Semestrální projekt MI-PRC ZS2019/2020:

GPU Cryptohash Recovery

Pavlína Kopecká, Libor Kuchař

magisterské studium, FIT CVUT, Thákurova 9, 160 00 Praha 6

Prosinec 3, 2019

Definice problému

Mnoho aplikací (zejména těch webových) si musí nějakým způsobem ukládat hesla svých uživatelů. V současné době stále existují weby, které ukládají hesla jako plaintext. Toto řešení je nebezpečné, protože pokud se útočník nějakým způsobem dostane k databázi hesel, tak mu už nic nebrání se dostat i k samotným heslům. Proto je doporučeno hesla zahashovat pomocí silné kryptografické hashovací funkce a následně tuto hodnotu uložit namísto hesla.

Bezpečná hashovací funkce by měla splňovat následující tři body:

1. **Odolnost vůči získání předlohy** – hashovací funkce je jednosměrná.
2. **Odolnost vůči získání jiné předlohy** – hashovací funkce má omezený obor hodnot, existuje tedy více předloh, který mají stejný výsledný hash.
3. **Odolnost vůči nalezení kolize:** Je obtížné systematicky najít dvojici vstupů (x,y), pro které h(x)=h(y).

Jak je patrné z 1. bodu, nelze z hashe systematicky získat původní hodnotu. Jediný způsob, jak získat původní hodnotu, je nějakým způsobem „odhadnout“ tuto hodnotu a zahashovat jí pomocí použité kryptografické funkce. Následně tyto hodnoty porovnat a tím zjistit zda náš odhad byl správný. Jakým způsobem ale původní heslo odhadnout?

1. **Útok hrubou silou** – jediné 100% spolehlivé řešení, vyzkouší se všechny možné kombinace hesel v dané délce. Počet kombinací je ale $\mathbf{a}^{\mathbf{L}}$ kde **a** je délka abecedy (pro malá + velká písmena a čísla je **a** 62) a **L** je délka hesla. Od určité délky není časově možné heslo prolomit.
2. **Slovníkový útok** – používají se hesla ze slovníku. Problém tohoto typu útoku spočívá v tom, že pokrývá pouze malé množství všech možných hesel. Na druhou stranu velká část uživatelů používá velmi slabá hesla, na která tento typ útoku může postačovat. V dnešní době existují už komplexní slovníky obsahující nejčastější hesla.
3. **Rozšířený slovníkový útok** – tato metoda využívá jako základ slovníkový útok, ale slova z tohoto slovníku nějakým způsobem upravuje (např. zaměňuje velikost písmen, přehazuje podobná písmena, přidává před/za/do slova řetězce různé délky atd.)
4. **Duhové tabulky**

Všechny zmíněné útoky lze provádět pomocí CPU (např. slovníkový útok nemá cenu vůbec provádět na GPU z důvodu, že bottleneck je stejně čtení z disku), v případě delších komplexnějších hesel ale CPU již selhává. Tato práce se tedy zabývá nástrojem na získávání hesel z MD5 hashů za použití CUDA, konkrétně se jedná o **útok hrubou silou** a **rozšířený slovníkový útok.**

Popis sekvenčního řešení

Hashovací funkce MD5 byla převzata a mírně upravena z https://gist.github.com/creationix/4710780.

Útok hrubou silou

Největším problémem tohoto útoku bylo samotné generování všech řetězců dané délky. Původně byl tento typ útoku řešen rekurzí, kvůli přenositelnosti na GPU bylo nutné tento algoritmus upravit na iterativní.

Iterativní algoritmus funguje následovně:

1. K řetězci délky L, který obsahuje znaky z abecedy délky A se budeme chovat jako k číslu o počtu číslicíc L a základu A. K tomuto číslu je možné navrhnout softwarovou sčítačkou a pomocí této sčítačky můžeme postupně proiterovat všechny řetězce o délce L. V případě že dojde k přenosu z nejvyššího řádu (poslední znak se vrátí opět na první) tak víme, že jsme proiterovali všechny možnosti.
2. Nastavíme hodnoty sčítačky na počáteční hodnotu (v sekvenčním algoritmu to jsou samé 0).
3. Tento řetězec zahashujeme pomocí funkce MD5.
4. Porovnáme výsledný hash se zadaným hashem. Pokud se shoduje, vrátíme daný řetězec = jedná se o původní hodnotu a pokračujeme ke kroku 7), jinak pokračujeme na krok 5)
5. Inkrementujeme řetězec o 1 (přičteme k nejnižší číslici číslo 1).
6. Pokud nastane přenos z nejvyššího řádu tak byly vyzkoušeny všechny kombinace – pokračujeme krokem 7, jinak pokračujeme na krok 3)
7. V případě nalezení dané hodnoty jí vypíšeme, jinak vypíšeme „No matches“, uvolní se prostředky, program se ukončí.

Rozšířený slovníkový útok

Finální verze programu obsahuje pouze omezený rozšířený slovníkový útok, podporuje pouze přidávání řetězců různé délky za dané slovo. Tento typ útoku slouží spíše pro ilustraci zrychlení na GPU.

Sekvenční řešení vždy přečte jeden řetězec ze slovníku, následně tento řetězec zahashuje pomocí funkce MD5. Tento hash se poté porovná se zadaným hashem. Pokud se hash shoduje tak tento řetězec je původní heslo. Pokud se hash neshoduje, tak se za daný řetězec pomocí bruteforce funkce přidávají všechny řetězce dané délky a abecedy. Tyto rozšířené řetězce se taktéž hashují pomocí funkce MD5 a následně porovnávají se zadaným hashem.

Popis řešení pro GPU (CUDA)

Útok hrubou silou

Nejprve **host** nakopíruje používanou abecedu a zadaný hash do paměti konstant GPU a připraví paměť pro uložení nalezeného řetězce. Poté se spustí **kernel** s N bloky a M vlákny (obě konstanty jsou nastavitelné během spuštění programu). Kernel se spouští zvlášť pro řetězec každé délky (takže např. pokud zkoušíme hesla v rozsahu 1-3 znaků, tak se nejprve pustí kernel pro řetězce délky 1, pak 2 atd.). Po spuštění kernelu se čeká na jeho dokončení a následně host zkontroluje, jestli se podařilo nalézt původní řetězec. Pokud ano, tak se vypíše a program končí. Pokud ne, pokračuje se s dalším rozsahem. Pokud se vyčerpaly všechny možnosti, tak program končí hláškou „No matches“.

Poté se na **device** pro každé vlákno zjistí, od jaké počáteční konfigurace sčítačky má začít počítat. Pro ošetření, aby se projely všechny kombinace se používá zaokrouhlování přídělu práce směrem nahoru, je tedy možné, že některá vlákna vykonávají stejnou práci. Začátek práce je určeno 64-bitovým číslem (bylo by možné rozšířit i na 128, či 256-bitové číslo), počáteční permutace se zjistí tak, že se toto číslo postupně modulí a dělí (stejný princip jako bychom převáděli třeba desítkové číslo na dvojkové).

Poté se zavolá funkce podobná sekvenční funkci, ovšem s jiným počátečním nastavením sčítačky než samé 0. Tato funkce také vždy po určitém počtu iterací (podle nastavení THRESHOLD) kontroluje, jestli nějaké z vláken již původní hodnotu nenašlo (pokud ano skončí práci).

V případě nalezení správného hesla (hashe se rovnají) se toto heslo zapíše do paměti, kterou připravil host. Vzhledem k nízké pravděpodobnosti, že dvě vlákna naleznou řetězec se stejným otiskem se vůbec neřeší atomické operace.

Rozšířený slovníkový útok

Vzhledem k libovolné velikosti slovníku (teoreticky může být velký i několik TB) a předem neznámé velikosti RAM a VRAM program tento slovník „porcuje“ po blocích určité délky. Načítání slov na hostovi probíhá ve dvou fázích. V první fázi se zjišťuje, jak dlouhé je nejdelší slovo v dané skupině slov a kolik slov bude vlastně načteno. Všechna slova budou muset být zarovnána na délku nejdelšího slova (aby bylo možné se stringy smysluplně pracovat). Počet slov se určí v závislosti na nastavení GRANULARITY (určuje na kolik slov bude zarovnávat, např. pokud je GRANULARITY 100 a při 565. slovu nebude stačit paměť načte se pouze 500 slov) a MEMORY_RATIO, který určuje, jak velkou část paměti VRAM využít na ukládání slov. Maximální délka slova je omezena konstantou MAX_WORD_LENGTH.

Po získání počtu slov v dané iteraci se následně tato slova načtou do paměti hostitele, vzhledem k nemožnosti použití strlen na device se tato informace ukládá na poslední bajt daného slova (vždy je alokováno max(strlen) + 2, jeden bajt pro null byte a druhý právě pro uložení délky aktuálního slova). Vzhledem k datovému rozsahu byte (resp. unsigned char) pro uložení maximální délky slova tato implementace nepočítá se delšími slovy, než je 255 znaků.

Před samotným spuštěním kernelu je ještě do paměti symbolů nakopírován hash, slovníky a pravidla pro rozšířený slovníkový útok. Taktéž se připraví paměť na uložení nalezeného řetězce, který má stejný hash jako hledaný hash. Poté se nakopírují slova ze slovníku do paměti VRAM, slova jsou uložena v 1D poli a každé slovo je zarovnané na délku nejdelšího slova + 2. Poté se konečně zavolá kernel, během vykonávání kernelu se připraví do paměti hosta další slova ze slovníku.

Na straně device se podle bloku a vlákna získá oblast paměti nad kterou má dané vlákno pracovat, poté je podobný postup jako v sekvenční variantě s tím rozdílem, že vždy po konstantním počtu

vyzkoušených slov (určený v DICTIONARY_THRESHOLD) se kontroluje, jestli již náhodou nebyl řetězec s odpovídajícím hashem nalezen.

Měření

Měření bylo provedeno na dvou strojích. V následujících tabulkách je shrnutí.

Sestava 1 – Windows	
Procesor	AMD Ryzen 5 1600 **@3.4GHz**
Grafická karta	MSI GeForce GTX 1080 SEA HAWK X (GDDR5X 10108MHz) GTX1080 (1708MHz)
Paměť	RAM: G.SKILL 16GB KIT DDR4 3200MHz CL14 Flare X for AMD
Pevný disk (umístění slovníku)	Seagate BarraCuda 2TB 7200RPM
Operačný systém	Windows 10 Professional
Poznámka	Úplná optimalizace (/Ox)

Sestava 2 – STAR	
Procesor	2ks 6core Xeon 2620 v2 @ 2.1Ghz
Grafická karta	GeForce RTX 2080 Ti
Paměť	32 GB
Operačný systém	CentOS Linux 7 (Core)
Poznámka	bez optimalizace

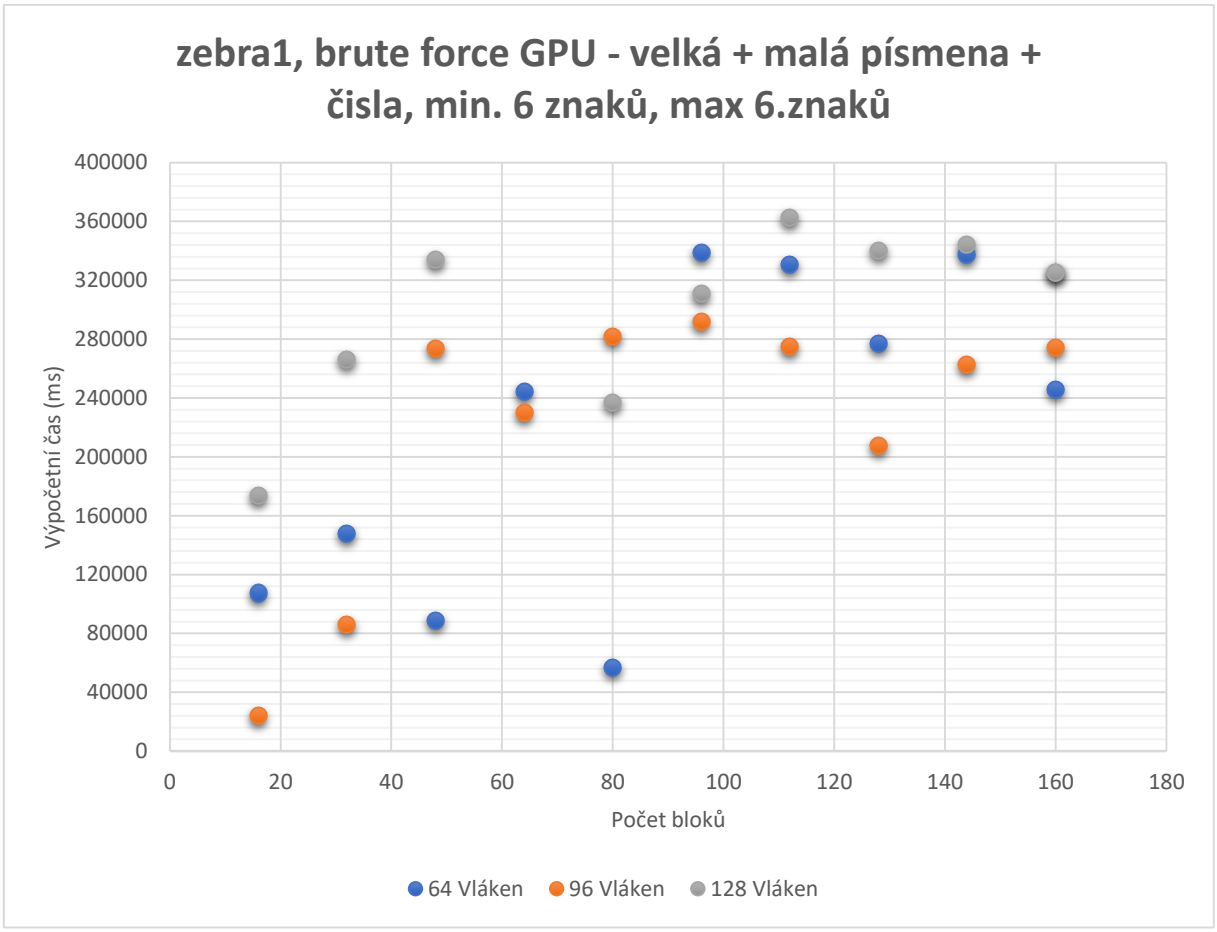
Parametry blocks a threads

Tyto dva parametry mají na výpočetní čas obrovský vliv. Správné nastavení těchto parametrů dokáže chod programu značně urychlit, naopak špatně zvolené parametry délku výpočtu zvyšují. Bohužel měření ukazují, že neexistuje univerzální nastavení těchto parametrů pro všechny hashe, ke všemu se tyto parametry liší i grafická karta od grafické karty. I při špatném nastavení parametrů je výpočet na GPU rychlejší, než výpočet na CPU.

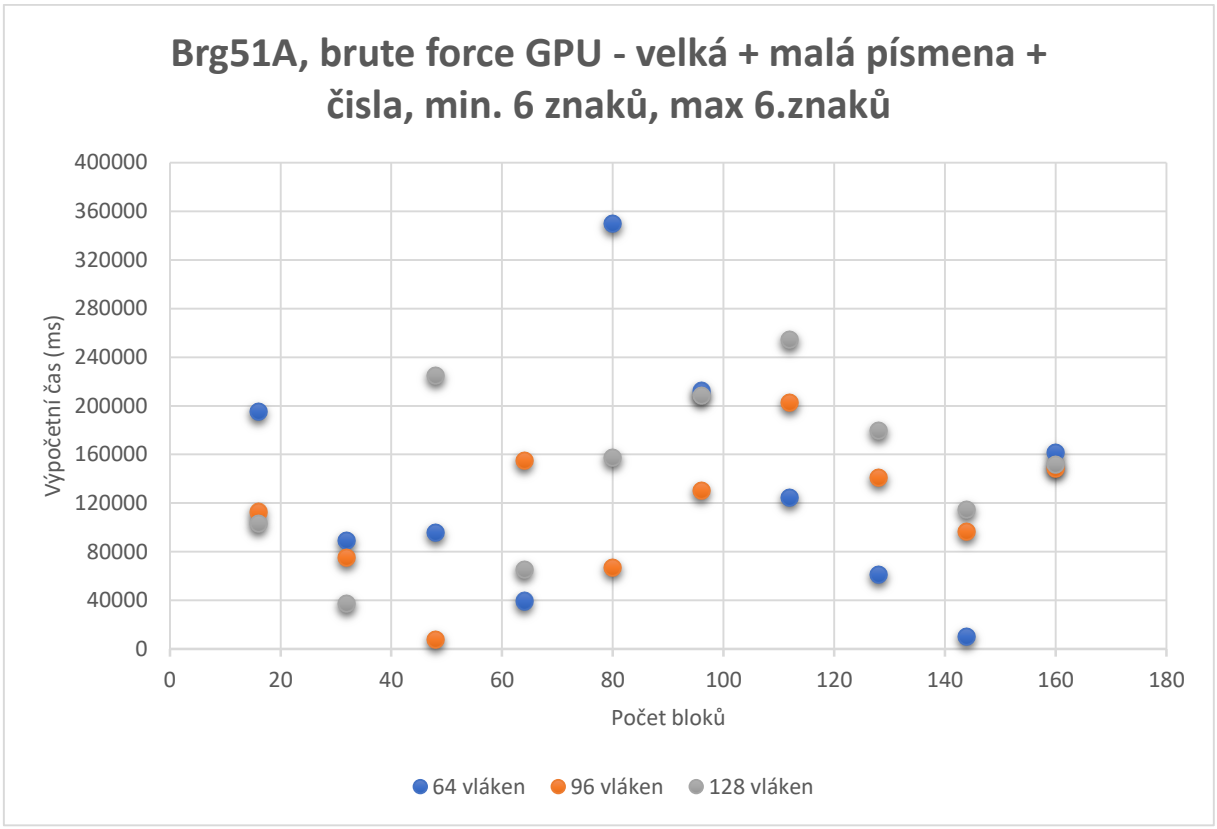
Měření na sestavě 1

Brute force GPU

Následující data byla měřena na sestavě 1.



Nejkratší čas pro zebra1 vychází na **24 sekund** pro konfiguraci 96 vláken a 16 bloků, nejvyšší čas je **362 sekund** pro konfiguraci 128 vláken a 112 bloků. Průměrný čas napříč všemi konfiguracemi je **247 sekund.** Pro 64 vláken je průměrný čas **217 sekund,** pro 96 vláken **221 sekund** a pro 128 vláken **301 sekund**.

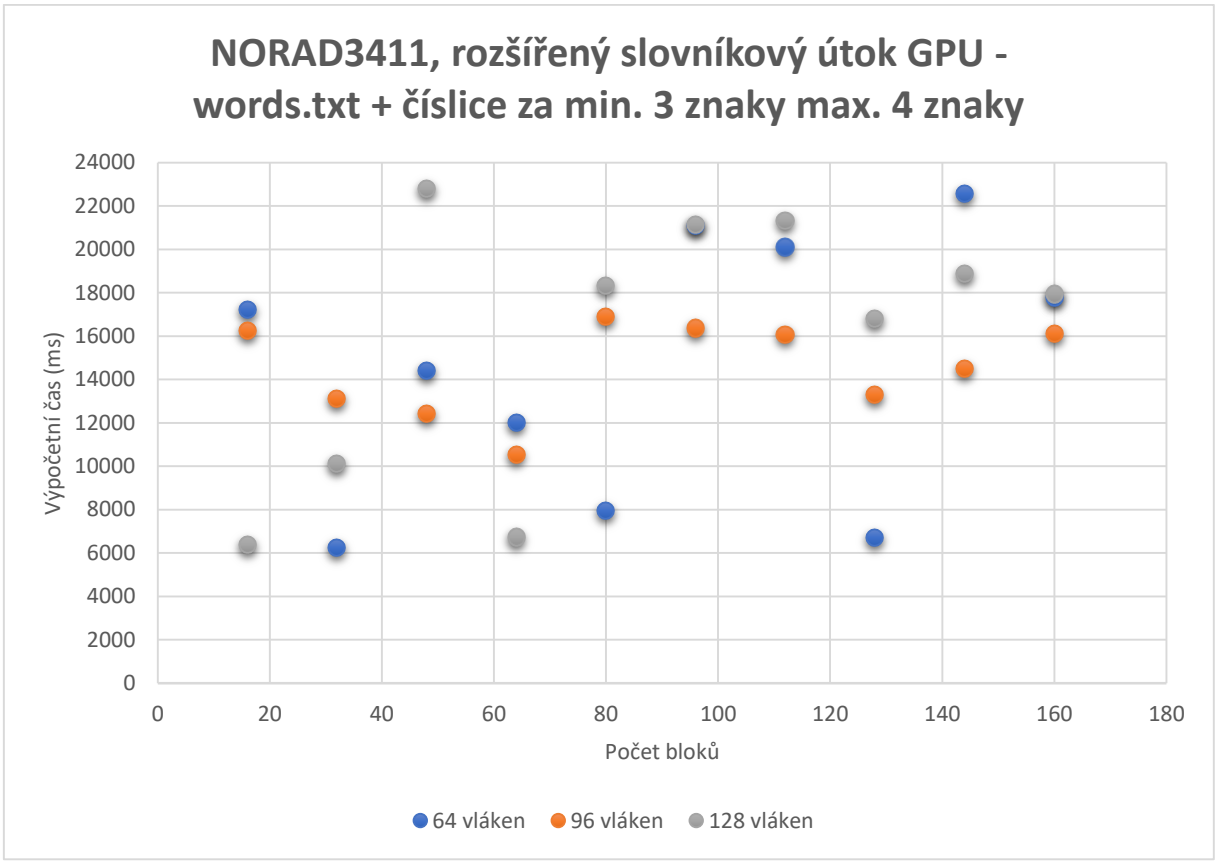


Nejkratší čas pro slovo Brg51A vychází **7,8 sekund** pro konfiguraci 96 vláken a 48 bloků, nejdelší čas je **350 sekund** pro konfiguraci 64 vláken a 80 bloků. Průměrný čas je **132 sekund.** Průměrný čas pro 64 vláken je **134 sekund**, průměrný čas pro 96 vláken je **113 sekund,** průměrný čas pro 128 vláken je **150 sekund.**

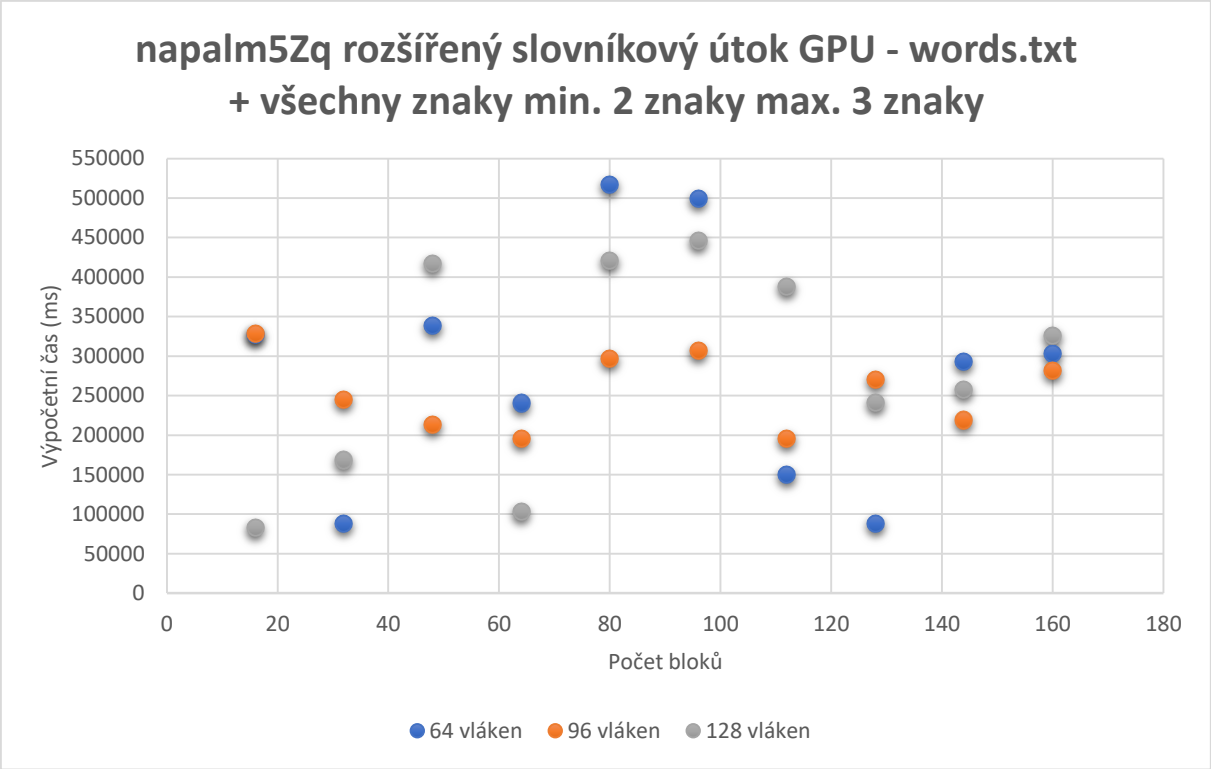
Tato dvě měření ukázala, že nastavení parametrů blocks/threads má v případě útoku hrubou silou velký vliv. Nejspíše ale nebude existovat kombinace parametrů, která by byla optimální pro všechny hashe.

Rozšířený slovníkový útok GPU

Následující data byla měřena na sestavě 1.



Byl testován vliv bloků/vláken na rozšířený slovníkový útok. Jako slovník se používal slovník některých anglických slov (cca. 500 000 slov) a jako pravidla se použilo přidávání řetězců dané abecedy (pouze číslice) v rozsahu 3-4 znaků. Jako heslo bylo zvoleno slovo **NORAD3411.** Nejkratší čas vychází na **6,2 sekund** pro kombinaci 64 vláken a 32 bloků, nejdelší čas **22,8 sekund** pro kombinaci 128 vláken a 48 bloků. Průměrný čas vychází na **15 sekund.** Průměrný čas pro 64 vláken vychází na **14,6 sekund**. Průměrný čas pro 96 vláken vychází na **14,55 sekund**, průměrný čas pro 128 vláken vychází na **16 sekund.**



Nejnižší čas je **82 sekund** pro kombinaci 128 vláken a 16 bloků, nejdelší výpočetní čas je **517 sekund,** průměrný čas je **274 sekund**. Průměrný čas pro 64 vláken je **284 sekund**, pro 96 vláken **255 sekund** a pro 128 vláken je **285 sekund.**

Testování na sestavě 1

Sekvenční řešení

Útok hrubou silou - CPU		
Heslo	**Abeceda**	**Výpočetní čas (ms)**
99999	Malé a velká písmena, číslice rozsah min. 5 max 5	236912
9999	Malé a velká písmena, číslice rozsah min. 4 max 4	3839
zebra1	Malé a velká písmena, číslice rozsah min. 6 max 6	13680000
~A9C	Všechny znaky min. 4 max 4	6115
J@K1!	Všechny znaky min 4 max 5	1262598
Test výkonu[1]	Všechny znaky min 4 max 4	20031

Výpočetní výkon pro sekvenční útok hrubou silou:

$$\frac{Hash}{s} = \frac{[Celkový\ počet\ slov]}{[doba\ vykonávání]s}$$

$$\frac{Hash}{s} = \frac{94^4}{20} = \frac{78074896}{20}$$

$$\frac{Hash}{s} = 3903744$$

Sekvenční řešení útoku hrubou silou má výpočetní výkon **3,9 MHash /s.**

Rozšířený slovníkový útok - CPU

[1] Zkouška výpočetní síly, vložení špatného hashe => nutnost prozkoumat celý stavový prostor

Heslo	Slovník	Pravidla	Výpočetní čas (ms)
ZZZ989	Cca 500k slov	Přidávání za slovo čísla délky 1 a čísla délky 3	121666
napalm5e	Cca 500k slov	Přidávání za slovo čísla a písmena délky 2	246923
NORAD3411	Cca 500k slov	Přidávání za slovo čísla v rozsahu 3-4 znaků	750494
azotemic9S	Cca 500k slov	Přidávání za slovo čísla a písmena délky 2	30351
Test výkonu	Cca 500k slov	Přidávání za slova čísla délky 2	12303

Výpočetní výkon pro rozšířený slovníkový útok:

Výpočet je stejný jako v případě výpočtu výkonu pro útok hrubou silou, počet všech možných slov se získá tím způsobem, že se vezme následující suma:

$$\sum_{k=0}^{počet\ znaků} [počet\ slov\ slovníku] * [velikost\ abecedy]^k$$

Testovací slovník obsahuje přesně **466551 slov**, za každé z těchto slov můžeme přidat celkem 100 různých čísel (00-99) celkově teda program musí otestovat **466551 + 466551*100** slov.

Rozšířený slovníkový útok pomocí CPU má výpočetní výkon přibližně **3,830MHash/s**, takže přibližně srovnatelný se výpočetním výkonem útoku hrubou silou.

Řešení pomocí CUDA

Vzhledem k velkému vlivu parametrů threads a blocks na celkový čas bude v tabulce uveden nejlepší možný naměřený čas, nejhorší možný naměřený čas a průměrný naměřený čas.

Útok hrubou silou - GPU				
Heslo	**Abeceda**	**Výpočetní čas (ms)**		
		Nejlepší	**Nejhorší**	**Průměrný**
zebra1	Malé a velká písmena, číslice rozsah min. 6 max 6	24033	362224	246851
99999	Malé a velká písmena, číslice rozsah min. 5 max 5	2892	6218	5136
9999	Malé a velká písmena, číslice rozsah min. 4 max 4	378	432	413
~A9C	Všechny znaky min. 4 max 4	362	677	518
J@K1!	Všechny znaky min 4 max 5	4643	47982	26766
Test výkonu	Všechny znaky min 5 max 5	22163	48871	40996

Výpočetní výkon pro útok hrubou silou realizovaný na GPU:

Výpočet je totožný s výpočtem pro sekvenční útok na hrubou silou, jen se vezme jiný počet všech slov (94^5).

S optimálně nastavenými parametry THREADS/BLOCKS vychází výpočetní síla hrubé síly na GPU na **331 MHash/s** (téměř 85x více než CPU varianta), při špatném nastavení parametrů vychází výpočetní výkon na **150 MHash/s** (cca 39x rychlejší než CPU varianta) a průměrný výpočetní výkon napříč nastaveními vychází je **179 MHash/s** (cca 46x rychlejší než CPU varianta).

Rozšířený slovníkový útok - GPU					
Heslo	**Slovník**	**Pravidla**	**Výpočetní čas(ms)**		
			Nejlepší	**Nejhorší**	**Průměrný**
ZZZ989	*	Přidávání za slovo čísla délky 1 a čísla délky 3	2020	3257	2724
napalm5e	*	Přidávání za slovo čísla a písmena délky 2	2753	9151	5866
NORAD3411	*	Přidávání za slovo čísla v rozsahu 3-4 znaků	6240	22782	15066
azotemic9S	*	Přidávání za slovo čísla a písmena délky 2	2439	8902	4688
Test výkonu		Přidávání za slova čísla délky 2	15506	26764	22184

* Cca 500k slov

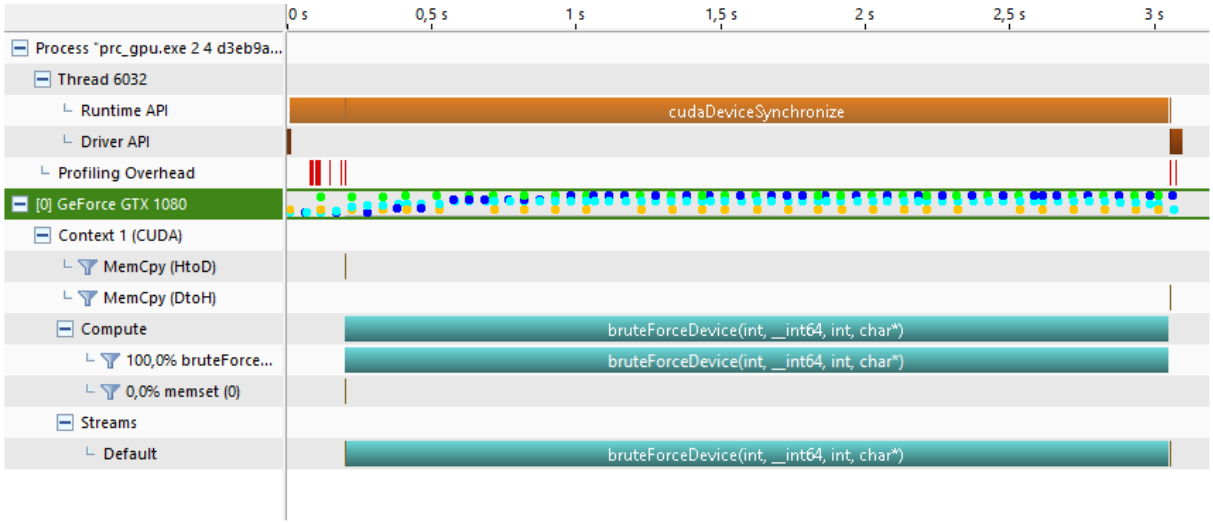
U slovníkového útoku hodně záleží na umístění daného slova ve slovníku, např. u slova azotemic, které je ve slovníku na začátku není zrychlení na GPU tak výrazné.

Výpočetní výkon pro rozšířený slovníkový útok

Výpočet probíhá stejným způsobem jako v případě sekvenčního řešení. S optimálně nastavenými parametry je výpočetní síla rozšířeného slovníkového útoku na GPU **301 MHash/s** (cca 79x rychlejší než sekvenční řešení), při špatném nastavení parametrů je výpočetní síla **174 MHash/s** (cca. 46x výkonější) a průměrný výpočetní výkon je **210 MHash/s** (cca 55x výkonnější než sekvenční řešení).

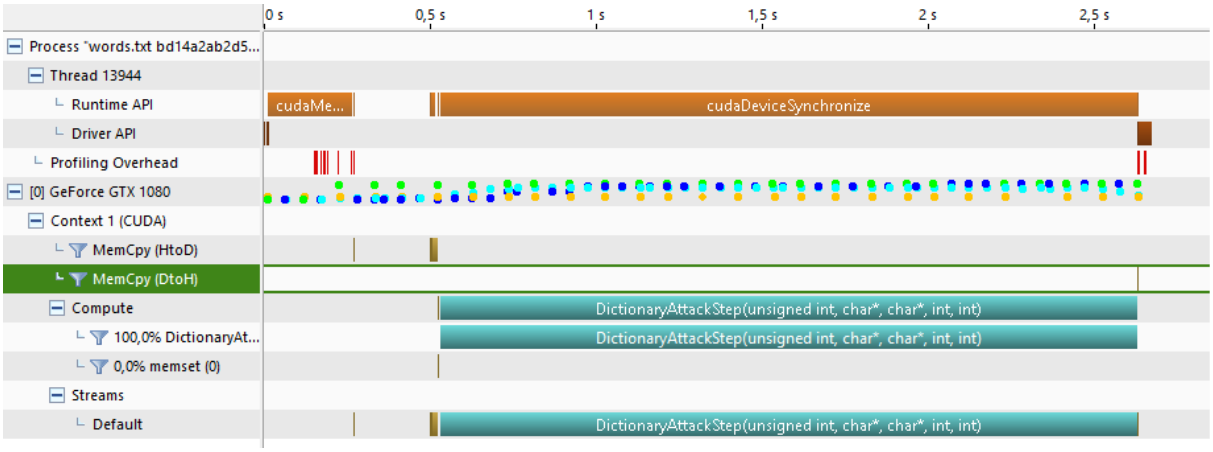
Profilování

Útok hrubou silou



V případě útoku hrubou silou nelze na profileru vidět nic moc zajímavého. Před samotným spuštěním kernelu se provádí kopírování abecedy a hashe do paměti symbolů. Poté se vykonává samotný kernel a následně se výsledek kopíruje zpátky na hostitele.

Rozšířený slovníkový útok



V případě rozšířeného slovníkového útoku je to podobné jako v případě útoku hrubou silou. Tentokrát se ale kopíruje větší množství informací do paměti konstant, takže tato doba je výraznější. Poté se kopírují samotná slova ze slovníku do globální paměti, poté se pro daná slova ze slovníku spouští samotný kernel. Vzhledem k tomu, že paměť pro slova byla nastavena hodně vysoko (4GB), takže se všechna slova ze slovníku do paměti GPU překlopí v jedné iteraci, v případě většího slovníku by iterací proběhlo více. Během profilování jsem v kódu našel chybu v podobě špatného využití cudaMemcpy, tato funkce vykonává synchronní kopírování do paměti, takže se to odklání od toho, že se další slova budou do globální paměti kopírovat během výpočtu kernelu. Pro správné chování by bylo nutné využít cudaMemcpyasync.

Testování na sestavě 2

Sekvenční řešení

Útok hrubou silou - CPU		
Heslo	**Abeceda**	**Výpočetní čas (s)**
99999	Malé a velká písmena, číslice rozsah min. 5 max 5	550.667179668
9999	Malé a velká písmena, číslice rozsah min. 4 max 4	8.682245876
~A9C	Všechny znaky min. 4 max 4	14.116950316

J@K1!	Všechny znaky min 4 max 5	2935.052946226

Rozšířený slovníkový útok - CPU			
Heslo	**Slovník**	**Pravidla**	**Výpočetní čas (s)**
ZZZ989	Cca 500k slov	Přidávání za slovo čísla délky 1 a čísla délky 3	292.202
napalm5e	Cca 500k slov	Přidávání za slovo čísla a písmena délky 2	582.233
azotemic9S	Cca 500k slov	Přidávání za slovo čísla a písmena délky 2	72.538

Řešení pomocí CUDA

Zde je opět uváděn vždy pouze nejlepší, nejhorší a průměrný čas (kde se liší počty threadů a bloků).

Útok hrubou silou - GPU				
Heslo	**Abeceda**	**Výpočetní čas (s)**		
		Nejlepší	**Nejhorší**	**Průměrný**
zebra1	Malé a velká písmena, číslice rozsah min. 6 max 6	28.71657738	374.7755428	115.0451145
99999	Malé a velká písmena, číslice rozsah min. 5 max 5	2.842191706	7.49441396	4.145423028
9999	Malé a velká písmena, číslice rozsah min. 4 max 4	1.230937135	1.379836392	1.292344389
~A9C	Všechny znaky min. 4 max 4	1.226114675	1.870448209	1.39379288
J@K1!	Všechny znaky min 4 max 5	3.113347551	48.61088376	14.96271825

Všechna naměřená data jsou v přiloženém souboru. Lze si všimnou, že pro zebra1 je rozdíl mezi nejlepším a nejhorším časem zhruba 170 %. Zatímco pro 99999 se jedná o 90 %. U zebra1 bylo nejlepšího výsledku dosaženo za použití 128 bloků a 96 threadů. U 99999 64 bloků a 64 threadů.

Pro slovo 99999 je nejlepší čas zhruba 180x lepší než při sekvenčním řešení. I nejhorší naměřený čas je stale mnohem lepší než sekvenční řešení. To platí i pro ostatní případy, vždy došlo ke zlepšení.

Rozšířený slovníkový útok - GPU					
Heslo	**Slovník**	**Pravidla**	**Výpočetní čas(s)**		
			Nejlepší	**Nejhorší**	**Průměrný**
ZZZ989	Cca 500k slov	Přidávání za slovo čísla délky 1 a čísla délky 3	2.026819944	3.934579849	2.549185324
napalm5e	Cca 500k slov	Přidávání za slovo čísla a písmena délky 2	9.176009893	58.19028997	20.57271666

NORAD3411	Cca 500k slov	Přidávání za slovo čísla v rozsahu 3-4 znaků	18.45876002	158.6113901	50.30224834
azotemic9S	Cca 500k slov	Přidávání za slovo čísla a písmena délky 2	2.510080099	14.02815986	5.847591321

Opět jsou naměřené časy lepší než sekvenční. Například pro azotemic9S je průměrný naměřený čas lepší zhruba 12x než u sekvenčního řešení.

Vzhledem k tomu, že měření na Sestavě 2 bylo provedeno s využitím jiného kompilátoru a bez optimalizace, tak porovnání měření na těchto dvou strojích může být zavádějící.

Závěr

Měření ukázalo, že grafická karta je vhodný nástroj na prolamování kryptografických hashů. I ne příliš dobře optimalizované řešení nabízelo výrazné zrychlení. Jedná se o zcela základní řešení, které toho příliš mnoho neumí a ani z daleka nedosahuje kvalit ostatních nástrojů (hashcat). Co by bylo možné do budoucna zlepšit:

- **Rozšíření možností slovníkového útoku** – aktuálně pouze podporuje přidávání řetězců za slova, v budoucnu by bylo možné triviálně implementovat přidávání znaků před/do slova. Dále by bylo možné implementovat nahrazování znaků za jiné znaky (např. 0 → O, I → 1 atd.), či třeba spojování více řetězců za sebe. U těchto rozšíření by bylo ale třeba zvážit potřeba řazení daných slov podle délek, aby nedocházelo k nerovnoměrné práci napříč vlákny.
- **Optimalizace MD5 algoritmu**
- **Změření vlivu THRESHOLD, DICTIONARY_THRESHOLD, GRANULARITY a dalších na výpočetní čas**
- **Vytvořit velký statistický vzorek na zkoumání vlivu hodnot BLOCKS a THREADS** – pomocí tohoto vzorku by bylo pak možné určit „nejideálnější“ paremetry napříč problémy.
- **Atd**..

Literatura

- https://cs.wikipedia.org/wiki/Kryptografická_hašovací_funkce
- https://courses.fit.cvut.cz/MI-PRC/media/labs/report_crypto.pdf