Semestrální práce z předmětu KIV/PPR

Prolomení šifry SkipJack

Zdeněk Valeš

22.11. 2018

1 Zadání

Vaším úkolem bude prolomit šifru SkipJack. Tuto šifru je výpočetně náročné prolomit hrubou silou, nicméně lze zkusit i sofistikovanější metody např. genetické a evoluční algoritmy. Abyste prolomení urychlili, lze referenční kód přepsat a vektorizovat na úrovni instrukcí, pomocí GPU, případně ho distribuovat pomocí MPI.

Samostatná práce využije alespoň dvě z celkem tří možných technologií:

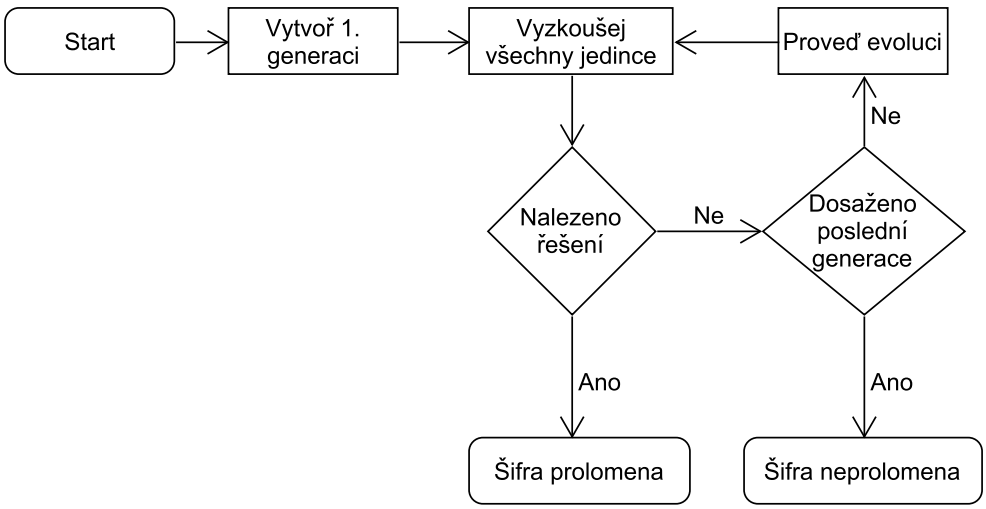
- Paralelní program pro systém se sdílenou pamětí - C++, popř. WinAPI
- Program využívající asymetrický multiprocesor - konkrétně x86 CPU a OpenCL kompatibilní GPGPU - C++ AMP
- Paralelní program pro systém s distribuovanou pamětí - C++ MPI

Práci implementujte v jazyce C++ do základu aplikace, který je k dispozici na CourseWare.

2 Popis řešení

Vstupem prolamovací funkce je zašifrovaný blok a referenční blok. Cílem algoritmu je nalezení klíče (jedince) po jehož použití v dešifrovací funkci bude vstupní blok identický s referenčním. Algoritmus je znázorněn na obrázku 1.

K prolomení šifry jsem použil diferenciální evoluci s Hammingovou vzdáleností jako cenovou funkcí. Části programu jsou paralelizované pomocí knihoven Intel TBB a OpenCL.



Obrázek 1: Algoritmus řešení

2.1 Diferenciální evoluce

Diferenciální evoluce je v principu velmi podobná klasickým genetickým algoritmům. Liší se od nich počtem rodičů (>2) potřebným k tvorbě potomka a pořadím operací mutace, křížení [1].

Při evoluci jedince se nejprve mutací získá šumový vektor, který se pak zkříží s aktivním jedincem (právě zpracovávaný jedinec v generaci). Pokud má takto vzniklý jedinec lepší skóre než aktivní, použije se do nové populace. Evoluce je řízena dvěma parametry: mutační konstantou F a prahem křížení CR. Doporučené hodnoty jsou 0,3 – 0,9 pro F a 0,8 – 0,9 pro CR [1]. Hodnoty použité v mém řešení jsou uvedeny v tabulce 1.

Parametr	Hodnota
F	0,3
CR	0,5
Velikost populace	4160
Maximální počet generací	1000 - 2000

Tabulka 1: Hodnoty parametrů evoluce

V algoritmu diferenciální evoluce je možné použít několik mutačních funkcí [1], na doporučení vyučujícího jsem použil mutační funkci *best/2*. Ta je znázorněna v rovnici 1, v_{best} je nejlepší jedinec v současné generaci, v_1, v_2, v_3, v_4 jsou náhodně vybraní, nestejní jedinci (pro každého aktivního jedince jsou vybráni znovu) a F je výše zmíněná mutační konstanta. Výsledkem mutace je šumový vektor $noise$.

$$noise = v_{best} + F \cdot (v_1 + v_2 - v_3 - v_4) \tag{1}$$

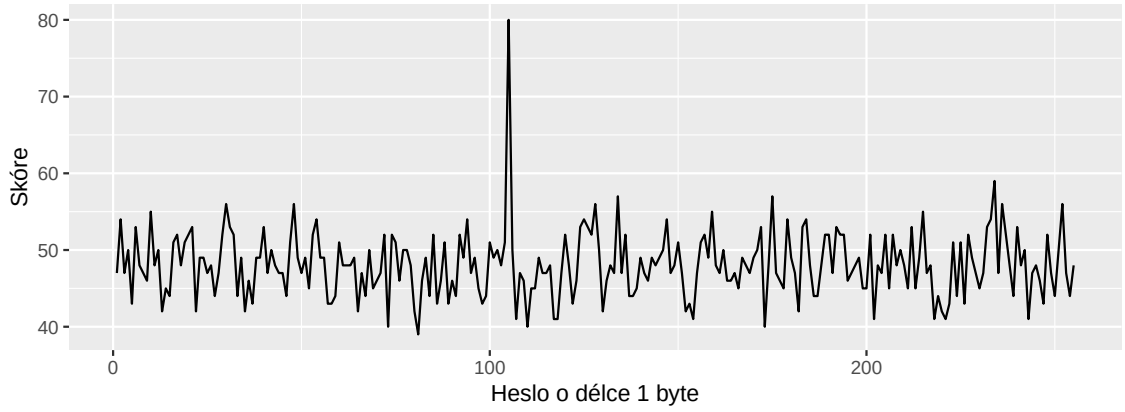
Šumový vektor je následně zkřížen (binomické křížení) s aktivním jedincem a pokud je výsledný vektor lepší než aktivní jedinec, použije se v nové generaci.

2.1.1 Cenová funkce

Jako cenovou funkci jsem použil variantu Hammingovo vzdálenosti dešifrovaného bloku od bloku referečního. Během implementace jsem experimentoval s více cenovými funkcemi a držel jsem se konvence větší skóre znamená lepšího jedince. U Hammingovo vzdálenosti to ale neplatí, proto jsem tuto vzdálenost odečetl od maximální možné vzdálenosti (viz rovnice 2).

$$fitness = D_{max} - D(decrypted, reference) \tag{2}$$

Tato cenová funkce nekonverguje, vzhledem k povaze šifry SkipJack, příliš dobře, jak je vidět na obrázku 2. Během přednášek nám byla vyučujícím doporučena dvojrozměrná cenová funkce, tu se mi bohužel nepodařilo funkčně naimplementovat.



Obrázek 2: Hodnoty fitness funkce pro klíč délky 1

Kromě Hammingovo vzdálenosti dešifrovaného bloku od referenčního jsem experimentoval i se vzdáleností jedince od referenčního hesla. V tomto případě algoritmus konverguje dobře (100-200 generací), nicméně znalost hesla není běžný případ a proto jsem tuto cenovou funkci použil pouze k odstranění chyb v implementaci evolučního algoritmu.

2.2 Paralelizace

Výpočetně nejnáročnější část algoritmu je (kromě fitness funkce, která vytváří klíč a dešifruje) mutační funkce. Proto jsem s paralelizací začal právě v tomto místě. K tomu jsem postupně využil tři technologie: vektorové instrukce, knihovnu Intel TBB a knihovnu OpenCL.

Vektorové instrukce Procesor v mém počítači podporuje vektorové instrukce SSE 2 a nižší. Tyto instrukce využívájí vektor délky 128 bitů. Do tohoto vektoru se vejde 16 bytů, což by při maximální délce hesla 10 bytů bylo dostačující, nicméně mutace *best-2* vyžaduje násobení floatem. Nejmenší float, který lze do vektoru uložit zabírá 32 bitů, do vektoru se tedy vejdou jen 3. Aby bylo možné vektorové instrukce v mutační funkci použít, je tedy nutné každý 1 byte hesla roztáhnout na 32 bitů, provést potřebné operace a poté jej z 32 bitů převézt zpátky na 8 bitů. Tato nadbytečná režie způsobila neefektivnost vektorových instrukcí a proto jsem je v mutační funkci nakonec nepoužil.

2.2.1 Intel TBB

Intel TBB je knihovna pro jazyk C++ sloužící k paralelizacei programů.

NAPSAT LÍP

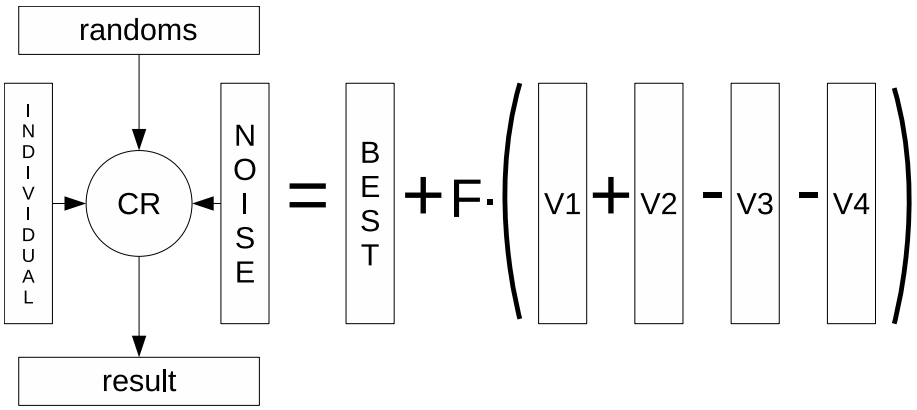
Knihovnu jsem použil k paralelizaci evoluce. Původně jsem populaci manuálně rozdělil na dávky, které jsem pak zpracovával pomocí *parallel_for()*. Tento přístup se ukázal jako neefektivní, protože rozdělení dat mezi vlákny zvládá knihovna sama (a lépe). Proto

jsem nakonec logiku evoluce jedince přesunul do funktoru, který následně předávám do *parallel_for()*.

2.2.2 OpenCL

OpenCL slouží mimojiné k paralelizaci výpočtů pomocí CPU, nebo GPU. Výpočet probíhá v tzv. kernelu, což je funkce napsaná v jazyce OpenCL a přeložená pro dané výpočetní zařízení. Parametry funkce jsou typicky ukazatele na pole s daty z nichž kernel vybere jeden 'řádek' nad kterým provede operaci.

V mém řešení jsem do kernelu přenesl mutaci a křížení jednoho bytu hesla – kernel je znázorněný na obrázku 3. V kroku evoluce se tedy nejprve nageneruji potřebná data pro celou populaci (pole s vektory $v1$,$v2$,$v3$,$v4$ a pole *randoms* s náhodnými čísly pro křížení), nad nimi proběhne paralelizovaný výpočet evoluce a nakonec jsou z výsledků vybráni jedinci, kteří přežijí do následující generace.



Obrázek 3: Operace prováděná v kernelu

Kvůli nutnosti předpřipravit všechna data před začátkem výpočtu zabírá program při použití OpenCL velké množství paměti – pole s vektory a náhodnými čísly zabírají téměř 2 MB. V mém počítači mám bohužel jen intergrovanou grafickou kartu ...

3 Výsledky

Všechna testování jsem prováděl na svém počítači jehož specifikace jsou uvedeny v tabulce 2. Pro různá nastavení populace a způsobu zpracování jsem program nechal proběhnout dvacetkrát. Výsledky získané měřením jsou uvedeny v tabulkách 3, 4 a 5. Měření bylo prováděno s délkou hesla 10 bytů. Šifru se podařilo prolomit pouze při snížení délky hesla na 1 byte.

Parametr	Hodnota
CPU	Intel Pentium B970, 2 jádra, 2,30 GHz, 2MB L3 Cache
GPU	integrovaná
RAM	8 GB
OS	Windows 7 Ultimate SP 1

Tabulka 2: Specifikace počítače

Počet generací	Průměrná doba běhu [s]	Skóre nejlepšího jedince
1000	12,457	54
1500	18,589	54
2000	24,774	52

Tabulka 3: Tabulka s výsledky pro běh bez paralelizace

Počet generací	Průměrná doba běhu [s]	Skóre nejlepšího jedince
1000	6,825	53
1500	10,238	53
2000	13,611	52

Tabulka 4: Tabulka s výsledky pro běh s paralelizací knihovnou Intel TBB

Počet generací	Průměrná doba běhu [s]	Skóre nejlepšího jedince
1000	13,111	52
1500	19,5305	54
2000	25,923	50

Tabulka 5: Tabulka s výsledky pro běh s paralelizací knihovnou OpenCL

Z měření vyplývá, že největšího urychlení bylo dosaženo použitím knihovny Intel TBB. Oproti tomu použití knihovny OpenCL program v podstatě neurychlilo, což může být způsobeno špatnou implementací programu, nebo použitím na nevhnodnou část výpočtu.

4 Uživatelská příručka

Program lze přeložit a sestavit pomocí standardního C/C++ překladače (gcc, cl).

Při sestvení musí mít linker přístup k tbb a opencl.

Použití: program.exe ref enc

5 Závěr

Šifru se nepodařilo prolomit, vyzkoušel jsem si paralelizaci, OpenCL se mi nepodařilo naimplementovat tak, aby to zrychlovalo.

Reference

[1] HLAVÁČEK, Jiří. *Moderní adaptivní diferenciální evoluce* [online]. Zlín, 2015 [cit. 2018-11-22]. Dostupné z: <https://theses.cz/id/vs0ow2/>. Diplomová práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky. Vedoucí práce doc. Ing. Roman Šenkeřík, Ph.D