Paralelní programování - semestrální projekt

Robert Čížek, Lukáš Hlavatý, Peter Smatana

27. června 2017

1 Zadání projektu

Cílem projektu bylo vytvořit program který se snaží hrubou silou rozšifrovat šifrový text. Šifrovaný text je vytvořený pomocí Vigenèrovy šifry.

2 Požadavky na softwarové vybavení

Projekt je vytvořený v progamovacím jazyku Java ve verzi 1.7. Pro vytvoření stuktury projektu, pro běh a tvorbu *.jar* souborů používáme technologii Maven. Pro běh projektu je třeba mít nainstalované Java Runtime Environment. Pro práci na projektu je třeba mít Java Development Kit.

3 Popis projektu

Naše řešení je schopno text šifrovat i dešifrovat. Šifrování se provádí ve třídě `Encryptor` v metodě `Encrypt`. V této metodě iterujeme nad otevřeným textem a podle klíče jej šifrujeme. O to se stará třída `Alphabet`, která má metody `shiftUp` a `shiftDown`. Metoda `shiftUp` šifruje text podle klíče a metoda `shiftDown` dešifruje text podle klíče. Ve skutečnosti se jedná o hledání patřičného znaku ve Vigenèrově čtverci.

Ve zdrojovém kódu č. 1 je vidět jak se pracuje Vigenèrovým čtvercem. Nalezení zašifrovaného znaku je přičtení ordinální pozice znaku klíče v ASCII tabulce.

```
1 public static char shiftUp(char c, char key) {
2     int num = (int) c - 97;
3     if ((num >= 0) && (num <= 25)) {
4         num += (int) key - 97;
5         if (num > 25) {
6             num -= 26;
7         }
```

```
8           return alphabet.charAt(num);
9       } else {
10          return c;
11      }
12  }
```

Zdrojový kód č. 1: Šifrování znaku podle znaku v klíči.

Pro dešifrování máme třídu `Decryptor` ve které jsou tři metody kde uplatňujeme odlišné přístupy na dešifrování. První metoda `KeyDecrypt` dešifruje text inverzně stejně jako je implementováno šifrování. Vezme šifrový text, klíč a obdobně jako v ukázce zdrojového kódu č. 1 dešifruje text. Druhý přistup reprezentuje metoda `FreqDecrypt`, která provádí frekvenční analýzu. Frekvenční analýza je přístup, jak najít v šifrovém textu nějakou podobnost s přirozeným jazykem, ve kterém je zpráva napsaná. Poslední přístup je hledání klíče hrubou silou podle slovníku. Hádáme klíč, klíčem rozšifrováváme slovo a toto slovo hledáme ve slovníku.

4 Paralelní vylepšení programu

Paralelizaci jsme prováděli pro dešifrování textu. Šifrování jsme implementovali pro potřebu získání zašifrovaného textu.

Dešifrování je provedeno rozdělením šifrovaného textu do částí které jsou následně předány vláknům (viz zdrojový kód č. 2). Vlákna nezávisle na sobě zpracovávají dané části a výsledky zapisují do společné proměnné pomocí metod nebo do patřičného pole podle svého ID. Počet spuštěných vláken je závislý na délce šifrovaného textu.

```
1  public static String Encrypt(String text, String key){
2      sKey = key;
3      StringBuilder result = new StringBuilder();
4      String[] batch = Stringer.Split(text);
5      Thread[] bank = new Thread[batch.length];
6      count = new CountDownLatch(batch.length);
7      sBatch = new String[batch.length];
8      for(int i = 0; i < batch.length; i++){
9          bank[i] = new Thread(new Encryptor(batch[i], i));
10         bank[i].start();
11     }
12
13     try {
14         count.await();
15     } catch (InterruptedException ex) {
16         Logger.getLogger(
```

```
17                 Encryptor.class.getName()).log(Level.SEVERE, null, ex);
18         }
19
20         for (String s : sBatch) {
21             result.append(s);
22         }
23         return result.toString();
24     }
```

Zdrojový kód č. 2: Rozdělení textu a předání vláknům.

5 Měření

Používali jsme tři zdroje dat pro šifrování a dešifrování (viz tabulka č. 1).

název souboru	**počet znaků**	**počet slov**
blabol1k.txt	1 134	147
blabol10k.txt	11 224	1 495
blabol40k.txt	44844	6005

Tabulka č. 1: Testovací data

číslo měření	**lineární zpracování**			**paralelní zpracování**		
	1k	10k	40k	1k	10k	40k
1	2,45	3,36	9,11	2,63	2,92	2,46
2	2,51	3,10	8,88	2,55	2,29	2,66
3	2,61	4,07	8,84	2,54	2,50	2,56
4	3,03	3,46	9,79	2,60	2,46	2,69
5	2,42	3,01	8,55	2,44	2,66	2,75
6	2,46	3,27	8,48	2,41	2,43	2,63
7	2,29	3,07	9,06	2,49	2,57	2,48
8	2,87	3,65	8,58	2,52	2,53	2,66
9	2,78	2,86	8,78	2,53	2,62	2,58
10	2,50	3,43	8,82	2,46	2,43	2,71
průměr	2,59	3,33	8,89	2,52	2,54	2,62

Tabulka č. 2: KeyDecrypt

číslo měření	**lineární zpracování**			**paralelní zpracování**		
	1k	10k	40k	1k	10k	40k
1	2,95	3,04	6,29	2,81	2,93	2,72
2	2,67	2,92	6,77	2,39	2,97	2,81
3	2,79	2,82	6,16	2,55	2,57	2,79
4	2,42	2,87	6,04	2,94	2,64	2,84
5	2,41	2,75	6,04	2,42	2,95	2,68
6	2,64	2,64	5,75	2,85	2,76	2,98
7	2,44	2,88	5,85	2,97	2,65	2,90
8	2,35	3,21	5,91	2,36	2,86	2,73
9	2,50	2,91	5,43	2,47	2,75	2,95
10	3,16	2,75	5,65	2,69	2,81	2,74
průměr	2,63	2,88	5,99	2,64	2,79	2,81

Tabulka č. 3: Encrypt

číslo měření	**lineární zpracování**			**paralelní zpracování**		
	1k	10k	40k	1k	10k	40k
1	3,38	21,05	283,92	3,11	19,62	254,40
2	2,96	20,66	291,43	3,37	19,43	255,37
3	3,04	20,30	281,26	2,74	18,67	253,65
4	2,82	21,06	282,64	3,16	19,85	254,19
5	3,23	20,46	281,37	2,85	18,67	253,87
6	3,47	20,66	281,95	3,18	18,73	255,02
7	2,96	21,09	284,34	3,06	18,45	254,41
8	3,02	20,70	280,14	2,95	19,06	254,34
9	3,31	20,79	281,35	2,86	18,63	254,41
10	2,81	20,15	281,48	3,11	18,71	255,17
průměr	3,10	20,69	282,99	3,04	18,98	254,48

Tabulka č. 4: FreqDecrypt

	overhead		
číslo měření	1k	10k	40k
1	2,66	2,50	2,58
2	2,43	2,75	2,37
3	2,47	2,45	2,76
4	2,47	2,60	2,34
5	2,42	2,85	2,50
6	2,50	2,60	2,63
7	3,14	2,40	2,33
8	2,49	2,66	2,76
9	2,41	2,46	2,29
10	2,35	2,54	2,58
průměr	2,54	2,58	2,51

Tabulka č. 5: cccccccc

	Enc	KD	FD
A 1k	0,10	-0,04	0,47
A 10k	0,30	0,45	17,81
A 40k	3,48	2,90	277,00
T 1k	0,11	-0,13	0,40
T 10k	0,21	-0,25	16,19
T 40k	0,30	-0,20	251,67
A-T 1k	-0,01	0,08	0,07
A-T 10k	0,09	0,70	1,62
A-T 40k	3,18	3,09	25,33

Tabulka č. 6: cccccccc

5.1 Použitý hardware

Tabulka č. 7 ukazuje, na jakém hardware projekt běžel. Přestože je hardware různě výkonný, na výsledky to prakticky nemělo vliv. Za příčinu obdobných výsledků lze především považovat stejný počet procesorových jader, stejně tak jako absence technologie hyperthreading u starších procesorů značky Intel. Procesory se tak liší pouze v taktovaní, jež v paralelním zpracování nemá výrazný vliv.

Procesor	**Paměť**
Intel Pentium DualCore E6300	4BG DDR2
Intel Core i3-2340UE	4GB DDR3
AMD Phenom X2	4GB DDR2

Tabulka č. 7: Použitý hardware pro testování

6 Práce na projektu

Na projektu se všichni členové týmu podíleli stejným dílem. Projekt jsme umístili na github: `https://github.com/petersmatana/ppp`

7 Závěr

Z výsledků v tabulce č. 6 vyplývá, že časová náročnost algoritmu metody Encrypt a Keydecrypt je téměř totožná, ale u algoritmu metody FreqDecrypt je vidět, že její složitost je vyšši a tedy paralelizací získává největší časovou výhodu. Hlavní příčinou je paralelizace zpracování samotných částí textu - efekt zrychlení zpracování je zde závislý na velikosti souboru dat. Čím větší vstupní soubor bude, tím lepších výsledků aplikace v paralelním módu dosáhne. Alternativní možností paralelizace by pak mohlo být dešifrování (hledání) samotného šifrovacího klíče. V principu by se rozdělilo samotné hledání, včetně slovníkových operací na více vláken a tak by se zefektivnil proces opakovaného zpracování

velkého množství kombinací. Takové zpracování by pak nebylo závislé navelikosti vstupního souboru, ale na použitém slovníku či množství možných kombinací šifrovacího klíče.