0.000    0.000    0.000 fractions.py:451(_add)
      209    0.000    0.000    0.000    0.000 {built-in method math.gcd}
        2    0.000    0.000    0.001    0.000 {built-in method builtins.sum}
       70    0.000    0.000    0.000    0.000 fractions.py:356(forward)
```

```
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
       36    0.000    0.000    0.000    0.000 statistics.py:198(<genexpr>)
       36    0.000    0.000    0.000    0.000 statistics.py:721(<genexpr>)
      142    0.000    0.000    0.000    0.000 {built-in method __new__ of type object at
0x63a73bfe69a0}
        1    0.000    0.000   16.500   16.500 statistics.py:725(variance)
       72    0.000    0.000    0.000    0.000 {built-in method builtins.isinstance}
        1    0.000    0.000   21.735   21.735 {built-in method builtins.exec}
      141    0.000    0.000    0.000    0.000 fractions.py:256(numerator)
      141    0.000    0.000    0.000    0.000 fractions.py:260(denominator)
      2/1    0.000    0.000    0.000    0.000 {built-in method _abc._abc_subclasscheck}
        1    0.000    0.000   16.500   16.500 statistics.py:816(stdev)
        2    0.000    0.000    0.000    0.000 {built-in method _abc._abc_instancecheck}
        1    0.000    0.000    0.000    0.000 statistics.py:264(_convert)
       35    0.000    0.000    0.000    0.000 __init__.py:579(__missing__)
        2    0.000    0.000    0.000    0.000 fractions.py:368(reverse)
        1    0.000    0.000   16.500   16.500 check_std_correctness.py:8(compute_std)
        1    0.000    0.000    0.000    0.000 __init__.py:565(__init__)
        2    0.000    0.000    0.000    0.000 fractions.py:499(_div)
        2    0.000    0.000    0.000    0.000 abc.py:117(__instancecheck__)
        1    0.000    0.000    0.000    0.000 numbers.py:283(__float__)
      2/1    0.000    0.000    0.000    0.000 abc.py:121(__subclasscheck__)
        1    0.000    0.000    0.000    0.000 statistics.py:209(_coerce)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.iter}
        1    0.000    0.000   21.735   21.735 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 __init__.py:640(update)
        2    0.000    0.000    0.000    0.000 {method 'items' of 'dict' objects}
        1    0.000    0.000    0.000    0.000 {built-in method math.sqrt}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.issubclass}
        1    0.000    0.000    0.000    0.000 fractions.py:193(as_integer_ratio)
```

```
python3 check_std_correctness.py < test4-10to7.txt
577367520.9077069
Sat Apr 20 01:30:23 2024    profiling_stats

         50001136 function calls (50001134 primitive calls) in 19.740 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    6.105    6.105   16.560   16.560 statistics.py:697(_ss)
 20000000    3.665    0.000    3.665    0.000 {method 'as_integer_ratio' of 'numpy.float64'
objects}
        1    3.432    3.432    7.314    7.314 statistics.py:150(_sum)
 20000000    2.700    0.000    6.365    0.000 statistics.py:240(_exact_ratio)
        1    2.167    2.167    2.167    2.167 {built-in method numpy.array}
 10000000    0.656    0.000    0.656    0.000 {method 'get' of 'dict' objects}
        1    0.641    0.641    0.641    0.641 {method 'split' of 'str' objects}
        1    0.175    0.175    3.171    3.171 check_std_correctness.py:13(read_stdin)
        1    0.102    0.102    0.102    0.102 {built-in method _codecs.utf_8_decode}
        1    0.086    0.086    0.188    0.188 {method 'read' of '_io.TextIOWrapper' objects}
        1    0.008    0.008   19.739   19.739 check_std_correctness.py:18(main)
        1    0.001    0.001   19.740   19.740 <string>:1(<module>)
      142    0.000    0.000    0.000    0.000 fractions.py:62(__new__)
       70    0.000    0.000    0.000    0.000 fractions.py:451(_add)
      209    0.000    0.000    0.000    0.000 {built-in method math.gcd}
        2    0.000    0.000    0.001    0.000 {built-in method builtins.sum}
       70    0.000    0.000    0.000    0.000 fractions.py:356(forward)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.102    0.102 codecs.py:319(decode)
       36    0.000    0.000    0.000    0.000 statistics.py:198(<genexpr>)
        1    0.000    0.000   16.560   16.560 statistics.py:725(variance)
```

```
      142    0.000    0.000    0.000    0.000 {built-in method __new__ of type object at
0x61a6c089a9a0}
        1    0.000    0.000   19.740   19.740 {built-in method builtins.exec}
       36    0.000    0.000    0.000    0.000 statistics.py:721(<genexpr>)
      141    0.000    0.000    0.000    0.000 fractions.py:256(numerator)
       72    0.000    0.000    0.000    0.000 {built-in method builtins.isinstance}
      141    0.000    0.000    0.000    0.000 fractions.py:260(denominator)
      2/1    0.000    0.000    0.000    0.000 {built-in method _abc._abc_subclasscheck}
        1    0.000    0.000    0.000    0.000 statistics.py:264(_convert)
        1    0.000    0.000   16.560   16.560 statistics.py:816(stdev)
        2    0.000    0.000    0.000    0.000 {built-in method _abc._abc_instancecheck}
        2    0.000    0.000    0.000    0.000 fractions.py:368(reverse)
       35    0.000    0.000    0.000    0.000 __init__.py:579(__missing__)
        1    0.000    0.000   16.560   16.560 check_std_correctness.py:8(compute_std)
        1    0.000    0.000    0.000    0.000 __init__.py:565(__init__)
        2    0.000    0.000    0.000    0.000 fractions.py:499(_div)
        2    0.000    0.000    0.000    0.000 abc.py:117(__instancecheck__)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.iter}
        1    0.000    0.000    0.000    0.000 numbers.py:283(__float__)
      2/1    0.000    0.000    0.000    0.000 abc.py:121(__subclasscheck__)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        2    0.000    0.000    0.000    0.000 {method 'items' of 'dict' objects}
        1    0.000    0.000    0.000    0.000 statistics.py:209(_coerce)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.000    0.000    0.000    0.000 __init__.py:640(update)
        1    0.000    0.000    0.000    0.000 {built-in method math.sqrt}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.issubclass}
        1    0.000    0.000    0.000    0.000 fractions.py:193(as_integer_ratio)
```

Výsledky:

Považujeme skript *check_std_correctness.py* za přesný výsledek kvůli použití standardizované funkce na výpočet. Tato skutečnost bude zobrazena na grafu zobrazující chybu v přesnosti původního programu.

Kód v check_std_correctness.py byl dále optimalizován pomocí numpy.arr() funkce a vylepšenému čtení stdin.

Testy byly provedeny o velikostech: 10 10^3 10^6 10^7 10^8 číslic

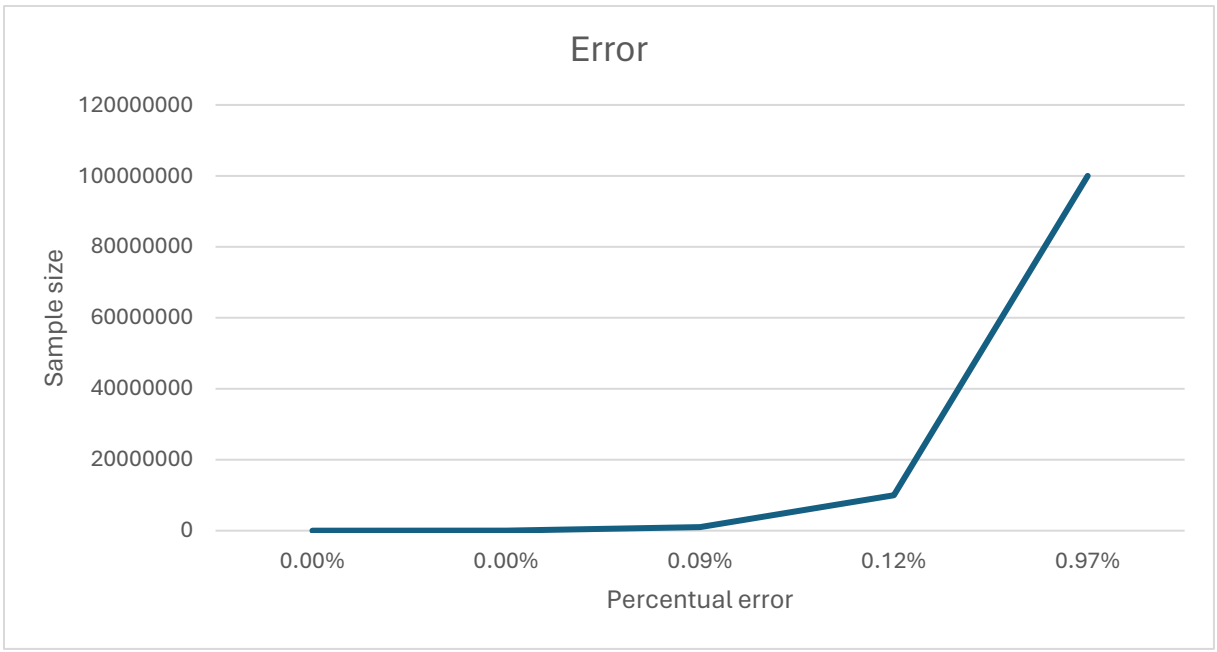
Odchylka:

Výpočet odchylky

$$\delta = \left|\frac{original - control}{original}\right| \cdot 100$$

	Sample size
test1	10
test2	1000
test3	1000000
test4	10000000
test5	100000000

	test1	test2	test3	test4	test5
original decimal	0.6970111	*0.806305*	0.7060071	0.9076962	0.700383
control decimal	0.6970111	*0.8063052*	0.7060004	0.9077069	0.700451
δ	0.00%	0.00%	0.09%	0.12%	0.97%


Error
120000000
100000000
80000000
60000000
40000000
20000000
0
Sample size
0.00%
0.00%
0.09%
0.12%
0.97%
Percentual error

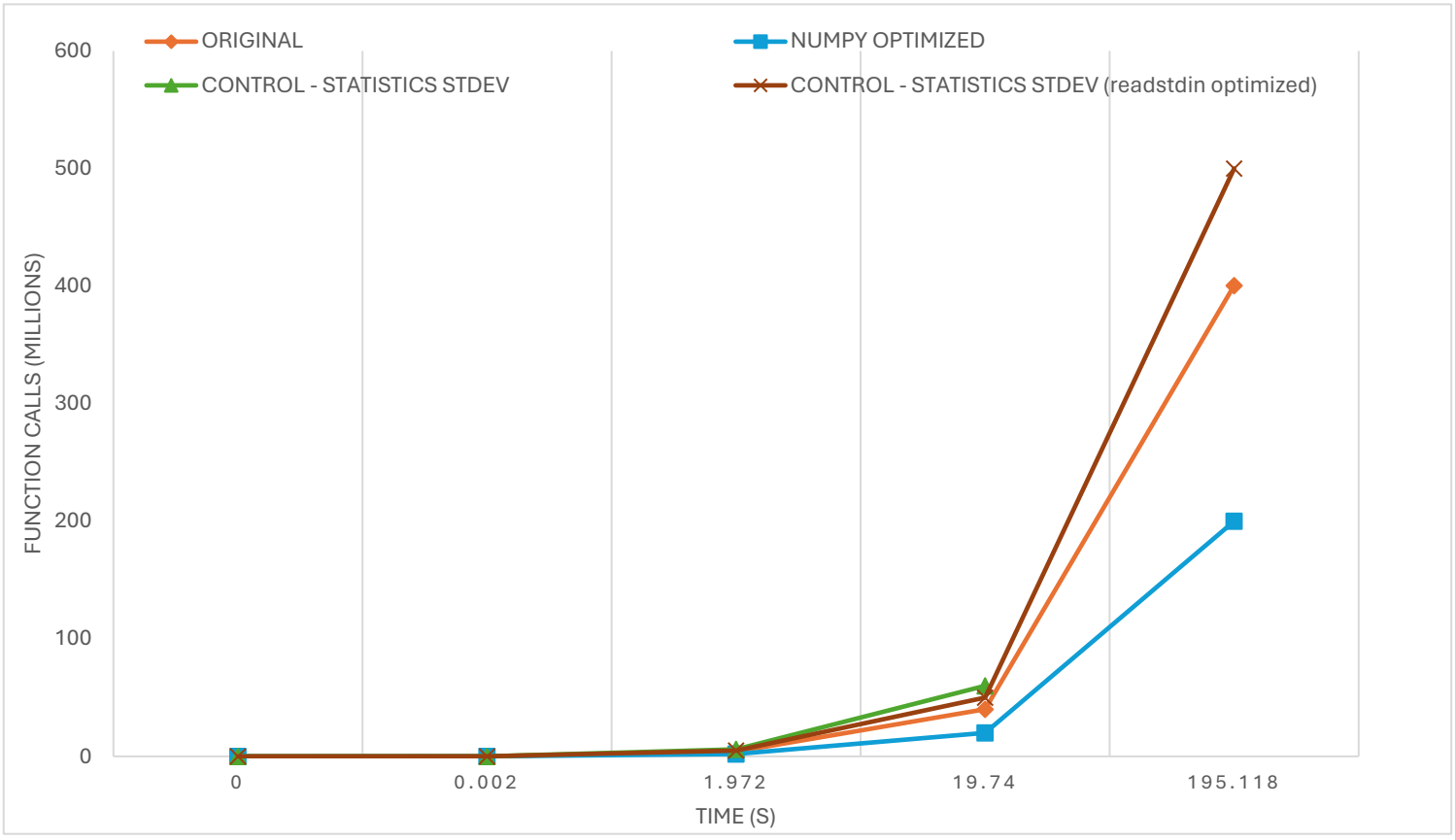
Výsledky profileru:

ORIGINAL					
	test1	**test2**	**test3**	**test4**	**test5**
time (s)	0	0.002	1.13	11.263	111.798
function calls	*60*	4024	*4004576*	*40045586*	400455692
values	563285339.697011	*576589468.806305*	577408887.706007	577367520.907696	577373407.700383

NUMPY OPTIMIZED					
	test1	**test2**	**test3**	**test4**	**test5**
time (s)	0.001	0.001	0.874	8.494	85.871
function calls	*45*	2025	*2000025*	*20000025*	200000025
values	563285339.697011	*576589468.806305*	577408887.706007	577367520.907696	577373407.700383

CONTROL - STATISTICS STDEV					
	test1	**test2**	**test3**	**test4**	**test5**
time (s)	0	0.003	2.116	20.993	
function calls	*239*	6400	*6005572*	*60046703*	
values	563285339.6970110	*576589468.8063050*	577408887.7060000	577367520.9077060	

CONTROL - STATISTICS STDEV (readstdin optimized)					
	test1	**test2**	**test3**	**test4**	**test5**
time (s)	0	0.002	1.972	19.74	195.118
function calls	*228*	5395	*5001015*	*50001136*	500001323
values	563285339.697011	*576589468.806305*	577408887.706000	577367520.907706	577373407.700451



Zjištění a návrh optimalizace:

Náš původní program označovaný *original*, či soubor *standard_deviation.py* nejvíce trpí na neefektivní čtení vstupu ze stdin. To zároveň značí zdvojnásobený počet funkčních volání.

Další dílčí ztráta času nastává při výpočtu sumy a volání funkce *sum_of_squared_differences*.

Použití *numpy.arr* nám pomohlo se zbavit extra for loopu, čímž se výrazně změnčil počet funkčních volání a čas výpočtu.

Za korektní výstup považujeme tabulky označené *CONTROL*, kde voláme funkci *stdev* z vestavěné knihovny *statistics*. Tímto dále určujeme odchylku při různých velikostí vstupů. Komplexnější struktura volání této funkce nám prodlužuje čas výpočtu, ale zároveň dosahuje vyšší přesnosti.

Optimalizace:

Některé optimalizace jsme již zmínili a máme k nim zároveň i data, tak tyto budeme již považovat za probrané.

Použití nativních operací jako +-*/ namísto volání externích funkcí.

Maticové operace pomocí *Numpy arrays*.

Paralelní zpracování např. u funkce *sum_of_squared_differences* by nám rapidně urychlilo výpočet. Použití *python multiprocessing* knihovny.

GPU akcelerace by byla také velmi užitečná pro práci s velkými vstupy. Dosažení tohoto by bylo možné pomocí např. *CuPy* knihovna, která má operace akcelerované pomocí *CUDA* technologie od Nvidie.

Kód

Original (standard_deviation.py):

```python
import cProfile
import pstats
from pstats import SortKey
import sys # for stdin use
from calculatormathlib import CalculatorMathLib # Used for CalculatorMathLib() object


nums = []
def read_stdin():
    for line in sys.stdin:
        # Get rid of whitespaces
        lineVals = line.strip().split()

        # Only consider correct float values from input
        for value in lineVals:
            floatVal = float(value)
            nums.append(floatVal)

# Object for mathlib functions
Math = CalculatorMathLib()

# Calculates the sum of all numbers in an array
def sum(nums):
    res = 0
    for num in nums:
        res = Math.add(res, num)

    return res

# Sums the difference of (num - barNum)^2 for all numbers from the array nums
# barNum is supposed to be the mean
def sum_of_squared_differences(nums, barNum):
    res = 0
    for num in nums:
        powRes = Math.pow2(num - barNum)
        res = Math.add(res, powRes)
    return res


def standard_deviation(nums):
    sumOfNums = sum(nums)
    argsLen = len(nums)     # Accounts for 0eth element correctly

    # barX median also medX
    barX = Math.div(sumOfNums, argsLen)


    # Final equation using s = sqrt(sum_of_squared_differences/argsLen-1)
    S = Math.sqrt(Math.div(sum_of_squared_differences(nums, barX), Math.sub(argsLen, 1)))
    return S


def main():
    # Fills up nums[] array with values
    read_stdin()

    # Calcualates the standard_deviation using the numbers from the nums array
    S = standard_deviation(nums)
    print(S)

if __name__ == "__main__":
    cProfile.run('main()', "profiling_stats")
    p = pstats.Stats("profiling_stats")
    p.strip_dirs().sort_stats('time').print_stats()
```

Numpy Optimized (np_standard_deviation.py):

```
import cProfile
import pstats
from pstats import SortKey
import sys # for stdin use
import numpy as np
from calculatormathlib import CalculatorMathLib # Used for CalculatorMathLib() object


def read_stdin():
    input_data = sys.stdin.read()
    float_vals = np.array(input_data.split(), dtype=np.float64)
    return float_vals


# Object for mathlib functions
Math = CalculatorMathLib()

# Sums the difference of (num - barNum)^2 for all numbers from the array nums
# barNum is supposed to be the mean
def sum_of_squared_differences(nums, barNum):
    res = 0
    for num in nums:
        powRes = Math.pow2(num - barNum)
        res = Math.add(res, powRes)
    return res


def standard_deviation(nums):
    sumOfNums = np.sum(nums)
    argsLen = len(nums)     # Accounts for 0eth element correctly

    # barX median also medX
    barX = Math.div(sumOfNums, argsLen)

    # Final equation using s = sqrt(sum_of_squared_differences/argsLen-1)
    S = Math.sqrt(Math.div(sum_of_squared_differences(nums, barX), Math.sub(argsLen, 1)))
    return S


def main():
    # Fills up nums[] array with values
    nums = read_stdin()

    # Calcualates the standard_deviation using the numbers from the nums array
    S = standard_deviation(nums)
    print(S)

if __name__ == "__main__":
    cProfile.run('main()', "profiling_stats")
    p = pstats.Stats("profiling_stats")
    p.strip_dirs().sort_stats('time').print_stats()
```

Control - STATISTICS STDEV (readstdin optimized) [check_std_correctness.py]:

```
import sys
import statistics
import numpy as np
import cProfile
import pstats
from pstats import SortKey

def compute_std(arr):
    # S = np.std(arr) - CALCULATES POPULATION
    S = statistics.stdev(arr)
    return S

def read_stdin():
    input_data = sys.stdin.read()
    float_vals = np.array(input_data.split(), dtype=np.float64)
    return float_vals

def main():
    # Print the computed standard deviation
    nums = read_stdin()
    print(compute_std(nums))

if __name__ == "__main__":
    cProfile.run('main()', "profiling_stats")
    p = pstats.Stats("profiling_stats")
    p.strip_dirs().sort_stats('time').print_stats()
```

Generátor čísel (generate_nums.py):

```python
import random
import sys

def generate_floats(num_floats, lower_bound, upper_bound):
    return [random.uniform(lower_bound, upper_bound) for _ in range(num_floats)]

# Number of floats to generate
num_floats = 1_000
# Interval bounds
lower_bound = -1_000_000_000
upper_bound = 1_000_000_000

if __name__ == "__main__":
    if len(sys.argv) > 1:
        num_floats = int(sys.argv[1])
    floats = generate_floats(num_floats, lower_bound, upper_bound)
    # Num of decimals_places = 7
    decimals_places = 10
    rounded_floats = [round(fl, decimals_places) for fl in floats]
    print(" ".join(map(str, rounded_floats)))
```