Zpravodaj Československého sdružení uživatelů TeXu

Martin Ruckert

Profilování TeXových dokumentů

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 34 (2024), No. 1-4, 44–58

Persistent URL: http://dml.cz/dmlcz/152650

Profilování TEXových dokumentů

Martin Ruckert

Profiler je nástroj pro analýzu rychlosti kódu. Profiler může poskytovat informace o rychlosti jednotlivých řádků kódu nebo i celých procedur. Tyto informace jsou užitečné při optimalizaci kódu pro dosažení maximální rychlosti.

Ačkoliv jsou profilery běžně dostupné, doposud neexistoval profiler pro programátory, kteří připravují TEXová makra. V tomto článku představuji `texprof` a `texprofile`: dva programy, které společně tvoří profiler pro TEXové dokumenty.

Klíčová slova: TEX, profilování, `texprof`, `texprofile`

1 Kdo potřebuje profiler?

TEXový profiler je nástroj pro programátory, kteří píšou makra pro TEX. To však neznamená, že autor, který příležitostně napíše nějaké makro v TEXu, by měl tento nástroj používat nebo že ho vůbec potřebuje. Optimalizace maker pro rychlost by měla být prováděna pouze tehdy, pokud se makro používá *velmi* často.

Abychom získali lepší představu, co znamená „*velmi* často“, provedeme následující úvahu: Za rozumných předpokladů[1] vede jedna sekunda CPU času k produkci 28 mg CO_2. Opět za rozumných předpokladů[2] způsobí jízda 200 km emise 28 kg CO_2. To znamená, že bychom museli ušetřit miliony sekund CPU času, aby to mělo nějaký podstatný vliv na naši uhlíkovou stopu nebo rozpočet.

Příležitostní autoři TEXových maker mají lepší způsoby, jak trávit čas, než optimalizovat výkon. Existují ale oblíbené makrobalíčky s miliony uživatelů, jejichž makra se spouští mnohokrát za den. Pokud jsme autorem takového balíčku, možná nás zajímá, zda existují příležitosti k optimalizaci, kde tyto příležitosti v kódu hledat a jak velkého zrychlení můžeme docílit.

Možná ještě důležitější je, že profiler nám může říci, kde optimalizaci nehledat a zda změny v kódu přinesly požadovaný efekt. Obecně platí, že bychom nikdy neměli optimalizovat kód pro rychlost bez použití profileru.

2 Jak funguje TEXový profiler?

2.1 Mapování příkazů na vstupní soubory a řádky

TEXový stroj je interpretr, který čte vstupní soubory a vykonává vestavěné příkazy TEXu, jako je například vytvoření horizontálního boxu, zvýšení hodnoty registru

Z anglického originálu [1] přeložil se svolením autora a vydavatele Vít Starý Novotný.

[1] Spotřeba našeho počítače je 200 wattů a na kWh elektřiny připadá emise 500 g CO_2.

[2] Spotřeba našeho vozu je 6 l paliva na 100 km a na litr paliva připadá emise 2370 g CO_2.

 DOI: 10.5300/2024-1-4/44

čítače nebo přidání znaku do aktuálního odstavce. TEXový profiler, nazvaný `texprof`, je TEXový stroj s rozšířeními, která umožňují mapovat každý příkaz na vstupní soubor a na řádek v tomto souboru. K tomuto účelu využívá `texprof` datové struktury, které si každý TEXový stroj (dále jen TEX, pozn. překl.) udržuje, aby mohl zobrazovat užitečná hlášení v případě chyby. Pokud je příkaz součástí formátového souboru, název vstupního souboru a číslo řádku nejsou známy. Dále se podíváme, jak se použití formátového souboru můžeme vyhnout.

I když se vyhneme použití formátového souboru, mnoho příkazů nepochází přímo ze vstupního souboru, ale z výsledku použití maker. Když definujeme makro, TEX uloží příkazy z těla makra do tabulky. Když makro použijeme, TEX načte z tabulky příkazy a vloží je na vstup. Jelikož jsou název vstupního souboru a číslo řádku známy během definice makra, může `texprof` tyto informace také uložit do tabulky a načíst je spolu s příkazy při použití makra. Stejný mechanismus používá `texprof` také v případě, že načítá vstup, například při hledání klíčového slova, a zjistí, že část příkazů musí vrátit na vstup pro pozdější zpracování. Tyto příkazy si s sebou také nesou informace o názvu vstupního souboru a čísle řádku.

Existuje několik vzácných případů, na které se výše uvedený mechanismus nevztahuje. TEX např. vkládá složené závorky kolem výstupní rutiny, aby zajistil, že příkazy provedené ve výstupní rutině nezapříčiní neočekávané globální změny. Z pohledu `texprof`u tyto složené závorky pochází z fiktivního řádku nula ve fiktivním vstupním souboru nazvaném „systém“.

Existují i další fiktivní čísla řádků ve fiktivním vstupním souboru „systém“, která `texprof` používá. TEX vyvolává systémové procedury, jako je rozdělení odstavce na řádky nebo zapsání stránky do souboru DVI, které mohou být časově náročné. Bylo by zavádějící, kdybychom čas strávený v těchto rutinách přiřadili příkazu, který byl proveden na konci odstavce nebo způsobil zapsání stránky do souboru DVI. Proto `texprof` přiřazuje tyto časy k fiktivnímu souboru „systém“ a používá číslo řádku k označení odpovídající systémové procedury TEXu.

2.2 Mapování doby provádění příkazů

Většina příkazů TEXu se vykonává v proceduře `main_control` TEXu. V této proceduře najdeme smyčku, která čte příkazy a poté je vykonává pomocí kódu za návěštím `big_switch` (dále jen „za `big_switch`em“, pozn. překl.). Na začátku každé iterace této smyčky zaznamená `texprof` čas z hardwarových hodin pomocí nízkoúrovňové rutiny operačního systému.

Zaznamenaný čas je začátkem nového časového intervalu a zároveň koncem předchozího časového intervalu. Po zaznamenání počátečního času `texprof` pokračuje v běžném zpracování TEXu a načte další příkaz ze vstupu. Jakmile je příkaz znám, `texprof` zaznamená příkaz, vstupní soubor, číslo řádku (a také makro, ze kterého příkaz pochází; zaměříme se teď ale pouze na příkazy, makra

budou vysvětlena později). Poté pokračuje běžné zpracování a příkaz je vykonán pomocí `big_switch`e, který skončí návratem na začátek smyčky. Tam `texprof` zaznamená koncový čas, vypočítá rozdíl a uloží příkaz, vstupní soubor, číslo řádku a časový rozdíl do velkého pole.

Občas je příkaz v `big_switch`i nahrazen jiným příkazem a následně se `big_switch` použije znovu k jeho vykonání. `texprof` tuto výměnu ignoruje a zaznamená celý čas společně s příkazem, vstupním souborem a číslem řádku, které byly získány na začátku aktuální iterace. To do měření zavádí určitou nepřesnost, ale nezpůsobuje to výrazné chyby.

Mnohem větší vliv na přiřazení času souboru a řádku má existence hlavní smyčky TEXu za návěštím `main_loop`. Kód za návěštím `big_switch` do této smyčky přejde vždy, když narazí na znak, a zůstane v ní, dokud čte příkazy typické pro běžný text: znaky, mezery, kerning, ligatury, změny písma a další podobné věci. Celý čas strávený v této smyčce je poté zaznamenán s příkazem, vstupním souborem a řádkem, které tuto smyčku zahájily. Čas potřebný ke zpracování celého odstavce tak může být přiřazen k prvnímu písmenu tohoto odstavce.

Rozhodnutí nepřiřazovat čas přesněji je v tomto případě odůvodněno následujícími úvahami: Za prvé, takový odstavec je obvykle zpracován pouze jednou a doba zpracování obvykle nepředstavuje významnou část celkového běhu programu. Za druhé, při použití profileru nás obvykle nezajímá čas strávený nad písmeny, mezerami a dalšími částmi běžného textu. Koneckonců žádný autor neoptimalizuje text pro rychlost jeho zpracování. Toto rozhodnutí také výrazně snižuje počet časových intervalů, které musí `texprof` zaznamenávat.

Procedura `main_control` končí vykonáním příkazu `stop`, který zároveň ukončí zaznamenávání časových intervalů. Jako součást závěrečné procedury TEXu `texprof` zapíše všechna sesbíraná data do binárního souboru. Název tohoto souboru je získán připojením přípony `.tprof` k názvu vstupního souboru z vestavěného příkazu `\jobname`. Další zpracování sesbíraných dat neprovádí `texprof`, ale druhý program nazvaný `texprofile`. Jeho použití bude ilustrováno dále.

2.3 Problém měření času

Operační systémy obvykle poskytují několik hodin, ze kterých lze vybírat. `texprof` používá POSIXovou funkci `clock_gettime`, která měří CPU čas aktuálního vlákna v nanosekundách. Ačkoliv nám čas v nanosekundách může připadat jako velmi přesný, skutečná přesnost je spíše v řádu stovek mikrosekund. To má několik důvodů, které si nyní popíšeme.

Moderní počítače provádí mnoho procesů současně a sdílí dostupná CPU mezi prováděnými procesy, přičemž čas potřebný k přepnutí procesů zahrnuje zneplatnění obsahu keše. Instrukce pro načtení dat z hlavní paměti může trvat podstatně déle, pokud se daná paměťová oblast již nenachází v keši. Tento dodatečný čas je poté přiřazen k příkazu TEXu, který se zrovna vykonává.

Moderní CPU také používají techniku zvanou frekvenční škálování pro snížení spotřeby energie. Pokud jen píšeme text v editoru, může náš notebook běžet s frekvencí nižší než 1 GHz. Krátce poté, co spustíte TEX, si operační systém může všimnout, že je potřeba vykonat hodně práce, a může zvýšit frekvenci až na 4 GHz. Všechny příkazy, které se vykonají po této změně, zaberou méně času než stejné příkazy před touto změnou.

V posledních letech začali výrobci kombinovat na jednom čipu několik výkonnostních jader s několika jádry úspornými. Zatímco výkonnostní jádra používají sofistikované superskalární pipeline pro vykonání více instrukcí v jednom cyklu, úsporná jádra používají jednoduché sekvenční provádění instrukcí, které sice spotřebuje podstatně méně energie a produkuje mnohem méně tepla, ale může potřebovat více cyklů na jednu instrukci.

Pokud se operační systém rozhodne přesunout `texprof` z úsporného jádra na výkonnostní jádro uprostřed běhu, bude to mít drastický vliv na časy příkazů. Operační systém ho také může přesunout zpět na úsporné jádro, pokud `texprof` začne provádět vstupně-výstupní operace a musí čekat na disk.

Existuje několik možností, jak tyto efekty zmírnit, například výpočtem průměru přes více běhů nebo použitím syntetických časů. Žádná z těchto metod však není momentálně implementována.

2.4 Mapování výpočetních časů na makra

Když TEX narazí na aktivní znak nebo řídící sekvenci, ví, že musí vykonat primitivní příkaz nebo makro. U makra vyhledá seznam příkazů, které tvoří tělo makra, a tento seznam vloží na svůj zásobník vstupů. Když TEX potřebuje další příkaz ze vstupu, vezme ho z nejvyššího seznamu na zásobníku. Jakmile TEX dosáhne konce tohoto seznamu, odstraní ho ze zásobníku a pokračuje ve čtení příkazů ze seznamu pod ním.

Zásobník vstupů se nepoužívá pouze při použití maker, ale také pro implicitní volání jiných rutin TEXu: Když je připravena nová stránka, konstruktor stránek vloží na zásobník vstupů výstupní rutinu. Na začátku odstavce systém vloží na zásobník příkazy z vestavěného příkazu `\everypar`. Při použití vestavěného příkazu `\input` se na zásobník vloží celé soubory. Je také běžné, že TEX načítá vstup, například kvůli kontrole klíčového slova, a vloží nepoužité tokeny zpět na zásobník, pokud klíčové slovo nebylo nalezeno.

Většinu informací, které `texprof` potřebuje ke sledování maker, lze nalézt na zásobníku vstupů TEXu, ale ne všechny. Zásobník vstupů totiž neobsahuje informace o správné úrovni zanoření maker. Důvodem je chytrá optimalizace, kterou TEX používá na svém zásobníku vstupů pro podporu koncové rekurze: Před vložením těla nového makra na zásobník TEX opakovaně kontroluje, zda je nejvyšší seznam na zásobníku již prázdný. Pokud ano, odstraní prázdný seznam ze zásobníku. Teprve poté, co jsou všechny prázdné seznamy odstraněny, je tělo nového makra vloženo na zásobník.

Díky této optimalizaci může smyčka, která je implementována jako rekurzivní makro provádějící samo sebe jako poslední akci těla makra, běžet bez přetečení zásobníku. Bohužel tato technika odstraní makro ze zásobníku vstupů, zatímco jeho poslední podmakro stále běží, a nové makro se tak vloží na nižší úroveň zanoření, než je jeho skutečná úroveň zanoření.

Proto `texprof` přidává informace o skutečné aktuální úrovni zanoření maker do zásobníku vstupů TEXu a udržuje podpůrný zásobník, který obsahuje informace o všech makrech až po skutečnou úroveň zanoření: název makra, vstupní soubor a číslo řádku, kde bylo makro definováno. Tento podpůrný zásobník poskytuje informace o skutečném začátku a konci provádění makra, které jsou zaznamenávány společně s časovými údaji o vykonaných příkazech.

3 Analýza profilovacích dat

Surová data, která `texprof` zapisuje do výstupního souboru, jsou jen dlouhý seznam tisíců záznamů ve formátu „příkaz, vstupní soubor, řádek, čas“ prokládaných záznamy, které odrážejí změny na zásobníku maker. Extrahování užitečných informací z těchto dat je úkolem samostatného programu nazvaného `texprofile`.

Pro vysvětlení použití `texprof`u a `texprofile`ru je nejlepší použít skutečné příklady. Pro první příklad jsem hledal velký dokument zaměřený na text. Hledáním na internetu jsem našel TEXovou verzi Bible [2]. Několika změnami jsem zajistil, že používá formát plain TEX, takže využívá pouze omezené množství maker. Většina maker pochází z plain TEXu, ale jsou zde také uživatelsky definovaná makra.

Spuštěním příkazu `texprof -prof bible` vznikne soubor `bible.tprof` o velikosti 17 MB. Po spuštění příkazu `tprof bible` bez dalších voleb se zobrazí následující souhrn:

Celkový čas měření:	728,92 ms
Celkový počet vzorků:	2 157 642
Průměrný čas na vzorek:	337,00 ns
Celkový počet souborů:	69
Celkový počet maker:	1 097
Maximální hloubka zásobníku:	7

Pomocí dalších voleb můžeme specifikovat, které datové tabulky má `texprofile` zobrazit a jak by měl informace zobrazovat.

3.1 Deset nejpomalejších řádků

Při použití volby `-T` projde `texprofile` data, sečte časy pro každou kombinaci vstupního souboru a řádku, seřadí výsledky a zobrazí deset nejpomalejších řádků.

soubor	řádek	procenta	čas [ms]	počet	průměr	soubor
systém	shipout	17,68 %	156,05	1 130	138,09 µs	systém
5	29	17,63 %	155,65	54 649	2,85 µs	`bible.tex`
systém	linebrk	15,21 %	134,26	25 777	5,21 µs	systém
systém	buildpg	1,69 %	14,89	55 190	269,00 ns	systém
5	56	0,86 %	7,61	4 750	1,60 µs	`bible.tex`
5	15	0,62 %	5,43	6 183	878,00 ns	`bible.tex`
3	555	0,47 %	4,17	8 549	487,00 ns	`plain.tex`
3	1204	0,28 %	2,44	3 390	719,00 ns	`plain.tex`
3	1201	0,26 %	2,33	2 260	1,03 µs	`plain.tex`
3	1203	0,25 %	2,20	2 258	973,00 ns	`plain.tex`

Tabulka 1: Výstup příkazu `texprofile -T bible`

Výstup příkazu `texprofile -T bible` je zobrazen v Tabulce 1.

První řádek je přiřazen k fiktivnímu souboru „systém“. Záznam ukazuje kumulativní čas důležité systémové rutiny, přičemž číslo řádku identifikuje konkrétní rutinu. Zde jde o zapsání stránky do souboru DVI (shipout), o něco níže pak vidíme rozdělení odstavce na řádky (linebrk) a rozdělení dokumentu na stránky (buildpg). Uvedené časy nezávisí na použití maker, ale pouze na velikosti dokumentu.

Na druhém řádku tabulky vidíme, že řádek 29 v souboru `bible.tex` je zodpovědný za 17,63 % celkové doby běhu programu, a je tedy vhodným kandidátem pro optimalizaci, o kterou se pokusíme v následující sekci. Řádek samotný je poměrně rychlý, v průměru zabere pouze 2,85 µs, ale je používán velmi často: 54 649krát.

To, že zbývajících šest řádků přispívá k celkové době běhu programu méně než 1 %, znamená, že nemusíme uvažovat o jejich optimalizaci.

3.2 Optimalizace makra `\Verse`

Řádek 29 v souboru `bible.tex` definuje makro `\Verse`, vizte Ukázku 1. Makro se používá k vysázení čísla verše zvýšeným textem, vizte Obrázek 1. Toto makro můžeme rychlostně optimalizovat následně: Předpona `\global` není potřeba, protože makro je používáno pouze na globální úrovni. Klíčové slovo `by` je volitelné a můžeme ho vynechat. Každý literál jako `1` je v těle makra uložený jako posloupnost znaků. Tyto znaky jsou při každém vykonání makra opětovně načteny a převedeny na celé číslo. Efektivnější je za tímto účelem použít registry TEXu. Použití matematického módu je zbytečně nákladné pro sazbu zvýšeného textu.

Optimalizovanou verzi makra v novém souboru `bible-opt.tex` můžete vidět v Ukázce 2. Používá dva registry pro potřebné konstanty a používá navíc makro `\leavevmode`, protože vestavěný příkaz `\raise` nelze použít ve vertikálním módu.

```
\def\Verse{\global\advance\vcount by 1${}^{\the\vcount}$}
```

Ukázka 1: Původní definice makra `\Verse`

```
\newcount\1 \1=1 \newdimen\3 \3=3.6pt
\def\Verse{\advance\vcount\1\leavevmode\raise\3\hbox{\sevenrm\the\vcount}}
```

Ukázka 2: Optimalizovaná definice makra `\Verse`

[17]And Moses and Aaron took these men which are expressed by their names: [18]And they assembled all the congregation together on the first day of the second month, and they declared their pedigrees after their families, by the house of their fathers, according to the number of the names, from twenty years old and upward, by their polls.
[19]As the LORD commanded Moses, so he numbered them in the wilderness of Sinai.
[20]And the children of Reuben, Israel's eldest son, by their generations, after their families, by the house of their fathers, according to the number of the names, by their polls, every male from twenty years old and upward, all that were able to go forth to war; [21]Those that were num-

Obrázek 1: Příklady použití makra `\Verse`

soubor	řádek	procenta	čas [ms]	počet	průměr	soubor
systém	shipout	18,35 %	156,29	1 130	138,31 µs	systém
systém	linebrk	15,64 %	133,23	25 777	5,17 µs	systém
5	29	12,85 %	109,48	60 839	1,80 µs	`bible-opt.tex`
3	666	1,95 %	16,61	55 847	297,00 ns	`plain.tex`
systém	buildpg	1,74 %	14,85	55 190	269,00 ns	systém
5	55	0,78 %	6,67	3 552	1,88 µs	`bible-opt.tex`
5	15	0,63 %	5,39	6 183	871,00 ns	`bible-opt.tex`
3	555	0,49 %	4,18	8 549	489,00 ns	`plain.tex`
3	1204	0,29 %	2,43	3 390	716,00 ns	`plain.tex`
3	1201	0,27 %	2,29	2 260	1,01 µs	`plain.tex`

Tabulka 2: Výstup příkazu `texprofile -T bible-opt`

Deset nejpomalejších řádků po optimalizaci je zobrazeno v Tabulce 2. Řádek 29 v `bible.tex` klesl z druhého místa se 17,63 % na třetí místo s pouhými 12,85 %.[3] Ale to není vše: nově se na čtvrtém místě objevil řádek 666 v `plain.tex` s podílem 1,95 %. Celkově jsme dosáhli zrychlení téměř o 3 %, z 17,63 % na 14,80 %.

Pokud chceme zjistit, co způsobilo zvýšené využití plain TEXu, můžeme se podívat na graf volání, který nám může poskytnout další informace.

3.3 Graf volání

Graf volání nám poskytuje informace na vyšší úrovni abstrakce než pouhé sledování deseti nejpomalejších řádků. Zvažme situaci, kdy by odlišné rozložení kódu rozdělilo makro `\Verse` na řádku 29 v souboru `bible.tex`, které jsme analyzovali, do deseti různých řádků. V takovém případě bychom měli deset záznamů s přibližně 1,8 % každý a žádný z nich by se neprobojoval na začátek seznamu.

Pokud chceme být nezávislí na konkrétním rozložení kódu, můžeme se místo řádků zaměřit na makra. Makra poskytují společné jméno pro posloupnost příkazů a obvykle svým názvem označují úlohu, kterou mají příkazy vykonat. Pro splnění úlohy makro obvykle volá další podmakra, která mohou volat další podmakra. Během řetězce volání maker může makro dokonce volat sebe samo, čímž vytvoří rekurzivní smyčku, při které se stane svým vlastním předkem, jak uvidíme níže.

Když se tedy podíváme na běh TEXového programu z pohledu maker, chceme vědět, kolik času jsme strávili v určitém makru včetně všech podmaker, protože to je celkový čas strávený plněním úlohy z názvu makra. Tomuto času říkáme kumulativní čas makra. Dále nás zajímá, jak se kumulativní čas rozděluje na čas, který používá samotné makro, a na čas, který používají jeho podmakra. Všechny tyto informace zjistíme z grafu volání.

Tabulka 3 ukazuje čtyři makra, která zabírají největší procento celkové doby běhu programu. Vidíme, že na prvním místě je makro `\Verse`; vykonávání tohoto makra zabere 174,51 ms, téměř čtvrtinu celkové doby běhu programu. Avšak během 31 011 volání tohoto makra strávíme přímo v makru `\Verse` pouze 101,50 ms, tedy 58,16 %, zatímco zbývajících 73,01 ms strávíme v makru `\leavevmode`.

[3] Změna počtu použití řádku 29 z 54 649 v souboru `bible.tex` na 60 839 v souboru `bible-opt.tex` má následující vysvětlení: Program `texprofile` pro každý spuštěný příkaz zkontroluje, zda pochází z jiného souboru či řádku než příkaz předchozí. Pokud ano, `texprofile` navýší počet pro nový řádek. Několik příkazů z téhož řádku a souboru se tedy počítá pouze jednou.

Když ale TEX v souboru `bible.tex` vstoupí do matematického módu, přejde také z vertikálního do horizontálního módu, což může vyvolat volání dalších maker jako `\everypar` nebo `\output`. V těchto případech se řádek započítá dvakrát. Pokud byl však TEX již v horizontálním módu při vstupu do matematického módu, řádek se započítá pouze jednou. Také když TEX v souboru `bible-opt.tex` volá příkaz `\leavevmode`, vznikne odbočka z `\Verse` k `\leavevmode` (definovanému v souboru `plain.tex`) a pak zpět do `\Verse`, takže řádek se také započítá dvakrát.

Přesnějšího počtu řádků bylo možné dosáhnout sledováním úrovně zanoření na zásobníku maker, aby se rozlišilo, zda vstup na řádek probíhá v rámci volání makra, mimo volání makra, nebo při návratu z volání, kde se úroveň zanoření zmenšuje. Úroveň zanoření se však může zmenšit i v případě, že po m návratech následuje n volání s $m > n$, což nám úkol ztěžuje.

čas	smyčka	procenta	počet/celkem	podmakro
`\Verse`				
174,51 ms		24,64 %	*	`\Verse`
101,50 ms		58,16 %	31 011	`\Verse`
73,01 ms		41,84 %	31 011/31 011	`\leavevmode`
`\output`				
121,22 ms		17,11 %	*	`\output`
1,15 ms		0,95 %	1 130	`\output`
120,07 ms		99,05 %	1 130/1 130	`\plainoutput`
`\plainoutput`				
120,07 ms		16,95 %	*	`\plainoutput`
106,71 ms		88,87 %	1 130	`\plainoutput`
6,99 ms		5,82 %	1 130/1 130	`\makeheadline`
2,76 ms		2,30 %	1 129/1 130	`\pagebody`
2,75 ms		2,29 %	1 130/1 130	`\makefootline`
859,36 µs		0,72 %	1 130/1 130	`\advancepageno`
`\leavevmode`				
73,01 ms		10,31 %	*	`\leavevmode`
20,09 ms		27,52 %	31 011	`\leavevmode`
52,92 ms		72,48 %	495/1 130	`\output`

Tabulka 3: Výstup příkazu `texprofile -G bible-opt`

Z poslední skupiny záznamů v Tabulce 3 vidíme, jak se čas strávený v makru `\leavevmode` rozdělil: Přibližně jedna čtvrtina byla strávena v makru samotném, zatímco zbývající tři čtvrtiny způsobily 495 volání výstupní rutiny `\output`. Celkový počet volání této rutiny je 1 130, což odpovídá počtu stran dokumentu.

Ze záznamů pro registr `\output` a pro makro `\plainoutput` vidíme, že registr `\output` pouze volá makro `\plainoutput`, které vykonává většinu práce a deleguje pouze malou část úloh na vhodně pojmenovaná podmakra.

3.4 Emulace pdfTEXu

Náš druhý příklad je samotný `texprof` – přesněji řečeno, jeho dokumentace. `texprof` je rozšířením TEXu, a protože je TEX implementován jako „literární“ program, je `texprof` implementován jeho rozšířením. Tento literární program lze zpracovat a získat tak jak soubor `texprof.c`, ze kterého může překladač vytvořit spustitelný program, tak soubor `texprof.tex`, ze kterého může TEX nebo pdfTEX vytvořit krásně vysázený dokument.

Vytvoření souboru PDF pomocí pdfTEXu je výrazně pomalejší (2 327 ms) než vytvoření souboru DVI pomocí TEXu (273 ms). Formát PDF je samozřejmě mnohem složitější formát než DVI, což vysvětluje část rozdílů, ale i při deaktivaci vytváření souboru PDF zůstává při použití pdfTEXu značný rozdíl v čase (1 602 ms). Samotný příkaz `texprof -prof texprof` nestačí k nalezení zdroje zpomalení, protože `texprof` vytváří soubor DVI a běží, s přihlédnutím k režii profilování, přibližně stejnou rychlostí (443 ms) jako TEX. Rozdíl v rychlosti je zjevně způsoben makry, která se používají pouze v pdfTEXu.

```
\def\pdftexversion{140}
\def\pdfoutput{1}
\def\pdfdest#1fith{}
\def\pdfendlink{}
\def\pdfannotlink#1goto num#2\Blue#3\Black\pdfendlink{#3}
\def\pdfoutline goto#1 #2 #3{}
\def\pdfcatalog#1{}
```

Ukázka 3: Soubor `fakepdf.tex`, který implementuje zástupná makra pro vestavěné příkazy pdfTEXu

Musíme tedy přinutit `texprof` předstírat, že je pdfTEX. Toho lze dosáhnout tak, že zpracujeme soubor `fakepdf.tex` z Ukázky 3 před zpracováním souboru `texprof.tex`. Příkaz `texprof -prof -jobname=texprof \input fakepdf.tex \input texprof.tex` zabere očekávaných 1 771 ms.

Deset nejpomalejších řádků v Tabulce 4 pak odhaluje, že téměř polovina doby běhu je způsobena čtyřmi řádky (156–159) v souboru `cwebacromac.tex`. Tyto řádky, které vidíte v Ukázce 4, obsahují makra `\addtoks`, `\poptoks` a `\maketoks`.

soubor	řádek	procenta	čas [ms]	počet	průměr	soubor
9	156	20,29 %	522,50	225 137	2,32 µs	`cweb…ac.tex`*
9	157	12,86 %	331,08	140 088	2,36 µs	`cweb…ac.tex`
9	158	9,13 %	235,03	140 938	1,67 µs	`cweb…ac.tex`
9	159	5,88 %	151,27	115 319	1,31 µs	`cweb…ac.tex`
9	172	3,68 %	94,64	15 759	6,00 µs	`cweb…ac.tex`
9	173	3,18 %	81,95	36 954	2,22 µs	`cweb…ac.tex`
systém	shipout	3,16 %	81,27	775	104,87 µs	systém
systém	linebrk	3,14 %	80,93	27 370	2,96 µs	systém
9	152	2,16 %	55,61	67 026	829,00 ns	`cweb…ac.tex`
9	166	1,86 %	47,95	13 042	3,68 µs	`cweb…ac.tex`

*Celé jméno vstupního souboru je `cwebacromac.tex`.

Tabulka 4: Výstup příkazu `texprofile` -T texprof

Pro informace o účelu těchto řádků můžeme konzultovat graf volání. Tabulka 5 ukazuje tři makra, která zabírají největší procento doby běhu. Podívejme se na tato makra jedno po druhém.

3.5 Rekurzivní makra

Na vrcholu je makro `\pdfnote` následované číslem vstupního souboru 7 a číslem řádku 152 v hranatých závorkách. Tyto dodatečné informace jsou poskytovány při použití volby `-i` programu `texprofile` a jsou nezbytné k rozlišení dvou maker, která mají stejný název, jak uvidíme níže. Makro `\pdfnote` vytváří odkazy na různé části dokumentace, je voláno 8 473krát a každé volání vede k volání podmakra `\maketoks`, které je zodpovědné za téměř veškerý čas makra `\pdfnote`.

Každé volání makra `\maketoks` dále deleguje většinu své práce podmakru `\next`. Pokud se podíváme na informace o souboru a řádku pro makra `\maketoks` a `\next`, zjistíme, že obě makra jsou definována ve stejném souboru a na stejném řádku 159. V Ukázce 4 vidíme na řádku 159 definici makra `\maketoks` a na řádku 164 příkaz `\let`, který přiřazuje makru `\next` stejný význam, jaký má makro `\maketoks`. Volání `\next` v řádku 170 ukončuje definici makra `\maketoks`. Ve skutečnosti tedy makro `\maketoks` volá rekurzivně sebe samo. Rekurzivní makro tohoto typu, kde rekurzivní volání probíhá na samotném konci makra, se nazývá koncová rekurze a je optimalizováno tak, aby se při jeho běhu nezvyšoval zásobník vstupů, jak jsme si již vysvětlili v Sekci 2.4.

Makro `\next` rozděluje práci mezi podmakra `\addtokens`, `\poptoks` a několik volání podmakra `\makenote`. 9 271 volání podmakra `\next` v rámci makra `\maketoks` nakonec končí ve 9 271 voláních podmakra `\next` tak, jak je předefinováno na řádku 174, když je nalezena koncová tečka.

čas	smyčka	procenta	počet/celkem	podmakro
`\pdfnote` [7,152]				
1,30 s		61,17 %	*	`\pdfnote` [7,152]
26,54 ms		2,04 %	8 473	`\pdfnote` [7,152]
1,21 s		93,24 %	8 473/9 271	`\maketoks` [7,159]
46,59 ms		3,58 %	24 230/28 824	`\pdflink` [7,24]
14,67 ms		1,13 %	4 507/4 507	`\[` [5,334]
99,89 µs		0,01 %	80/80	`\ETs` [5,177]
54,34 µs		0,00 %	57/57	`\ET` [5,176]
404,00 ns		0,00 %	1/3	`\glob` [4,166]
`\maketoks` [7,159]				
1,24 s		58,35 %	*	`\maketoks` [7,159]
4,67 ms		0,38 %	9 271	`\maketoks` [7,159]
1,21 s		97,76 %	9 271/130 811	`\next` [7,159]
14,59 ms		1,17 %	9 271/140 082	`\poptoks` [7,157]
8,51 ms		0,69 %	9 271/225 136	`\addtokens` [7,156]
`\next` [7,159]				
1,21 s		57,05 %	*	`\next` [7,159]
53,94 ms		4,44 %	130 811	`\next` [7,159]
501,23 ms		41,29 %	142 326/225 136	`\addtokens` [7,156]
462,00 ms		38,06 %	130 811/140 082	`\poptoks` [7,157]
182,12 ms		15,00 %	28 737/28 737	`\makenote` [7,172]
12,19 ms		1,00 %	9 271/9 271	`\next` [7,174]
2,53 ms		0,21 %	1 456/2 254	`\makenote` [7,154]
0,00 ns	1,19 s	0,00 %	121 540/130 811	`\next` [7,159]

Tabulka 5: Výstup příkazu `texprofile -G -i texprof`

```
156 \def\addtokens#1#2{\edef\addtoks{\noexpand#1={\the#1#2}}\addtoks}
157 \def\poptoks#1#2|ENDTOKS|{\let\first=#1\toksD={#1}%
158   \ifcat\noexpand\first0\countB= #1\else\countB=0\fi\toksA={#2}}
159 \def\maketoks{\expandafter\poptoks\the\toksA|ENDTOKS|%
160   \ifnum\countB> 9 \countB=0 \fi
161   \ifnum\countB< 0
162     \ifnum0=\countC\else\makenote\fi
163     \ifx\first.\let\next=\maketoksdone\else
164        \let\next=\maketoks
165        \addtokens\toksB{\the\toksD}
166        \ifx\first,\addtokens\toksB{\space}\fi
167     \fi
168   \else \addtokens\toksC{\the\toksD}%
        \global\countC=1\let\next=\maketoks
169   \fi
170   \next
171 }
```

Ukázka 4: Řádky 156 až 171 v souboru `cwebacromac.tex`

Protože makro `\next` volá samo sebe, je současně svým vlastním potomkem i předkem. To má důsledky pro přiřazení kumulativních časů, jak je zobrazeno v grafu volání. První řádek v tabulce pro makro `\next` ukazuje celkový čas strávený v tomto makru, který činí 1,21 sekundy. Následující řádky rozdělují tyto 1,21 sekundy, což znamená, že časy uvedené v prvním sloupci by měly dohromady činit 1,21 sekundy a procentuální hodnoty by měly dohromady činit 100 %.

Pokud by program `texprofile` jednoduše určil pro každé podmakro začátek a konec a sečetl časové rozdíly, hodnoty pro makro `\next` jako potomka sebe samého by činily 1,19 sekundy, jak je uvedeno v sloupci „smyčka“. Když se však potomek makra `\next` vrátí, je tento čas již zahrnut na předchozích řádcích.

Proto program `texprofile` udržuje pro každého potomka dva akumulátory pro uplynulý čas: Pro čas zobrazený ve sloupci „smyčka“ `texprofile` sečte časové rozdíly pozorované při návratu z podmakra. Pro čas zobrazený ve sloupci „čas“ `texprofile` navíc odečte všechny časové rozdíly, které již byly přičteny k samotnému makru nebo k některému z ostatních podmaker, protože tyto rozdíly již byly zohledněny.

V jednoduché smyčce, jako je tato, je celý čas v makru `\next` jako potomkovi již zohledněn v makru `\next` jako rodiči. Proto sloupec „čas“ ukazuje 0,00 ns. U složitějších rekurzivních smyček tomu tak nemusí být vždy.

Tato sekce na příkladu souboru `cwebacromac.tex` demonstruje způsob, jakým `texprofile` pracuje s rekurzivními makry, ale nezodpovídá uspokojivě otázku, proč jsou makra `\addtoks`, `\poptoks` a `\maketoks` tak pomalá. Tuto otázku se chystá autor zodpovědět v článku „Efficient input file processing with TEX“, který má vyjít v čísle 46:1 časopisu *TUGboat*. (Pozn. red.)

4 Volby a vestavěné příkazy

Volby `texprofu` jsou stejné jako u ostatních TEXových strojů. Jedinou výjimkou je volba `-prof`, která zapne profilování již od začátku dokumentu. Pokud chceme profilovat jen vybrané části souboru, můžeme použít vestavěné příkazy `\profileon` a `\profileoff`.

Volby `texprofileru` se řídí obecným pravidlem, že volby, které vybírají datové tabulky, používají velká písmena, zatímco volby, které mění způsob zobrazení dat, používají malá písmena. Již jsme viděli volbu `-T` pro tabulku deseti nejpomalejších řádků, volbu `-G` pro graf volání a volbu `-i`, která doplňuje makra o čísla vstupních souborů a řádků.

Můžeme zobrazit kumulativní časy podle vstupních souborů volbou `-F`, podle vstupních řádků s volbou `-L` nebo podle TEXových příkazů s volbou `-C`. Volba `-R` pak zobrazí surové časy pro každý příkaz, který byl profilován. Výsledná tabulka může být velká, ale je užitečná, pokud bylo profilování zapnuto pouze na krátkou dobu nebo pokud chceme tabulku dále zpracovávat.

Pokud je tabulka určena pro další zpracování, volba `-m` upřednostňuje strojovou čitelnost před čitelností pro lidi. Zatímco časy jsou standardně zobrazovány v odpovídajících jednotkách (sekundy: `s`, milisekundy: `ms`, mikrosekundy: `us` a nanosekundy: `ns`), volba `-m` zobrazí všechny časy v nanosekundách bez jednotek.

Volba `-p`n potlačí v tabulkách řádky, které spadají pod n procent. Volba `-t`n upraví volbu `-T` tak, aby zobrazila n nejpomalejších řádků. Volba `-s` upraví volbu `-R` tak, aby do tabulky zahrnula informace o změnách na zásobníku maker.

5 Nedostatky, dočasná řešení a budoucí práce

Aktuální verze `texprofu` a `texprofileru` jsou dostupné na adrese `https://github.com/ruckertm/HINT`. Od první prezentace na konferenci TUG 2024 v Praze jsem provedl u obou programů několik vylepšení. Nejvýznamnější z nich je, že formátové soubory nyní obsahují informace o souborech a číslech řádků pro všechny zdrojové soubory použité při vytváření formátu. Připsání doby běhu neznámému souboru by proto mělo být nyní jen výjimečnou situací.

Metoda uvedená v Sekci 3.4, která umožňuje, aby `texprof` expandoval makra jako pdfTEX, je dočasné řešení. Jak je vidět v Ukázce 3, zástupné definice pro pdfTEXová primitiva odpovídají konkrétnímu použití těchto primitiv v daných souborech. Vestavěné příkazy jako `\pdfannotlink` mají složitou syntaxi, kterou je snadné implementovat jako vestavěné příkazy TEXového stroje, ale velmi obtížně pomocí TEXových maker. Zde je potřeba další práce, ať už ve zjednodušení implementace maker se žádanou syntaxí, nebo v přidání nové volby pro `texprofu`, která by zpřístupnila potřebné zástupné definice jako vestavěné příkazy.

Odkazy

1. RUCKERT, Martin. Profiling TEX Input Files. *TUGboat*. 2024, **45**(2), 203–210. Dostupné z DOI: 10.47397/tb/45-2/tb140ruckert-profiler.
2. ALLRED, Sean. *Various translations of the Bible, typeset in TEX* [online]. [cit. 2024-08-23]. Dostupné z: https://github.com/vermiculus/bible.

Summary: Profiling TEX Input Files

A profiler is a tool used by programmers to analyze the run time behavior of the code they write. The profiler can map the CPU time of a program to specific files and lines, or it can map the time to individual procedures. This information is necessary if a programmer wants to optimize the code for speed.

No such tool was so far available to programmers who write macro packages for TEX. This paper presents `texprof` and `texprofile`, two programs working together as a profiler for TEX input files.

Keywords: TEX, profiling, `texprof`, `texprofile`

Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München
Germany
martin.ruckert@hm.edu