

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IVS – 2. projekt

Profiling: Výstupní zpráva

# Obsah zprávy

Úvod .....	3
Gprof .....	3
Překlad programu .....	3
Profiling.....	3
Výstup .....	3
Callgrind.....	6
Překlad programu .....	6
Profiling.....	6
Výstup .....	6
Interpretace výsledků .....	11
Závěr.....	11
Příloha č. 1 – Skript pro generování vzorků.....	12
Příloha č. 2 – Úplné grafy vygenerované nástroji callgrind a graph2dot.....	13
10 hodnot.....	13
100 hodnot.....	14
1000 hodnot.....	15

# Úvod

V následujícím textu je popsán profiling programu *stddev*, který byl vytvořen v rámci vývoje kalkulačky *dumdumCalc*. Tento program načítá ze standardního vstupu hodnoty (celá nebo desetinná čísla) a následně z nich pomocí funkcí z matematické knihovny *math\_lib* (taktéž využívané v kalkulačce) počítá tzv. [výběrovou směrodatnou odchylku](#), kterou poté tiskne na standardní výstup.

Pro profiling byly využity, pro větší směrodatnost výsledků, hned dva programy – *callgrind* a *gprof*. Program byl profilován třikrát (s 10, 100 a 1000 hodnotami) na platformě Ubuntu 20.04 (64bit).

## Gprof

### Překlad programu

Program *stddev* byl přeložen pro gprof následujícím způsobem:

```
g++ -pg -g -O2 -Werror -Wall -pedantic -std=c++11 -c -o build/stddev.o
stddev.cpp

g++ -pg -g -O2 -Werror -Wall -pedantic -std=c++11 -c -o
build/math_lib.o math_lib.cpp

g++ -pg -g -O2 build/stddev.o build/math_lib.o -lm -o build/stddev
```

Přepínač *-O2* byl přidán za účelem automatické optimalizace (a lepších výsledků).

### Profiling

Samotný profiler byl spuštěn touto sekvencí příkazů:

```
./build/stddev < profiling_sample

gprof -b ./build/stddev
```

Kde soubor *profiling\_sample* byl vzorek s náhodně vygenerovanými 10/100/1000 hodnotami oddělenými náhodným počtem a typem bílých znaků (mezery, tabulátory, nové řádky). Pro účely tvorby vzorků jsme vytvořili jednoduchý skript v jazyce Python (viz [Příloha č. 1](#)).

### Výstup

Po profilování s 10, 100, 1000 byly výstupy programu gprof spojeny do souboru *vystup.txt* a odděleny oddělovači. Pro větší přehlednost zde uvádíme pouze prvních pět řádků Flat profile z každého běhu programu (kompletní výstup je dostupný ve výše zmíněném souboru).

10 hodnot:

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	20	0.00	0.00	add(double, double)
0.00	0.00	0.00	11	0.00	0.00	f_pow(double, double)

```

0.00      0.00      0.00      10      0.00      0.00  void
std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> >::_M_construct<char*>(char*, char*,
std::forward_iterator_tag)

0.00      0.00      0.00       5      0.00      0.00  void
std::vector<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> >, std::allocator<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > >
>::_M_realloc_insert<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> >
const&>(__gnu_cxx::__normal_iterator<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> >*,
std::vector<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> >, std::allocator<std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > > > >,
std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&)

0.00      0.00      0.00       5      0.00      0.00  void
std::vector<double, std::allocator<double>
>::_M_realloc_insert<double>(__gnu_cxx::__normal_iterator<double*,
std::vector<double, std::allocator<double> > >, double&&)

```

## 100 hodnot:

%	cumulative	self		self	total	
time	seconds	seconds	calls	Ts/call	Ts/call	name
0.00	0.00	0.00	200	0.00	0.00	add(double, double)
0.00	0.00	0.00	101	0.00	0.00	f_pow(double, double)
0.00	0.00	0.00	100	0.00	0.00	void
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::_M_construct<char*>(char*, char*, std::forward_iterator_tag)						
0.00	0.00	0.00	8	0.00	0.00	void
std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > >::_M_realloc_insert<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&>(__gnu_cxx::__normal_iterator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >*, std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > > >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)						
0.00	0.00	0.00	8	0.00	0.00	void
std::vector<double, std::allocator<double> >::_M_realloc_insert<double>(__gnu_cxx::__normal_iterator<double*, std::vector<double, std::allocator<double> > >, double&&)						

## 1000 hodnot:

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	2000	0.00	0.00	add(double, double)
0.00	0.00	0.00	1001	0.00	0.00	f_pow(double, double)
0.00	0.00	0.00	1000	0.00	0.00	void
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::_M_construct<char*>(char*, char*, std::forward_iterator_tag)						
0.00	0.00	0.00	11	0.00	0.00	void
std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >>::_M_realloc_insert<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > const&>(__gnu_cxx::__normal_iterator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >*, std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > > >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)						
0.00	0.00	0.00	11	0.00	0.00	void
std::vector<double, std::allocator<double> >::_M_realloc_insert<double>(__gnu_cxx::__normal_iterator<double*, std::vector<double, std::allocator<double> > >, double&&)						

# Callgrind

## Překlad programu

Program *stddev* byl přeložen pro callgrind následujícím způsobem:

```
g++ -O2 -Werror -Wall -pedantic -std=c++11 -c -o build/stddev.o
stddev.cpp

g++ -O2 -Werror -Wall -pedantic -std=c++11 -c -o build/math_lib.o
math_lib.cpp

g++ -O2 build/stddev.o build/math_lib.o -lm -o build/stddev
```

Oproti překladu pro gprof byly odebrány přepínače *-g* a *-pg*, které pro profilování callgrindem nejsou potřeba a naopak byly příčinou občasných chybových hlášení při jeho běhu.

## Profiling

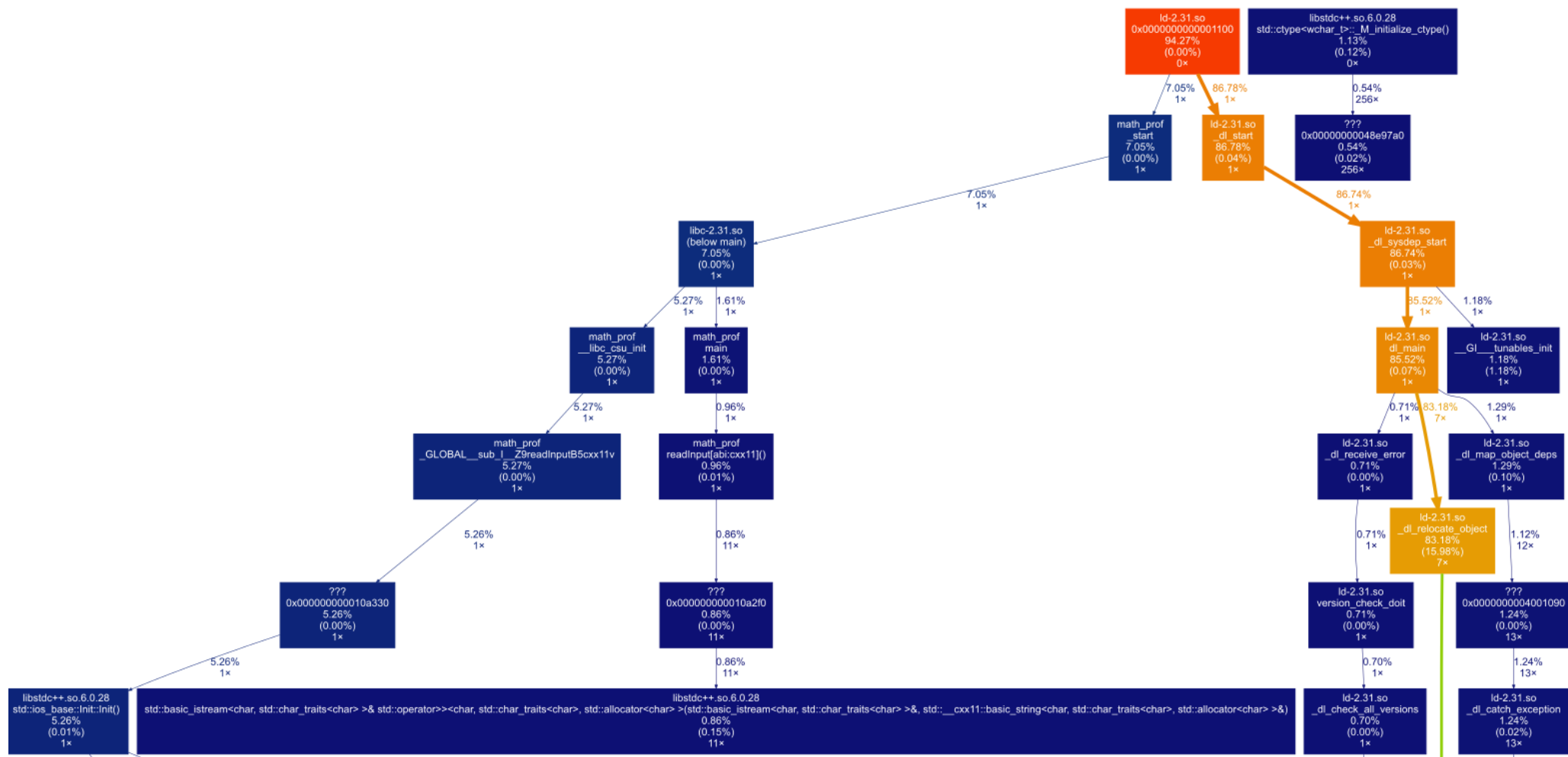
Nástroj callgrind byl po kompilaci spuštěn tímto příkazem:

```
valgrind --tool=callgrind --callgrind-out-file=callgrind-prof.out\
./build/stddev < profiling_sample
```

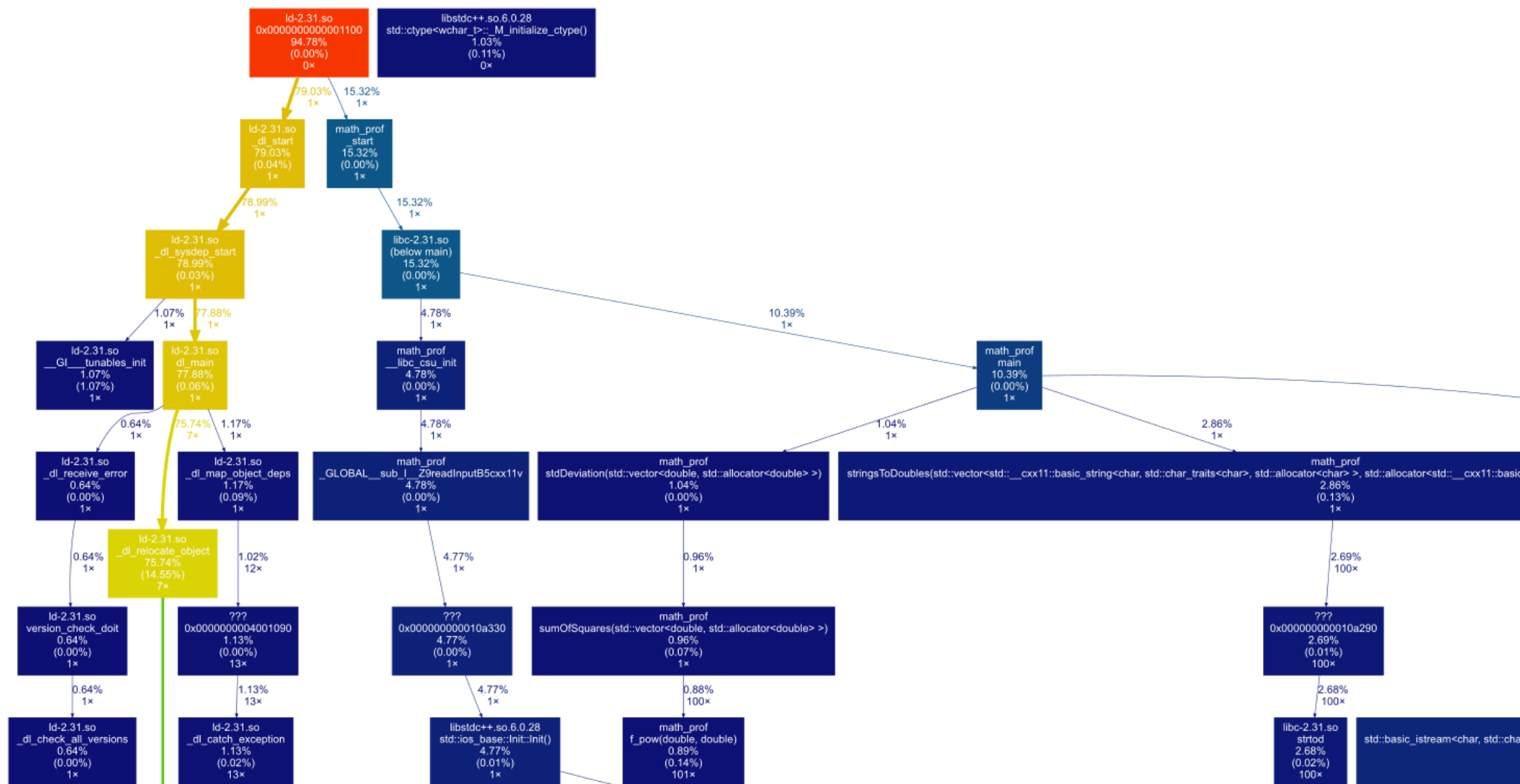
Kde soubor *profiling\_sample* byl vzorek s náhodně vygenerovanými 10/100/1000 hodnotami oddělenými náhodným počtem a typem bílých znaků (mezery, tabulátory, nové řádky).

## Výstup

Výsledné soubory v callgrind formátu byly pro lepší čitelnost následně konvertovány nástrojem *graph2dot.py* do grafické podoby. Pro větší přehlednost uvádíme výstřižky z grafů s kořeny příp. částmi grafu, které zachycují volání matematických funkcí z knihovny (viz následující stránky). Celé grafy je možné si prohlédnout [v příloze](#).

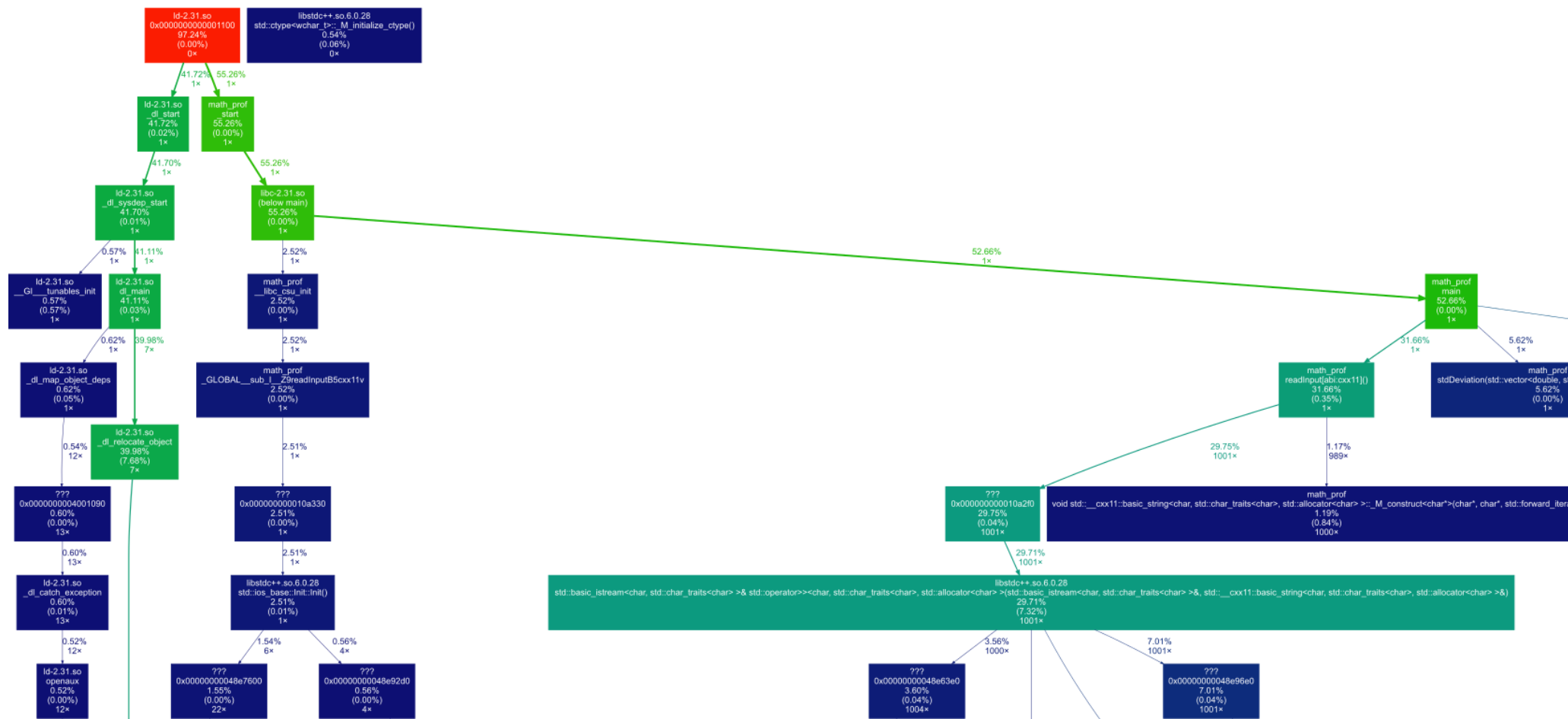


Obrázek 1 - Výsledek profilingu s 10 hodnotami (callgrind)

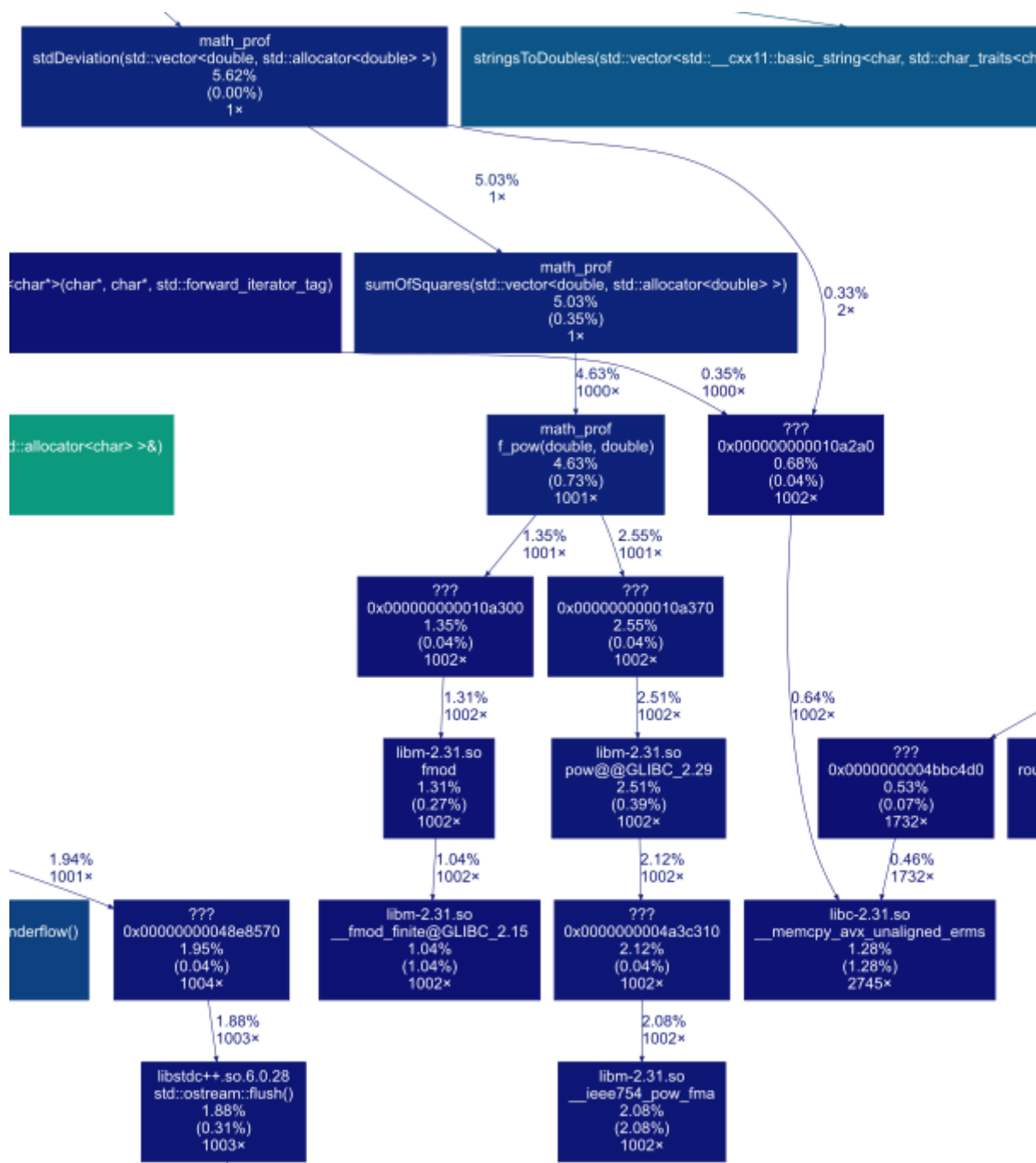


Obrázek 2 - Výsledek profilu se 100 hodnotami (callgrind)





Obrázek 3 - Výsledek profilingu s 1000 hodnotami (callgrind)



Obrázek 4 - Výsledek profilingu s 1000 hodnotami (callgrind) - volání matematických funkcí

## Interpretace výsledků

Vzhledem ke krátkému běhu programu, program gprof nenaměřil žádný čas. Z jeho výstupů však poměrně přehledně vidíme nejčastěji volané funkce.

Nejčastěji byly podle gprofu volány funkce *add* (tj. sčítání dvou reálných čísel) a *f\_pow* (umocňování s přirozeným exponentem) z matematické knihovny. S přibývajícím počtem hodnot roste pochopitelně také počet jejich volání, protože jsou tyto funkce volány pro každou načtenou hodnotu ze stdin. Dále jsou pak často volány funkce, které souvisejí s režii tříd *string* a *vector* (realokace, konstruktory), jež slouží pro uložení načtených hodnot.

Absence časových údajů způsobila nedůvěru v gprof a proto jsme se rozhodli získat podrobnější data prostřednictvím callgrindu. Výsledné grafy ukázaly, že se program velkou část doby svého běhu zabývá vyhledáváním funkcí ve sdílených knihovnách a voláním funkcí s tím spojených (např. *check\_match*). S rostoucím počtem hodnot ve vzorku se však podíl této operace na celkové době běhu snižuje (resp. počet volání zůstává stejný jak pro 10, tak i pro 1000 hodnot), protože se pravděpodobně jedná pouze o jednorázovou akci.

## Závěr

Při optimalizaci se doporučujeme zaměřit na umocňování, které v tomto případě (pouze umocňování druhou mocninou) lze snadno realizovat např. násobením, jež bude zřejmě ke zdrojům šetrnější.

Dále by bylo vhodné se při optimalizaci zabývat třídami sloužícími pro uchování a čtení hodnot (*vector* a *string*). Navrhujeme předávat hodnoty funkcím zásadně prostřednictvím ukazatele (nikoliv skrze hodnoty) nebo např. implementovat vlastní jednodušší struktury. Tyto úpravy by mohly vést ke snížení počtu volaných funkcí (a ke snížení nároků na zásobník).

Nakonec navrhujeme se ujistit, že všechna volání funkcí ze sdílených knihoven (byť standartních) jsou opravdu nutná, popř. počet těchto funkcí zredukovat na minimum. Mohl by se pak snížit čas potřebný k jejich přilinkování pomocí *ld-2.31.so*.

## Příloha č. 1 – Skript pro generování vzorků

Pro generování vzorků využitých při profilování byl využit následující skript v jazyce Python. Přijímá právě jeden argument, tj. počet náhodných hodnot, které mají být vygenerovány, oddělených náhodným počtem a typem bílých znaků.

```
#!/usr/bin/python

import sys
import random

separators = ["\n", " ", "\t"]

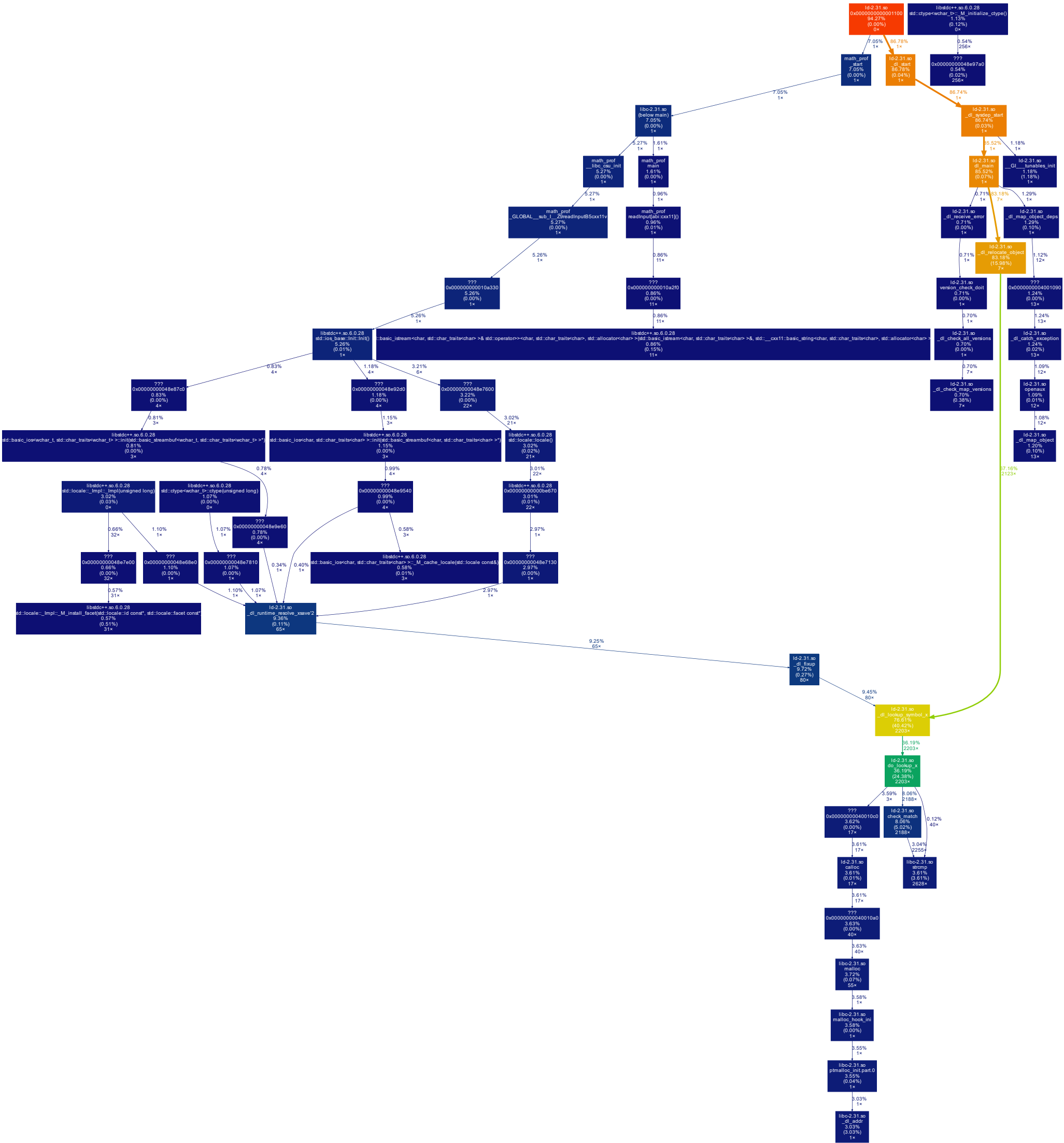
if len(sys.argv) == 1:
    print("Missing argument!")
    quit(1)

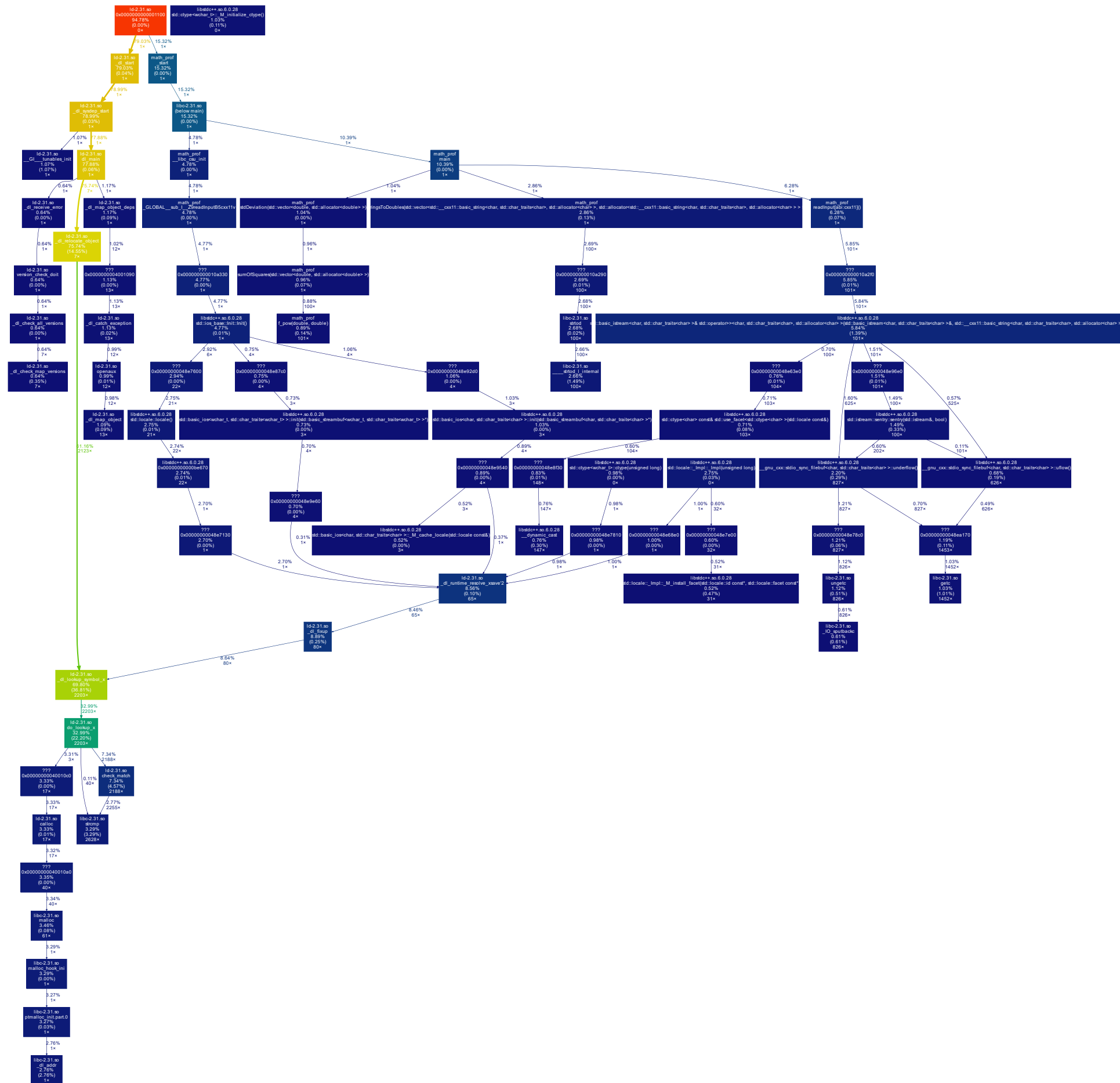
for x in range(int(sys.argv[1])):
    number = random.random()*pow(10, random.randint(0,3))
    number = round(number, random.randint(0,5))
    print(number, end=random.choice(separators))

print()
```

Příloha č. 2 – Úplné grafy vygenerované nástroji callgrind a graph2dot

10 hodnot





**1000 hodnot**

