Projekt Zespołowy

Etap projektu – projektowanie rozwiązania na zadaną architekturę

Autorzy:
Biernacka Kamila
Kania Dominik
Leśniak Mateusz
Maziarz Wojciech

kwiecień 2021

Streszczenie

Poniższe sprawozdanie jest wynikiem naszej pracy na drugim etapie projektu zespołowego z implementacji metody indeksu w architekturach GPU. Przedstawimy w nim przygotowane przez nas projekty i rysunki koncepcyjne wymaganych do zaimplementowania algorytmów.

Spis treści

1 Mnożenie modularne dużych liczb

W celu wykonania mnożenia dużych liczb $a, b \in \mathbb{F}_p$ wykorzystamy algorytm 2. Pierwszym krokiem jest przedstawienie liczb a, b w postaci $a = x_1 \cdot 2^{32} + y_1$ oraz $b = x_2 \cdot 2^{32} + y_2$.
Wtedy
$r_p = r_p(ab) = r_p((x_1 \cdot 2^{32} + y_1)(x_2 \cdot 2^{32} + y_2)) = r_p(x_1x_2 \cdot_p 2^{64}) +_p r_p(x_1y_2 \cdot_p 2^{32}) +_p r_p(x_2y_1 \cdot_p 2^{32}) +_p r_p(y_1y_2)$.
Do wyznaczenia pośrednich wartości r_p wykorzystywany jest algorytm 1. Algorytm mnożenia pośredniego działa analogicznie do algorytmu 2. Różnicą jest przedstawienie czynników jako $x \cdot 2^{16} + y$.

Algorithm 1: Mnożenie pośrednie, `halfMult`

Input: a, b - dwie liczby całkowite, p - modulnik
Output: *result* - wynik mnożenia

1 $x_1 \leftarrow a >> 16$
2 $y_1 \leftarrow a \;\&\&\; 0xffff$
3 $x_2 \leftarrow b >> 16$
4 $y_2 \leftarrow b \;\&\&\; 0xffff$
5 $half_a \leftarrow (((x_1x_2)\%p) \cdot r_p(2^{32}))\%p$
6 $half_b \leftarrow (((x_1y_2)\%p) \cdot r_p(2^{16}))\%p$
7 $half_c \leftarrow (((x_2y_1)\%p) \cdot r_p(2^{16}))\%p$
8 $half_d \leftarrow (y_1y_2)\%p$
9 **return** $(half_a + half_b + half_c + half_d)\%p$

Algorithm 2: Pełne mnożenie modularne dwóch liczb, `mult`

Input: a, b - dwie liczby całkowite, p - modulnik
Output: *result* - wynik mnożenia

1 $x_1 \leftarrow a >> 32$
2 $y_1 \leftarrow a \;\&\&\; 0xffffffff$
3 $x_2 \leftarrow b >> 32$
4 $y_2 \leftarrow b \;\&\&\; 0xffffffff$
5 $half_a \leftarrow (halfMult(x_1, x_2) \cdot r_p(2^{64}))\%p$
6 $half_b \leftarrow (halfMult(x_1, y_2) \cdot r_p(2^{32}))\%p$
7 $half_c \leftarrow (halfMult(x_2, y_1) \cdot r_p(2^{32}))\%p$
8 $half_a \leftarrow halfMult(y_1, y_2)$
9 **return** $(half_a + half_b + half_c + half_d)\%p$

2 Poszukiwanie relacji i faktoryzacja w bazie

2.1 Szybkie potęgowanie modularne

Metoda indeksu wymaga obliczenia wartości typu $a^b \ mod \ n$. Szybkie potęgowanie modularne jest prostym algorytmem pozwalającym zredukować liczbę mnożeń i dzieleń modulo z b do $O(log\ b)$.

Algorithm 3: Szybkie potęgowanie modularne, `fastPow`

Input : podstawa potęgi a, wykładnik potęgi b, modulnik n
Output: $a^b \ mod \ n$

```
1  bits ← to_bin(b)
2  nbits ← length(bits)
3  a ← a%n
4  result ← 1
5  x ← a
6  for i ← 0 to nbits do
7      if bits[i]==1 then
8          result ← result * x
9          result ← result%n
10     x ← x * x
11     x ← x%n
12 return result
```

2.2 Fakoryzacja w bazie

2.3 Budowa relacji

3 Eliminacja Gaussa w pierścieniu $\mathbb{Z}_{p-1}$

3.1 Algorytm Euklidesa

3.2 Rozszerzony algorytm Euklidesa