Politechnika Wrocławska

Wydział Elektroniki
Kierunek Informatyka

Architektura komputerów: Mnożenie liczb stałoprzecinkowych wymiernych w procesorze RNS

Konrad Stręk 248900, czw. TP 17:05
Maksim Birszel 241353, czw. TN 18:55

Prowadzący:
Dr inż. Piotr Patronik

Wrocław 2020

Spis treści

Rozdział 1

Wprowadzenie

1.1 Cele projektu

- Analiza i implementacja mnożenia liczb wymiernych w systemie RNS
- Implementacja schematu mnożenia w zadanym procesorze RNS według istniejącego schematu

1.2 Wstęp

Mnożenie modularne jest niezwykle często wykorzystywaną operacją matematyczną. Ma ogromne zastosowanie w algorytmach kryptografii klucza publicznego. Wyróżniamy szereg algorytmów przystosowanych do implementacji w architekturze szeregowej:

- Algorytm klasyczny mnożenia modularnego
- Mnożenie modularne Montgomery'ego
- Mnożenie z redukcją Barretta

Po analizie każdego z algorytmów, wybór padł na algorytm mnożenia modularnego Montgomery'ego.

Algorytm Montgomery'ego stanowi ulepszenie w stosunku do mnożenia przeplatanego z redukcją, ponieważ nie występuje tu propagacja przeniesień. Algorytm przetwarza dane bit po bicie, dzięki czemu nadaje się do implementacji szeregowej. [5]

1.3 Redukcja Montgomery‘ego

Redukcja Montgomery‘ego, powszechniej nazywana mnożeniem Montgomery‘ego jest metodą służącą do wykonywania szybkiego modułowego mnożenia.
Została wynaleziona w 1985 roku przez amerykańskiego matematyka Peter‘a L. Montgomery‘ego.

W porównaniu do klasycznego algorytmu modułowego mnożenia, jest on dużo szybszy i o wiele bardziej wydajny, ponieważ nie występuje z nim propagacja przeniesień. Dzięki temu nie wykonujemy wielu kosztownych porównań, które występują w algorytmie klasycznym.[6,5]

Rozdział 2

Podstawy matematyczne

2.1 Wstęp

System resztowy (RNS od ang. residue number system) – system liczbowy służący do reprezentacji liczb całkowitych wektorem reszt z dzielenia względem ustalonego wektora wzajemnie względnie pierwszych modułów.

Na liczbach reprezentowanych w systemie resztowym może być efektywnie przeprowadzonych wiele operacji arytmetycznych. Wykonuje się je, przeprowadzając niezależnie na odpowiednich resztach „zwykłe" operacje, a następnie operację modulo odpowiedniego modułu.[7,5]

W ramach implementowanego algorytmu mnożenia korzystamy z szeregu innych algorytmów opisanych poniżej.

2.2 Odwrotność multiplykatywna

Odwrotność multiplikatwna(z ang. multiplicative inverse) lub odwrotność modularna liczby całkowitej a to taka liczba x, dla której iloczyn ax przystaje do 1 modulo m, co możemy symbolicznie zapisać jako:

$$ax \cong 1\ (mod\ m)$$

Aby obliczyć odwrotność modularną można zastosować podejście naiwne, czyli dla kolejnych liczb sprawdzamy czy iloczyn ax przystaje do 1 modulo m, ale można także skorzystać z rozszerzonego algorytmu Euklidesa (2.3.1), który jest wydajniejszym rozwiązaniem.

Odwrotność modularna została wykorzystana w redukcji Montogomery'ego (2.5).[1,2,10]

2.3 Algorytm Euklidesa

Algorytm Euklidesa służy do wyznaczenia największego wspólnego dzielnika(NWD) dwóch liczb całkowitych. Jest to szczególnie przydane, gdy chcemy sprawdzić czy dane liczby a i b są względnie pierwsze. W takim przypadku zachodziła by równość:

$$NWD(a, b) = 1$$

NWD może zostać wyznaczone za pomocą prostego algorytmu używającego operacji modulo:

function NWD(a, b)
    while $b \neq 0$ **do**
        $c \leftarrow a\%b$
        $a \leftarrow b$
        $b \leftarrow c$
    end while
    return a
end function

Niemniej jednak, aby wyliczyć odwrotność modularną, potrzebna jest rozszerzona wersja algorytmu.[1,3]

2.3.1 Rozszerzony algorytm Euklidesa

Podobnie jak podstawowa wersja algorytmu, wersja rozszerzona wyznacza $NWD(a, b)$, ale rozwiązuje także równanie diofantyczne postaci:

$$ax + by = NWD(a, b)$$

Jeżeli wiemy, że liczby a oraz b są względnie pierwsze($NWD(a, b) = 1$), możemy wykonać kilka przekształceń, które pozwolą nam określić odwrotność modularną. [10]

$$ax + my = 1$$

$$ax + my \cong 1 \ (mod\ m)$$

$$ax \cong 1 \ (mod\ m)$$

A więc ostatecznie dostajemy równanie, którym zdefiniowana jest odwrotność modularna.

2.4 Chińskie twierdzenie o resztach

Chińskie twierdzenie o resztach mówi o tym, że znając znając reszty z dzielenia danej liczby a przez liczby z bazy $M = \{m_1, m_2, ..., m_n\}$ jesteśmy w stanie jednoznacznie wyznaczyć liczbę a, pod warunkiem, że liczby z bazy M są względnie pierwsze. Oznacza to, że liczbę a możemy przedstawić jako wektor $< a_1, a_2, ..., a_n >$, gdzie:

$$a_n = a\ (mod\ m_n)$$

Maksymalna liczba wartości jakie możemy wyznaczyć w ten sposób jest równa M_{max}, gdzie:

$$M_{max} = m_1 \cdot m_2 \cdot ... \cdot m_n$$

2.4.1 Konwersja z systemu resztowego na pozycyjny

Aby dokonać konwersji z systemu resztowego wykorzystywane jest równanie [8]:

$$a = \left(\sum_{i=1}^{n} \hat{m}_i \cdot (\hat{m}_i^{-1}\ mod\ m_i) \cdot a_i \right)\ mod\ M_{max}$$

gdzie:

$$\hat{m}_i = \prod_{j=1}^{n} m_j \text{ dla } j \neq i$$

2.5 Redukcja Montgomery'ego

Szukamy wartości mnożenia:

$$ABmodN$$

Działanie algorytmu Montgomery'ego opiera się na specjalnej reprezentacji liczb A i B. Liczba w RNS składa się ze zbioru argumentów (modułów) i zbioru reszt z dzielenia wartości tej liczby przez moduły.
Pierwszym krokiem jest znalezienie liczb, które będą do siebie względnie pierwsze:

$$m_{k-1}, m_{k-2}, ..., m_1, m_0$$

$$m_{k-1} > m_{k-2} > ... > m_1 > m_0$$

Przykładem takich liczb będą np. 7,5,3.[4]
Kolejnym krokiem jest znalezienie liczby całkowitej R takiej, że:

$$R >= N$$

oraz

$$NWD(N, R) = 1$$

W praktyce zawsze wybieramy R będące wielokrotnością 2^m, pasującym do poprzednich założeń.
Kolejnym krokiem jest znalezienie odwrotności multiplykatywnej N^{-1}.
Możemy do tego wykorzystać klasyczną, wolną metodę lub wspomniany wyżej algorytm Euklidesa, który został zostosowany przez nas.
W obecnym momencie posiadamy do dyspozycji następujące zmienne:

- liczby A i B przedstawione jako bazy względnie pierwszych modułów
- liczba całkowita N przedstawiona jako baza względnie pierwszych modułów
- liczba całkowita R przedstawiona jako baza względnie pierwszych modułów
- liczba całkowita N^{-1} przedstawiona jako baza względnie pierwszych modułów

W kolejnym kroku dla każdej z liczb z bazy A oraz B obliczamy:

$$A^{'} = A * RmodN$$

$$B^{'} = B * RmodN$$

Z tak obliczonych nowych baz modułów wyliczamy:

$$C^{'} = (A^{'} * B^{'} + A^{'}B^{'}N^{-1}modR * N)/R$$

Następnie obliczoną przed chwilą liczbę wykorzystajemy w obliczeniach:

$$C = (C^{'} + CN^{-1}modR * N)/R$$

Przy czym C to nasz ostateczny wynik.[11,7,5]

2.6 Wnioski

Wybrane i opisane przez nas algorytmy opierają się głównie na pętlach, instrukcjach wyboru, porównaniach oraz na operacjach dodawania, mnożenia i przesunięciach bitowych.
Algorytm Montgomery'ego wymaga wykorzystania wielu krótkich oraz stosunkowo prostych działań matematycznych, co czyni go bardzo szybkim oraz skutecznym.
Z drugiej jednak strony, jakikolwiek błąd we wcześniejszych algorytmach będzie skutkował błędnym wynikiem w ostatecznym algorytmie.

Rozdział 3

Implementacja

3.1 Wstęp

W ramach projektu wykorzystany został język programowania C++. Głównym elementem programu jest algorytm redukcji Montgomery'ego. Został on zaimplementowany na podstawie algorytmu przedstawionego w źródle [11].

3.2 Sposób implementacji

Na początku programu zdefiniowane są stałe tablice *baseN*(zbiór modułów) oraz *baseR*, które stanowią podstawę do użycia chińskiego twierdzenia o resztach oraz redukcji Montgomery'ego. Są one wypełnione w kodzie, aby uniknąć potrzeby sprawdzania ich poprawności.

Zaimplementowane funkcje operują przekazując między sobą tablice, wykonując za jednym razem krok algorytmu dla wszystkich pomniejszych elementów. Poniżej przykładowy kod - funkcja mnożąca dwie liczby:

```
int rns_montgomery_reduction(int a, int b){
        int * inversed_N = inverse_multiplicative(baseN,
            baseR);
        int * a_prim = compute_input_prim(a, baseR, baseN);
        int * b_prim = compute_input_prim(b, baseR, baseN);
        int * result_prim = compute_result_prim(a_prim,
            b_prim, baseN, inversed_N, baseR);
        int * result = compute_result(result_prim, baseN,
            inversed_N, baseR);
        int converted_result = convert_from_rns(result);
        return converted_result;
}
```

3.3 Wnioski

Wybierając jako język programowania c++ zachowaliśmy prostotę programu, jednocześnie zachowując umiarkowaną wydajność. Nie mniej jednak jest to jedynie prezentacja działania algorytmu, który w takiej formie nie mnoży liczb równolegle, lecz szeregowo.

Testując program zrozumieliśmy także, że taka implementacja ma inne wady. Na przykład nie jesteśmy wstanie wybrać takiej bazy modułów, aby obsługiwała dokładnie taki zakres jak 4-bajtowa liczba. Jeżeli zdecydujemy się na mniejszą bazę, zakres wartości jest mniejszy. W przeciwnym wypadku pewna część zakresu pozostaje nie wykorzystana.

Ważny jest także typ danych, na którym wykonywane są obliczenia. W niektórych miejscach wyniki obliczeń przekraczały zakres 4-bajtowej liczby, w związku z czym czasami konieczne było użycie 8-bajtowych liczb jako zmiennych pomocniczych.

Rozdział 4

Wnioski

Redukcja Montgomery'ego, powszechnie nazywana mnożeniem Montgomery'ego wykorzystuje wiele "zwykłych" operacji arytmetycznych połączonych z operacją modulo odpowiednich modułów.
Przekłada się to na możliwość wykorzystania w niej innych, powszechnych algorytmów wykorzystywanych w systemach resztowych.
Dzięki temu, samo mnożenie może być wykonane na wiele różnych sposobów, z wykorzystaniem różnych kombinacji algorytmów pośrednich.
W wybranej przez nas implementacji postawiliśmy na osiągnięcie jak największej wydajności dzięki następująym krokom:

- obliczanie odwrotności multiplikatywnej za pomocą algorytmu Euklidesa, zamiast metody klasycznej,
- wykorzystanie 9 liczbowej bazy względnie pierwszych modułów (bazę można zmienić ręcznie),
- konwersję wszystkich zmiennych stałych (A, B, N, R, N^{-1}) na bazy względnie pierwszych modułów, co pozwala na wykonywanie obliczeń na wielu mniejszych liczbach,
- wykorzystanie dobrze zoptymalizowanego języka C++ oraz pracę na wskaźnikach (nie używamy zewnętrznych bibliotek jak np. vector),

Dzięki zastosowaniu konwersji liczb na bazy względnie pierwszych modułów, wszystkie obliczenia z poszczególnych etapów można wykonywać równolegle (np. na wielu różnych wątkach procesora lub maszynach rozproszonych).
Ostateczny wynik, na potrzeby wyświetlenia użytkownikowi w formie, w jakiej wprowadził dane do programu jest zamieniany dzięki algorytmowi konwersji odwrotnej.

Bibliografia

[1] E. B. Olsen, *Residue number arithmetic logic unit*. July 14 2015, US Patent 9,081,608.

[2] E. B. Olsen, *System and method for improved fractional binary to fractional residue converter and multipler*. July 18 2017, US Patent 9,712,185.

[3] Piotr Patronik, Stanisław J. Piestrak, *Hardware/Software Approach to Designing Low-Power RNS-Enhanced Arithmetic Units*. May 2017, ieee on circuits and systems-i: regular papers, vol. 64, no. 5.

[4] Daniel Anderson, *Design and implementation of an instruction set architecture and an instruction execution unit for the rez9 coprocessor system*. 2011, Bachelor of Science in Computer Engineering University of Nevada Las Vegas.

[5] mgr inż. Jakub Piotr Olszyna, *Analiza i projektowanie układów kryptograficznych przeznaczonych do sieci czujnikowych*. Warszawa 2013, Politechnika Warszawska Wydział Elektroniki i Technik Informacyjnych.

[6] Gang Qu, *Course in the Cybersecurity Specialization - Hardware Security*. University of Maryland, College Park.

[7] *rosettacode.org/wiki/Montgomery_ reduction*

[8] *wikipedia.org/wiki/Systemresztowy*

[9] *michalp.net/open/AK/resztowe11.pdf*

[10] *https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/*

[11] *https://www.coursera.org/lecture/hardware-security/montgomery-reduction-DgleF?fbclid=IwAR19A3xjYGtOjKWoCnHHHVd3T8ZLSg1qUaHdCUH-wTqaLs8FiKlZyDxZO0Y*