FIT PEACHES

Výpočet směrodatné odchylky Zpráva z profilingu

Úvod

Protože jsme si zvolili pro vývoj aplikace včetně matematické knihovny, již bylo nutné ve výpočtu směrodatné odchylky využít, JavaScript, nemohli jsme použít doporučený postup pro jazyk C a podobné a museli jsme sáhnout po jiných metodách profilingu.

Pokusili jsme se použít doporučovaný nástroj Clinic.js¹, nicméně při snahách profilovat výpočet směrodatné odchylky o 10, 100 a 1000 vstupech nám hlásil, že běh programu byl tak krátký, že není co analyzovat. Provedli jsme tedy jedno profilování výpočtu směrodatné odchylky o milionu vstupů a jedno o deseti milionech vstupů, více informací dále v kapitole 4.

Protože s tímto nástrojem nebylo možné splnit zadání, museli jsme najít jiné řešení. Po testování různých *profilerů* jsme se rozhodli využít **easy-profiler**².

1 Způsob profilování

Knihovna a kód pro profilování je zanesen přímo v souboru profiling.js. Ať už bude výpočet spouštěn pomocí příkazu node nebo po přeložení klasickým způsobem (např. ./profiling, profiling a výstup profileru je možné zapnout jednoduše přidáním argumentu --profile programu.

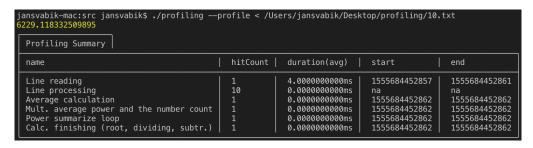
Sami jsme pro dodání čísel pro výpočet využili generátor desetinných číslic. Vždy jsme si tedy nejprve vygenerovali soubory se seznamem číselných vstupů prostřednictvím generátoru. Generovaná čísla byla v intervalu <-10000;10000> s desetinnou přesností na 4 místa.

2 Profilování 10, 100 a 1000 vstupů

Při deseti a sto vstupech se neukazoval žádný zásadní rozdíl v době vykonávání funkcí a cyklů ve výpočtu odchylky.

Na následujících výstupech profileru je patrné, že běh byl velice rychlý. Teoreticky je možné konstatovat, že při takto malém počtu vstupů není ani co profilovat (jak nám řekl i Clinic.js, který jsme chtěli využít pro profiling původně).

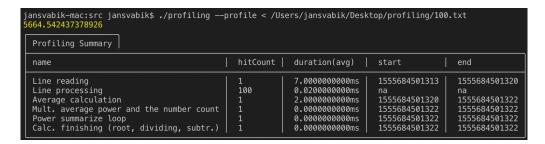
Celkový běh v prvním případě nepřesáhl čtyři milisekundy, ve druhém případě pak nepřesáhl milisekund jedenáct.



Obrázek 1: Profiling vstupu o deseti číslech

¹https://www.npmjs.com/package/clinic

²https://www.npmjs.com/package/easy-profiler



Obrázek 2: Profiling vstupu o sto číslech

Při profilování výpočtu o tisíci vstupech začíná být patrné, že nejvíce času trvá načítání dat. Je možné, že to je způsobeno čtením ze souboru, které je logicky vždy mnohem pomalejší než práce s operační pamětí.

V našem případě samotné načítání trvalo 32 milisekund, což je, dle našeho názoru, poměrně hodně (když vezmeme v potaz, že se načítalo *jen* tisíc čísel).

Celkový běh programu pro výpočet směrodatné odchylky o tisíci vstupech pak trval 41 milisekund. Načítání čísel k výpočtu tedy trvalo 78 % času – to jsou téměř čtyři pětiny času běhu programu.

<pre>jansvabik-mac:src jansvabik\$./profilingprofile < /Users/jansvabik/Desktop/profiling/1000.txt 5708.1850016533635 Profiling Summary</pre>								
name	r	nitCount	duration(avg)	start	end			
Line reading Line processing Average calculation Mult. average power and the number count Power summarize loop Calc. finishing (root, dividing, subtr.)	1 1 1 1 1	1 1000 1 1 1 1	32.0000000000ms 0.00500000000ms 2.00000000000ms 0.00000000000ms 2.00000000000ms 0.00000000000ms	1555684603655 na 1555684603687 1555684603689 1555684603689 1555684603691	1555684603687 na 1555684603689 1555684603689 1555684603691 1555684603691			

Obrázek 3: Profiling vstupu o tisíci číslech

3 Možná řešení

Možností jak urychlit běh programu není mnoho – také proto, že jsme neshledali očividným, že některý z výpočtů (ať už v cyklech nebo mimo ně) trvá znatelně dlouhou dobu (a to ani při milionech vstupů).

Jedinou zjevnou možností se zdá být vyřešit problém s dobou načítání vstupů před zpracováním. Je však možné, že by se dalo některé výpočty vykonávat již při zpracování každého jednotlivého řádku (zejména výpočet druhé mocniny aktuálně načtehé čísla). Protože je JavaScript asynchronní jazyk, měl by se tento výpočet vykonávat paralelně s načítáním vstupů a případnými dalšími výpočty mocniny. Takto by bylo možné běh programu nepatrně urychlit.

Jinou možností by bylo sáhnout po jiném programovacím jazyku – některém nižším, například C nebo C++, který bude pro tyto činnosti vhodnější než námi zvolený JavaScript.

4 Profilování programu s velmi vysokým počtem čísel

Protože nás zajímal i výstup jiného profileru, využili jsme zmiňovaný nástroj Clinic.js. Ten kromě zobrazování latence v rámci zpracovávání cyklů zobrazuje také využití operační paměti v čase a využití procesoru v čase, což jsou další užitečné data potřebná pro zefektivňování algoritmu.

Níže jsou vloženy screenshoty z hlášení profileru. Protože jsme spouštěli profiling několikrát, v jednom případě se nám *podařilo* spatřit i chybovou zprávu, resp. předpokládaný problém v algoritmu (viz Obrázek 7).



Obrázek 4: Profiling vstupu o milionu čísel



Obrázek 5: Profiling vstupu o deseti milionech čísel

Z výsledků profileru je vidět, že procesor je zatěžován, rozdělíme-li běh programu na stejně velké části, průměrně rovnoměrně. Využití operační paměti naproti tomu stále roste, u deseti milionu čísel na vstupu pak znatelně poskočí. Nepodařilo se nám zjistit příčinu, je však možné, že jde o časový úsek po ukončení načítání vstupů.

Z výstupu **easy-profiler**u pro sto tisíc vstupů je pak zjevné, že v načítání je opravdu zásadní problém. Při profilování vstupu se sto tisíci čísly trvalo načítání přibližně 30 sekund – zatímco výpočetním operačím stačilo v součtu 29 milisekund.

Zatížení jen načítáním dat se v tomto případě pohybuje již okolo 98,62~% z celkové doby běhu programu.

jansvabik-mac:src jansvabik\$./profiling —profile < /Users/jansvabik/Desktop/profiling/100000.txt 5769.879054561299								
Profiling Summary								
name	hitCount	duration(avg)	start	end				
Line reading Line processing Average calculation Mult. average power and the number count Power summarize loop Calc. finishing (root, dividing, subtr.)	1 100000 1 1 1 1	28.957sec 0.0037700000ms 6.0000000000ms 0.0000000000ms 23.0000000000ms 0.00000000000ms	1555690087682 na 1555690116640 1555690116646 1555690116670	1555690116639 na 1555690116646 1555690116646 1555690116669 1555690116670				

Obrázek 6: Profiling vstupu o stu tisících čísel

Na obrázku následujícím je pak vidět dříve zmiňované hlášení pravděpodobného problému z Clinic.js.



Obrázek 7: Profiling vstupu o deseti milionech čísel – s hlášením problému

Vojtěch Dvořák Lukáš Gurecký Jan Švábík Radim Zítka