CENTRUM MODELOWANIA MATEMATYCZNEGO

Kryptografia na procesorach wielordzeniowych

Andrzej Chmielowiec

26 maja 2008

Streszczenie

Artykuł opisuje możliwości implementowania zrównoleglonych algorytmów kryptograficznych na procesorach wielordzeniowych. Temat ten wydaje mi się bardzo interesujący w kontekście ostatnich zapowiedzi takich firm jak Intel i AMD. Wynika z nich, że zwiększanie liczby równolegle pracujących rdzeni procesora jest tendencją stałą. Taka architektura procesorów wiąże się jednak z koniecznością stosowania algorytmów, które przetwarzają dane w sposób równoległy. W związku z tym aplikacje kryptograficzne, które chcą w pełni wykorzystywać dostępną moc obliczeniową muszą być zmodyfikowane właśnie pod kątem zrównoleglenia obliczeń.

Słowa kluczowe: kryptografia, algorytmy równoległe, przetwarzanie współbieżne, przetwarzanie wielowątkowe, procesory wielordzeniowe, szyfrowanie równoległe

1 Wprowadzenie

Aktualne tendencje rozwoju procesorów pokazują szybki wzrost liczby jednostek pracujących równolegle. Niestety zastosowanie dwukrotnie większej liczby procesorów nie przekłada się w sposób bezpośredni na zwiększenie wydajności aplikacji lub bibliotek. Przygotowanie oprogramowania przystosowanego do pracy na kilku procesorach nie jest na ogół rzeczą automatyczną. Z najprostszą sytuacją mamy do czynienia w przypadku usług obsługujących dużą liczbę użytkowników. W tym przypadku zrównoleglenie uzyskuje się w sposób naturalny poprzez uruchomienie kilku identycznych procesów lub wątków. Na tym jednak kończy się możliwość automatycznego zwiększenia wydajności istniejącego oprogramowania. Jeśli bowiem chcemy efektywnie obsługiwać pojedynczego użytkownika, to należy w taki sposób rozdzielić zadania, aby później istniała możliwość szybkiego złożenia otrzymanych wyników cząstkowych w ostateczny rezultat. Wymaga to jednak ingerencji w istniejący już kod i mechanizmy wykonujące obliczenia.

Można się oczywiście zastanawiać po co pojedynczemu użytkownikowi wydajność większa niż ta, którą w aktualnie oferuje mu pojedynczy procesor. Z całą pewnością w tej chwili nie znajdzie to większego zastosowania. Mając jednak na uwadze przytoczone poniżej wypowiedzi, nie powinniśmy się zbytnio sugerować obecną sytuacją.

Światowe zapotrzebowanie szacuję na około pięć komputerów.

Uwaga przypisywana Thomasowi Watsonowi,
prezesowi IBM, 1943

640 kilobajtów powinno każdemu wystarczyć.

Uwaga przypisywana Bilowi Gatesowi,
prezesowi Microsoft, 1981

Przedstawione cytaty pokazują, że szacowanie zapotrzebowania na różnego rodzaju zasoby może być obarczone dużym błędem. W związku z tym myślę, że już teraz warto poświęcić trochę czasu na zapoznanie się z tematem równoległego przetwarzania danych, nie tylko w kontekście kryptografii i szyfrowania.

2 Algorytmy asymetryczne

Mechanizmy kryptografii asymetrycznej cechują się dużą złożonością wykorzystywanych algorytmów. Takie działania jak potęgowanie modularne, wyznaczanie krotności punktu, czy parowanie punktów krzywej eliptycznej są zazwyczaj bardzo czasochłonne i pochłaniają dużo zasobów procesora. Możliwość rozproszenia tego typu obliczeń może więc prowadzić do istotnej poprawy wydajności całego systemu. Zastosowanie zrównoleglonych algorytmów jest w niektórych przypadkach wręcz koniecznością, jeśli chcemy w pełni wykorzystać możliwości oferowane przez dany procesor lub mikrokontroler.

2.1 Potęgowanie modularne dla RSA

Operacja ta stanowi podstawę prawdopodobnie najbardziej rozpowszechnionego w tej chwili algorytmu, jakim jest RSA. Zanim przejdziemy do konkretnych metod zrównoleglania tego mechanizmu, przypomnijmy krótko na czym polega idea samego algorytmu.

Opis algorytmu RSA Aby wykonywać obliczenia publiczne i prywatne musimy najpierw wygenerować klucze. W tym celu wykonujemy następujące kroki:

1. znajdujemy dwie liczby pierwsze p i q, które mają zbliżoną liczbę bitów,
2. wyznaczamy $n = pq$ oraz liczby całkowite e i d takie, że $ed \equiv 1 \bmod (p-1)(q-1)$.

Po wygenerowaniu liczb możemy przyjąć, że pary (e, n) i (d, n) stanowią odpowiednio klucz publiczny i prywatny. Zaszyfrowanie wiadomości x polega wtedy na wykorzystaniu klucza publicznego (e, n) i wyznaczeniu liczby

$$c \equiv x^e \bmod n.$$

Do zdeszyfrowania wiadomości potrzebujemy klucza prywatnego (d, n), który pozwala na odtworzenie przesłanej wiadomości

$$c^d \equiv (x^e)^d \equiv x^{ed} \equiv x \bmod n.$$

W tym miejscu należy podkreślić kilka aspektów implementacyjnych RSA. Zasadniczo liczby p i q dobiera się tak, aby wykładnik publiczny e mógł mieć wartość $F_4 = 2^{2^4} = 65537$. Takie podejście znacznie przyspiesza wykonywanie operacji publicznej. W związku z tym istotne obciążenie dla algorytmu stanowi jedynie operacja prywatna. W ogólności bowiem wykładnik d ma podobną liczbę bitów, co moduł n. Zrównoleglenie potęgowania przy użyciu wykładnika d jest możliwe jeżeli tylko zastosujemy inną postać klucza prywatnego. Jeśli zapiszemy go w postaci (p, q, d_{p-1}, d_{q-1}), gdzie

$$d_{p-1} \equiv d \bmod p - 1,$$

$$d_{q-1} \equiv d \bmod q - 1,$$

to zasadniczą część obliczeń będziemy mogli wykonać w dwóch oddzielnych wątkach. Oznaczając procesory przez P1 i P2 otrzymujemy następujący algorytm

Algorytm:

P1 wyznaczamy $x_p \equiv c^{d_{p-1}} \bmod p$,

P2 wyznaczamy $x_q \equiv c^{d_{q-1}} \bmod q$,

P1 wykorzystując twierdzenie chińskie o resztach znajdujemy takie x, że

$$\begin{cases} x \equiv x_p \equiv c^{d_{p-1}} \bmod p, \\ x \equiv x_q \equiv c^{d_{q-1}} \bmod q. \end{cases}$$

Zaproponowana powyżej metoda jest naturalna i prowadzi do bardzo dobrych efektów. Ponieważ złożenie wyników cząstkowych na podstawie twierdzenia chińskiego o resztach jest bardzo szybkie, to w efekcie uzyskujemy niemal dwukrotny wzrost wydajności.

W tym miejscu warto zauważyć, że opisana powyżej metoda może być z powodzeniem stosowana również przy okazji innego typu zadań. Jeżeli musimy na przykład wykonać obliczenia na bardzo długich wielomianach lub szeregach potęgowych, to niezbędne działania można wykonać niezależnie modulo wiele małych liczb pierwszych. Następnie wykorzystujemy twierdzenie chińskie o resztach w celu odzyskania wyniku całościowego. Oczywiście zastosowanie tej metody bardzo zależy od konkretnego problemu, ale warto o niej pamiętać podczas projektowania algorytmu równoległego.

2.2 Podwajanie i dodawanie punktów krzywej eliptycznej

W przypadku krzywych eliptycznych dobór odpowiedniego układu współrzędnych pozwala na wykonywanie niektórych operacji arytmetycznych jednocześnie. Daje to możliwość wykorzystania takich mechanizmów procesora jak SIMD (Single Instruction Multiple Data) i znaczne przyspieszenie obliczeń. Zaproponowano już kilka tego typu rozwiązań. Jako przykład zaprezentujemy tutaj jedną z najnowszych metod, która jest dedykowana dla 3 procesorów. Zastosowano w niej reprezentację punktów, która bazuje na sześciu współrzędnych $(X, Y, Z, X^2, Z^2, Z^3/Z^4)$. Przy czym ostatnia współrzędna jest równa Z^3 lub Z^4 w zależności od tego jakie działanie będzie wykonywane po podwojeniu punktu. Ostatni warunek nie stanowi żadnego istotnego utrudnienia. Dzieje się tak dlatego, że podczas wyznaczania krotności punktu wiadomo jakie działania i w jakiej kolejności będzie trzeba wykonać.

Zaprezentowane poniżej formuły są przeznaczone dla ciał charakterystyki $p > 3$ i zostały zoptymalizowane pod kątem krzywych, dla których współczynnik $a = -3$.

Podwajanie punktu

Załóżmy, że mamy dany punkt $P_1 = (X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^4)$ i chcemy znaleźć jego podwojenie w postaci

- $P_3 = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^4)$ jeżeli kolejnym działaniem będzie również podwojenie,
- $P_3 = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^3)$ jeżeli kolejnym działaniem będzie dodawanie.

Kolejne współrzędne X_3, Y_3, Z_3 punktu P_3 mają w takiej sytuacji postać

$$\begin{aligned} X_3 &= \alpha^2 - 2\beta, \\ Y_3 &= \alpha(\beta - X_3) - 8Y_1^4, \\ Z_3 &= (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2, \end{aligned}$$

gdzie

$$\begin{aligned} \alpha &= 3(X_1^2 - Z_1^4), \\ \beta &= 2\left((X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\right). \end{aligned}$$

Tablica 1 zawiera kolejność wykonywania poszczególnych operacji mnożenia i kwadratowania, które prowadzą do wyznaczania reprezentacji punktu P_3. Takie działania jak dodawanie i odejmowanie zostały w tym zestawieniu pominięte ze względu na swój niski koszt.

Dodawanie punktu w postaci afinicznej

Załóżmy, że mamy dane punkty $P_1 = (X_1, Y_1, Z_1, X_1^2, Z_1^2, Z_1^3)$, $P_2 = (X_2, Y_2)$ i chcemy znaleźć ich sumę postaci $P_3 = (X_3, Y_3, Z_3, X_3^2, Z_3^2, Z_3^4)$. Kolejne współrzędne X_3, Y_3, Z_3 punktu

Tablica 1: Podwajanie punktu wykonywane na 3 procesorach.

Operacja	Procesor P1	Procesor P2	Procesor P3
Kwadratowanie	α^2	$(Y_1 + Z_1)^2$	Y_1^2
Kwadratowanie	Y_1^4	Z_3^2	$(X_1 + Y_1^2)^2$
Mnożenie	X_3^2	$\alpha \cdot (\beta - X_3)$	Z_3^3 lub Z_3^4

P_3 mają w takiej sytuacji postać

$$\begin{aligned} X_3 &= \alpha^2 - 4\beta^3 - 8X_1\beta^2, \\ Y_3 &= \alpha(4X_1\beta^2 - X_3) - 8Y_1\beta^3, \\ Z_3 &= (Z_1 + \beta)^2 - Z_1^2 - \beta^2, \end{aligned}$$

gdzie

$$\begin{aligned} \alpha &= 2(Z_1^3 Y_2 - Y_1), \\ \beta &= Z_1^2 X_2 - X_1. \end{aligned}$$

Tablica 2 zawiera kolejność wykonywania poszczególnych operacji mnożenia i kwadratowania, które prowadzą do wyznaczania reprezentacji punktu P_3. Podobnie jak w przypadku podwojenia punktu dodawanie i odejmowanie zostały w tym zestawieniu pominięte ze względu na swój niski koszt.

Tablica 2: Dodawanie punktów wykonywane na 3 procesorach.

Operacja	Procesor P1	Procesor P2	Procesor P3
Mnożenie	$Z_1^3 \cdot Y_2$	$Z_1^2 \cdot X_2$	—
Kwadratowanie	α^2	$(Z_1 + \beta)^2$	β^2
Mnożenie	$4X_1 \cdot \beta^2$	$4\beta \cdot \beta = 4\beta^3$	$2\beta \cdot Y_1$
Kwadratowanie	X_3^2	—	Z_3^2
Mnożenie	$4Y_1\beta \cdot 2\beta^2 = 8Y_1\beta^3$	$\alpha \cdot (4X_1\beta^2 - X_3)$	Z_3^4

2.3 Krotność punktu krzywej eliptycznej i potęgowanie modularne w przypadku ogólnym

Wyznaczanie krotności punktu lub wykonywanie potęgowania modularnego można również zrównoleglać. Niestety w tym przypadku nie można osiągnąć tak dobrych rezultatów, jak w przypadku operacji prywatnej RSA, czy podwajania i dodawania punktów krzywej eliptycznej.

Niemniej jednak warto nieco uwagi poświęcić również tym metodom.

Zasadniczym elementem, który daje możliwość przyspieszenia operacji jest dysproporcja pomiędzy czasem potrzebnym na dodawanie i podwojenie punktów (mnożenie i kwadratowanie). Zasadniczo bowiem tą drugą operację można wykonać szybciej. Dla ustalenia uwagi skupimy się w dalszej części jedynie na wyznaczaniu rezultatu potęgowania modularnego, choć przedstawione metody mają również zastosowanie do obliczania krotności punktu krzywej eliptycznej.

Załóżmy zatem, że interesuje nas podniesienie pewnej losowej liczby do potęgi k, która jest liczbą naturalną o n bitach. Niech liczba ta ma w zapisie binarnym postać

$$k = \sum_{i=0}^{n-1} k_i 2^i.$$

Naszym zadaniem jest takie rozdzielenie obliczeń, aby dwa procesory wykonały to potęgowanie szybciej niż jeden z nich. Aby osiągnąć nasz cel skorzystamy z następującej własności potęgowania. Jeżeli $k = r + s$, to $x^k = x^r x^s$.

Algorytm:

P1 wyznaczamy $x_r = x^r$,

P2 wyznaczamy $x_s = x^s$,

P1 wyznaczamy $y = x_r x_s$.

Teraz musimy tylko znaleźć takie liczby r i s dla których wykonanie operacji x^r i x^s będzie szybsze niż x^k. Załóżmy, że mnożenie modularne jest $\beta > 1$ razy wolniejsze od kwadratowania, a niezerowy bit w reprezentacji liczby k pojawia się z prawdopodobieństwem α. Okazuje się, że jeśli odpowiednio dobierzemy indeks m, to liczby

$$\begin{aligned} r &= \sum_{i=0}^{m-1} k_i 2^i, \\ s &= \sum_{i=m}^{n-1} k_i 2^i, \end{aligned}$$

pozwolą nam na szybsze wykonanie potęgowania. Zauważmy, że liczba r ma jedynie $m < n$ bitów. Oznacza to, że wykonanie potęgowania x^r powinno być szybsze niż x^k. Liczba s ma co prawda n bitów, ale najmłodsze m z nich ma wartość 0, co eliminuje konieczność wykonywania mnożeń w pierwszej fazie obliczeń. Powstaje zatem pytanie w jaki sposób wyznaczyć wartość indeksu m, aby jak najszybciej znaleźć wynik potęgowania. Odpowiedź na to pytanie jest

oczywista. Wystarczy tak dobrać liczbę m, aby operacje x^r i x^s trwały tyle samo. Korzystając ze współczynników α i β otrzymujemy następujące równanie

$$T_r = \alpha\beta m + m = \alpha\beta(n-m) + n = T_s$$

$$m(1+2\alpha\beta) = n(1+\alpha\beta)$$

$$m = n\frac{1+\alpha\beta}{1+2\alpha\beta}.$$

Jeśli przyjąć, że dla naszego przypadku $\alpha = 0.5$ (co drugi bit jest niezerowy) i $\beta = 1.33$, to uzyskujemy $m = 0.71 \cdot n$. Wnioskujemy zatem, że potęgowanie będzie wykonane około 30% szybciej.

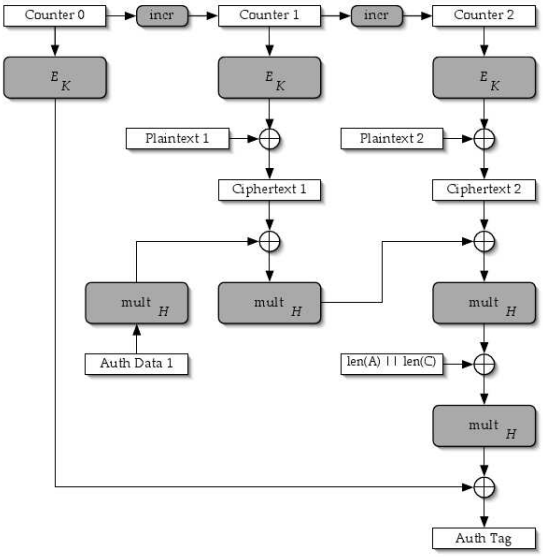
Efektywność tego algorytmu w istotny sposób zależy od parametrów α i β. Istnieją jednak takie struktury algebraiczne, w których opisana wyżej metoda jest bardzo efektywna i daje praktycznie 50% spadek czasu obliczeń. Z taką sytuacją mamy do czynienia w ciele charakterystyki 2, którego elementy reprezentowane są w bazie normalnej. W przypadku tych ciał mnożenie jest o wiele bardziej skomplikowane niż kwadratowanie, które polega na rotowaniu bitów reprezentacji.

Przedstawiona powyżej metoda może być również zastosowana dla większej liczby procesorów. Należy jednak podkreślić, że czas pracy takiego algorytmu nie będzie mniejszy niż wykonanie n operacji kwadratowania.

3 Algorytmy symetryczne i tryby szyfrowania

W przypadku algorytmów zrównoleglanie operacji byłoby bardzo proste, gdyby nie fakt, że szyfr pracuje zazwyczaj w pewnym trybie. Oczywiście najbardziej odpowiednim z punktu widzenia przetwarzania równoległego jest tryb ECB (Electronic Code Book), w którym kolejne kryptogramy nie są w żaden sposób ze sobą powiązane. Niestety jego minusem jest to, że zazwyczaj nie jest wykorzystywany w praktyce. Związane jest to z pewnymi słabościami tego trybu, które ułatwiają przeprowadzenie niektórych ataków na kryptosystem. Po drugiej stronie mamy tryb CBC (Cipher Block Chaining), który jest aktualnie jednym z najpowszechniej stosowanych trybów pracy. Ten jednak zupełnie nie nadaje się do zrównoleglenia. Jego unikalną cechą jest bowiem to, że aby zaszyfrować kolejny blok musimy mieć poprzedni szyfrogram. Stosowanie tego trybu skazuje nas więc na konieczność sekwencyjnego przetwarzania i wykorzystanie tylko jednego układu. Na szczęście istnieje jeszcze tryb GCM (Galois Counter Mode), który zyskuje coraz większą popularność i może być w relatywnie prosty sposób zrównoleglony. Rysunek 1 przedstawia schemat pracy tego trybu.

Rysunek 1: Schemat trybu szyfrowania GCM (Galois Counter Mode)



Łatwo zauważyć, że do zaszyfrowania bloku wiadomości potrzebujemy tylko wartości licznika i bloku tekstu jawnego. Daje to możliwość równoległego szyfrowania poszczególnych bloków wiadomości. Po zaszyfrowaniu musimy jedynie wygenerować kod uwierzytelniający, co jest o wiele szybsze niż szyfrowanie.

Tryb GCM nie jest oczywiście jedynym, który daje możliwość zrównoleglenia operacji szyfrującej. Podczas wyboru odpowiedniego mechanizmu należy jednak pamiętać, że niektóre z nich nadają się jedynie do przetwarzania sekwencyjnego. Należy ich unikać w przypadku takich implementacji jak:

1. utrzymywanie tunelu szyfrującego o dużej przepustowości (1Gb i więcej),
2. programy szyfrujące dyski i macierze dyskowe.

Poniżej znajduje się zestawienie najpopularniejszych trybów szyfrowania. Znakiem + oznaczono te, które można zrównoleglić:

+ ECB (Electronic Code Book),
– CBC (Cipher Block Chaining),
– OFB (Output Feedback),

– CFB (Cipher Feedback),

\+ CTR (Counter),

\+ GCM (Galois Counter Mode).

4 Narzędzia wspierające implementację algorytmów równoległych

Istnieje wiele przenośnych mechanizmów, które wspierają implementację algorytmów równoległych. W tej części opiszemy te, które mogą być istotne z punktu widzenia procesorów wielordzeniowych. Należy się też spodziewać, że w najbliższych latach nastąpi dalszy rozwój tych technologii. Niektóre z nich są dobrze znane osobom przygotowującym przenośne oprogramowanie serwerowe. Tutaj doskonałym przykładem jest biblioteka POSIX threads.

4.1 Biblioteka POSIX threads

Jest to standardowa implementacja mechanizmów wspierających przetwarzanie wielowątkowe. Choć dedykowana jest dla systemów klasy UNIX, to ma też swoją nieoficjalną implementację dla systemu Windows. Daje to możliwość tworzenia dość uniwersalnego i przenośnego oprogramowania, które będzie działać na wielu platformach. Biblioteka ta nie wspiera jednak w bezpośredni sposób implementacji algorytmów równoległych. Jest jedynie interfejsem umożliwiającym jednoczesne uruchamianie zadań. W związku z tym całość pracy nad przygotowaniem odpowiedniej kolejności przetwarzania spoczywa na programiście.

4.2 Przemysłowy standard OpenMP

Jest to zestaw dyrektyw preprocesora i bibliotek. Stanowią one swoiste API umożliwiające wydzielenie tych części programu, które mają być przetwarzane równolegle. W zamierzeniu autorów, standard ten ma być intuicyjny i uwalniać programistę od konieczności zapoznawania się z mechanizmami tworzenia i zarządzania wątkami. Minusem tego rozwiązania jest to, że musi być ono wspierane przez kompilator. Rozwiązanie przeznaczone jest dla kompilatorów języka C/C++ i Fortran.

4.3 Biblioteka Intel Ct

Technologia rozwijana przez firmę Intel. W zamierzeniu ma to być zestaw bibliotek języka C/C++, które będą wspomagały tworzenie oprogramowania działającego współbieżnie. Aktualnie firma Intel inwestuje duże środki w badania i rozwój technologii wspierających równoległe przetwarzanie danych.

5 Podsumowanie

W artykule krótko zostały opisane możliwości zrównoleglania mechanizmów kryptograficznych. Jest to w gruncie rzeczy temat bardzo obszerny. Dlatego też zaprezentowane rozwiązania należy traktować jedynie jako zarys tendencji panujących nie tylko w dziedzinie współczesnej

kryptografii obliczeniowej, ale i teorii obliczeń jako takiej. Tematem i algorytmami przetwarzania równoległego warto zająć się już teraz, gdyż najbliższe lata przyniosą najprawdopodobniej dalszy błyskawiczny rozwój tej dziedziny.