

# **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:

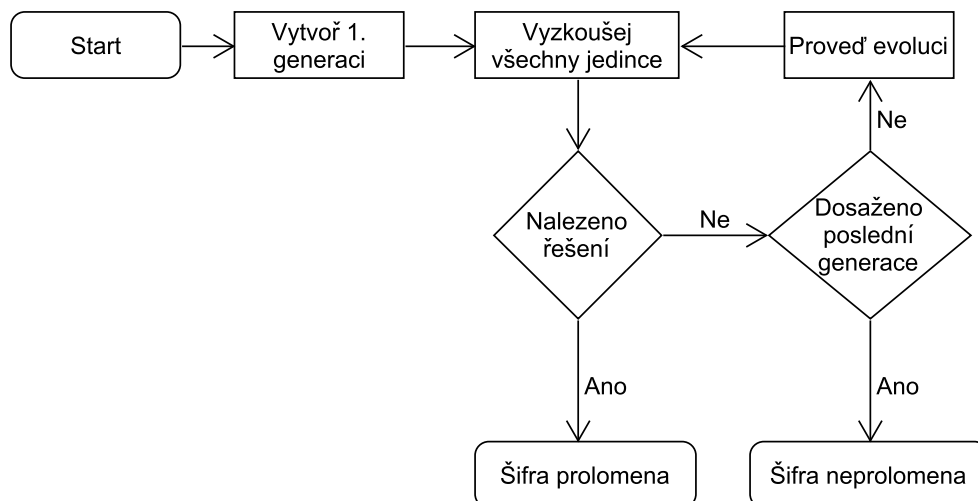
- Paralelní program pro systém se sdílenou pamětí - C++, popř. WinAPI
- Program využívající asymetrický multiprocesor - konkrétně x86 CPU a OpenCL kompatibilní GPGPU - C++ AMP
- Paralelní program pro systém s distribuovanou pamětí - C++ MPI

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	800

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*.

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

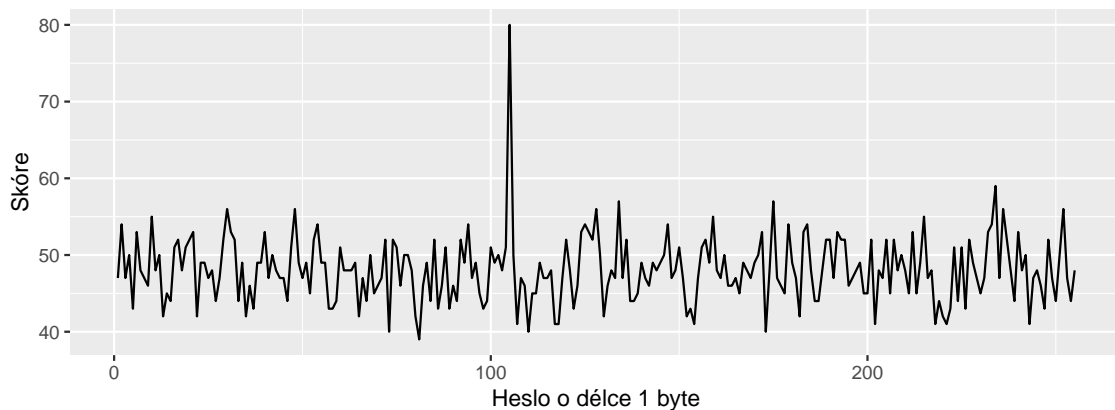
Šumový vektor je následně zkřížen (binomické křížení) s aktivním jedincem a pokud je výsledný vektor lepší než aktivní jedinec, použije se v nové generaci.

### 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).

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

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 využívají vektor délky 128 bitů. Do tohoto vektoru se vejde 16 bajtů, což by při maximální délce hesla 10 bajtů 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 bajt hesla roztáhnout na 32 bitů, provést potřebné operace a poté jej z 32 bitů převést 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 paralelizaci programů.

#### NAPSAT LÍP

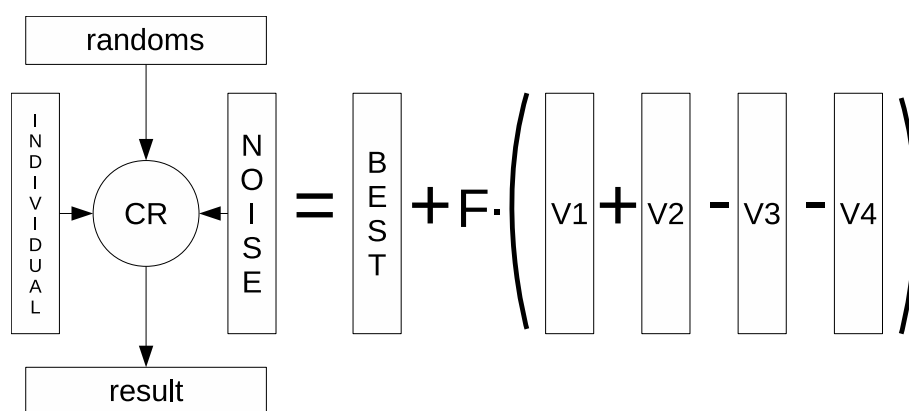
Knihovnu jsem použil k paralelizaci evoluce. Původně jsem populaci manuálně rozdělil na dávky, které jsem pak zpracovával pomocí *parallel\_for()*. Tento přístup se ukázal jako neefektivní, protože rozdělení dat mezi vlákna zvládá knihovna sama (a lépe). Proto

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

### 2.2.2 OpenCL

OpenCL slouží mimo jiné 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 prvku – kernel je znázorněn na obrázku 3. V kroku evoluce se tedy nejprve nagenarují potřebná data pro celou populaci (pole s vektory  $v_1, v_2, v_3, v_4$  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

V mém počítači mám bohužel jen integrovanou grafickou kartu a tak je rozdíl oproti použití knihovny TBB téměř nepatrný, jak je vidět v části 3.

## 3 Výsledky

## 4 Závěr

## 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