Vysoké učení technické v Brně
Fakulta informačních technologií

Praktické aspekty vývoje software

2020/2021

Profiling

Přemek Janda (xjanda28)

Zlín, 26. dubna 2021

Výpočet výběrové směrodatné odchylky

Zadání

Pomocí funkcí z Vaší matematické knihovny vytvořte program (jako samostatný spustitelný soubor) pro výpočet výběrové směrodatné odchylky z posloupnosti čísel, kterou program čte ze standardního vstupu (v C např. pomocí funkce scanf) až do konce souboru a musí být schopen načíst min. 1000 čísel. Vstupní soubor obsahuje pouze čísla oddělená bílými znaky (mezera, konec řádku nebo tabulátor) a jejich počet není předem dán. Vzorec pro výběrovou směrodatnou odchylku, který bude využit:

$$s = \sqrt{\frac{1}{N-1}\left(\sum_{i=1}^{N} x_i^2 - N\bar{x}^2\right)}$$

Příklad spuštění

Bez optimalizace

```
php profiling.php 10.txt
php profiling.php 100.txt
php profiling.php 1000.txt
...
php profiling.php < 10.txt
...
```

Optimalizovaná verze

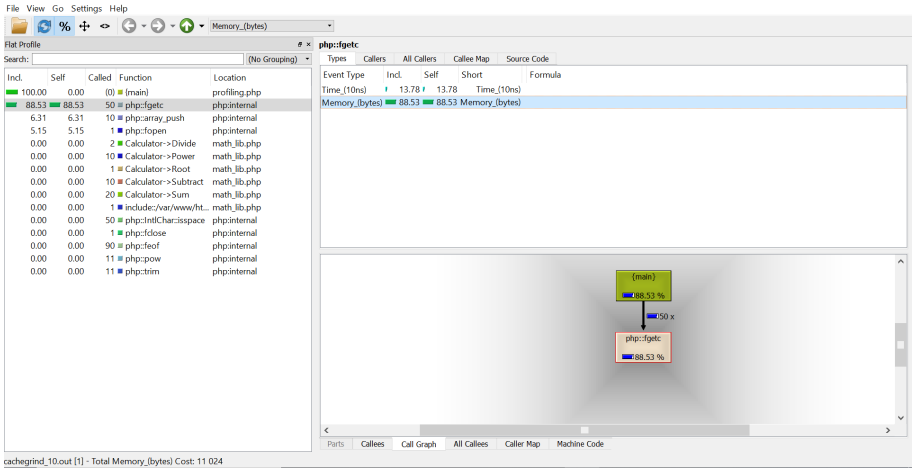
```
php profiling_opt.php < 10.txt
```

Nástroje

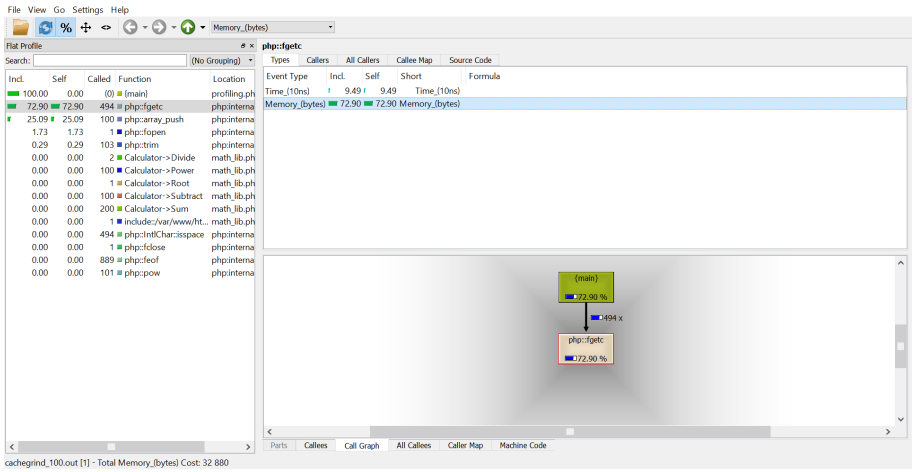
Použitý nástroj pro vygenerovaní souboru s daty z průběhu profilování byl `XDebug` a program pro vizualizaci výsledných údajů byl `WinCacheGrind`. V IDE `Visual Studio Code` byl prostřednictvím terminálu spuštěn program a `XDebug Profiler`, který využívá `Valgrind` a jeho instrumentační nástroj `Callgrind` za běhu programu vygeneroval soubory `Cachgrind_[opt]_[N].out`, ty byly zpracovány a zvizualizovány ve `WinCacheGrind` (viz Přiložené snímky obrazovky).

Výsledky

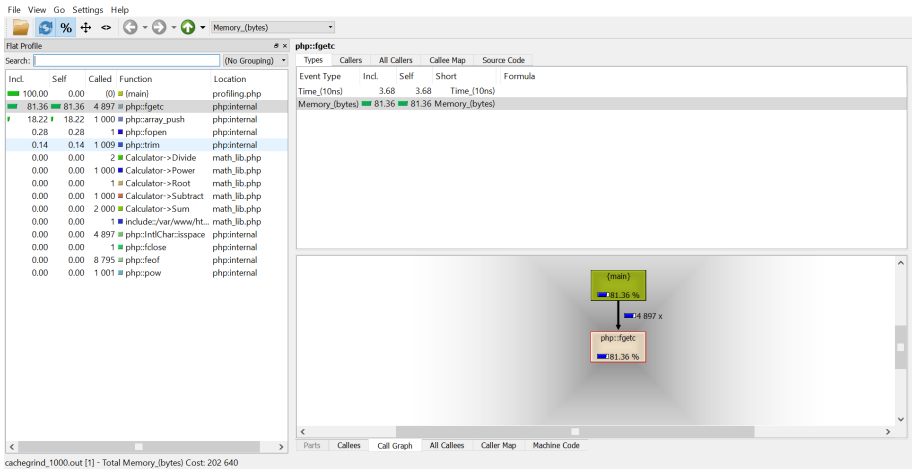
Z výsledů profileru plyne, že nejčastěji volanou funkcí kalkulačky bylo `Sčítání (SUM)`, ktere bylo spuštěno dvojnásobně jak počet načtených číslic. Další v pořadí byly funkce `Odčítání (SUBTRACT)` a `Mocnina (POWER)`, které byly zavolány pro každé číslo právě jednou, neboť byly zanořeny v cyklu foreach a počítaly rozdíl mezi kvadrátem čísla a aritmetického průměru. Dále byly už jen v jednotkách funkce `Odmocnina (ROOT)` a `Dělení (DIVIDE)` pro finální dopočet odbylky. Nezanedbatelnou částí u neoptimalizované verze také bylo vkládání čísel do pole `array_push`. Procesy načítání znaků a jejich ověřování apod. jsem v tomto kontextu zanedebal.



Obrázek 1: Vizualizace profilingu 10 načetených číslic.



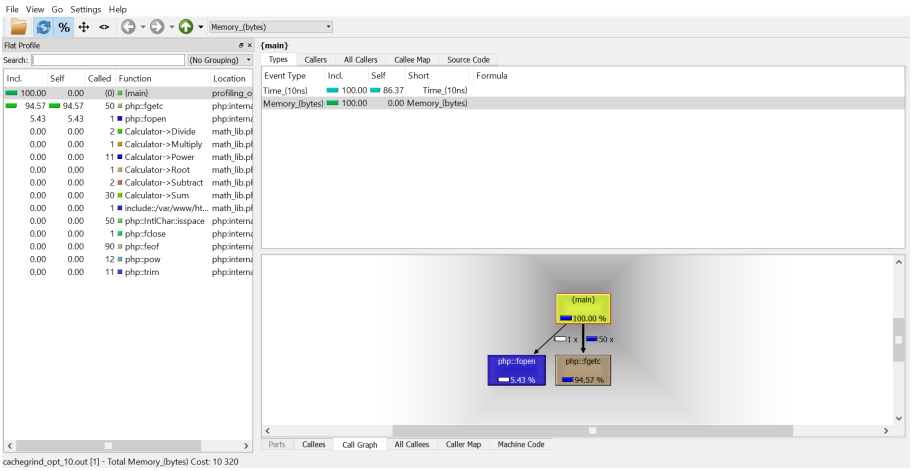
Obrázek 2: Vizualizace profilingu 100 načetených číslic.



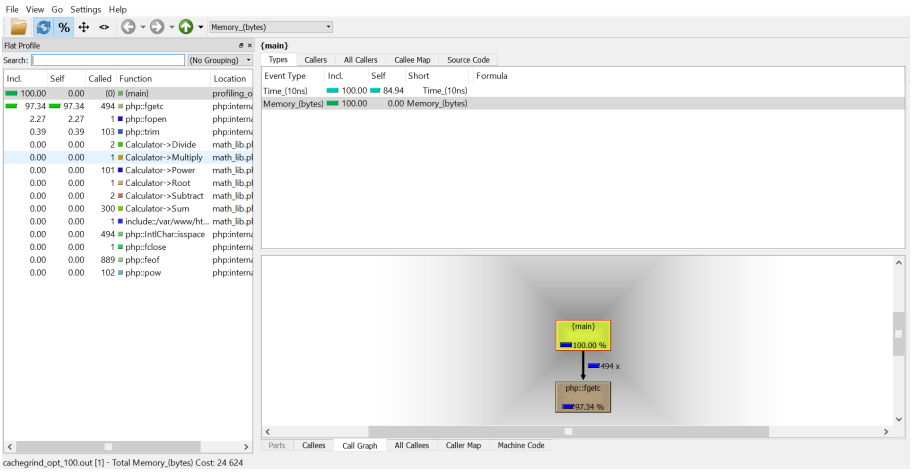
Obrázek 3: Vizualizace profilingu 1000 načetených číslic.

Optimalizace

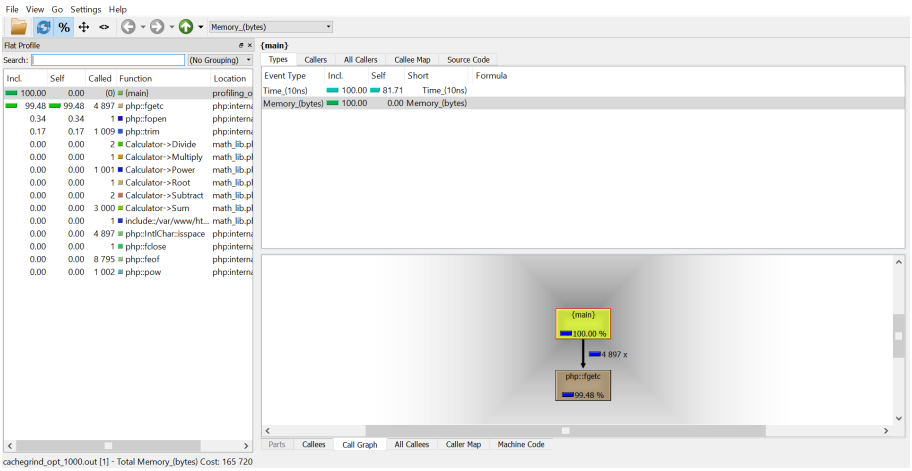
Optimalizace programu spočívala ve snížení počtu cyklů pro procházení polem načtených číslic za účelem vypočtení potřebných mezivýsledků. Původní počet 3 cyklů (2 foreach pro projití celého pole načtených prvků – 1 cyklus pro načtení číslic) jsem zredukoval na 1 cyklus pouze pro načtení dat. Zde jsem zároveň prováděl operace pro výpočet čtverců jednotlivých čísel x_i^2 a aritmetického průměru $\bar{x}$. Současně již nebylo třeba ukládat čísla do pole, čímž se také průběh zrychlil. Výsledkem bylo násobné zrychlení oproti nezoptimalizované verzi. Zajímavým faktem je, že u testování optimalizované verze pro 10 čísel se v grafu objevila funkce `fopen`, která byla v ostatních případech (např. u nezoptimalizovaného programu pro 10 č.) zanedbána pro nedostačující vliv na chod programu. To jest částečným potvrzením zefektivnění použitého algoritmu.



Obrázek 4: Vizualizace profilingu 10 načetených číslic.



Obrázek 5: Vizualizace profilingu 100 načetených číslic.



Obrázek 6: Vizualizace profilingu 1000 načetených číslic.