公開鍵暗号のための多倍長整数計算アルゴリズム

1C－5

太田　昌孝、　前野　年紀

東京工業大学　総合情報処理センター

1. はじめに

公開鍵暗号方式は、500~2000ビット程度の長さの整数計算を利用すると、実際上解読不可能な程度の難しさとなる。しかし、そのように長い整数の計算には、秘密情報を知っての暗号化／復号化にも、かなりの計算時間がかかる。特に、多倍長整数乗算の速度が全体の性能を決めることになるため、その高速化は重要である。多倍長整数乗算のアルゴリズムとして、単純な二乗時間のアルゴリズム、そのKaratsubaの方法による高速化、高速フーリエ変換による対数時間のアルゴリズム等が知られており、計算量のオーダーとしては後者ほど勝れているが、公開鍵暗号方式に必要な程度の長さの場合には、その差は微妙なものとなる。

そこで、最近の一般的なCPU上での実際のインプリメンテーションに関して、各種アルゴリズムの速度を比較、検討してみる。最近のCPUでは、レジスタ間での加算や乗算は高速であり、条件判断やデータの転送のほうに多くの時間をとられていることに注意する必要がある。

最近のCPUでは、各実行サイクルで、浮動小数点加算、浮動小数点乗算、を同時に実行できる。これに比べて整数の乗算は極めて遅いため、多倍長のデータは、適当なビット数に区切って複数の浮動小数点数にわけ、浮動小数点演算回路を利用して計算するのが効率的である。演算と並行して、さらに、浮動小数点のロードも各サイクルに1回実行できる。しかし、浮動小数点のストアに関しては、演算とは並行できるものの、ロードとは並行せずに2サイクルに1回しか行えないのが普通である。さらに多重度の大きなCPUでも、この演算とロード／ストアをの性能の比率はほぼ同様である。高速化のためにはレジスタを活用し、ロード／ストアを減らすことが重要である。公開鍵暗号のための多倍長整数計算ではそれほど大きなデータ量を扱わないので、すべてのデータは一次キャッシュに入ると考えてよい。

本稿では、こうした要因を考慮しつつ、公開鍵暗号のための多倍長整数計算アルゴリズムの高速なインプリメンテーションについて、クロック速度99MHz理論最大性能198MFLOPSのHP9000/755上で、640ビット、2000ビットの二つの桁数について、各種方法を比較考察する。一つの浮動小数点レジスタには20ビットを格納することとする。

2. 単純な二乗時間の方法

筆算方式に基づく単純な方法は、その計算時間は桁数の二乗に比例する。しかし、単純な方法であるだけに、ハイパフォーマンスコンピューティングの技法に基づく高速コーディング技術を効率的に適用可能である[1]。

キャリーの伝搬を抑制し、二重ループをレジスタブロッキング技法でK^2重に展開することにより、K^2回の浮動小数点加算乗算につき、2*K回のロード、K回のストアしか必要としない。K=4として、640ビットの場合中心となるループは理論最大性能の51%の20μ秒で実行できた。2000ビットの場合の時間は177μ秒であった。

後処理としてキャリーの伝搬が必要となるが、これは各アルゴリズムで共通であるため、評価対象より省く。

Multiprecision Integer Arithmetic Algorithms for Public Key Encryption. Masataka Ohta (mohta@necom830.cc.titech.ac.jp), Toshinori Maeno (tmaeno@cc.titech.ac.jp), Computer Center, Tokyo Institute of Technology.

3. Karatsubaの方法

Karatsuba の方法は、 (2^N*a+ b) * (2^N*c+ d) = 2^(2*N)*A+ 2^N*B+ C としたとき、A = a*c、C = b*d、B = a*d+ b*c = (a+ b)*(c+ d)-A-C となる性質を利用し、2*N桁の整数乗算を、N桁の整数乗算3回と、整数加減算4回で済ませようというものである。

Nが小さい場合は加減算の手間の追加のほうが大きいが、ある程度大きなNに対しては再帰的に適用可能であり、計算量はO(N^1.585) となる。[1]での技法と一段だけ組み合わせた場合、640ビットの乗算は、320ビットの乗算3回、320ビットの加算2回、640ビットの減算2 回に分解される。

320ビットの乗算1回は、ループが小さくなるため640ビットの場合の1/4より多めの6.4μ秒かかる。しかも、320ビットの加算1回には、16＊2回のロードと16回のストアが必要で、少なくとも64サイクルが必要となる。加減算全体に必要な時間は384サイクル(3.9 μ 秒)、合計時間は23.1 μ 秒で、Karatsubaの方法は有効ではない。

同様に、2000ビットを1000ビットに分割する場合も、1000ビットの乗算のための53μ秒と、加減算のための2400サイクルが必要で、合計183μ秒となり、Karatsubaの方法はやはり有効ではなかった。

4. FFT(高速フーリエ変換)による方法

たたみこみ定理によると、多倍長整数の上位桁に0を追加し2倍の桁数にし、フーリエ変換し、要素を個別に掛けて、逆フーリエ変換すれば、結果は整数の多倍長乗算となる。

そこで、FFTアルゴリズムを用いて高速多倍長乗算を行うことができる。FFTの構成要素はバタフライ演算と呼ばれる複素演算であり、浮動小数点演算に分解すると4回の浮動小数点乗算と6回の浮動小数点加減算となる。2^N要素のFFTは、N/2*logN回のバタフライ演算からなる。

640ビットの場合、0を追加して1280ビットに拡張後、64要素のFFTを正方向に2回、逆方向に1回行う。正方向のFFTは虚数部分が0なので、加減算と乗算をさらに128回追加すれば一つのFFTにまとめることができる。64要素のFFT二回の計算量は、バタフタイ演算にして384回、つまり1536回の乗算と2304回の加算である。さらに要素個別の複素乗算に加減算128回乗算256回が必要である。乗算と加算は完全に重ね合わせて実行できるものとし、その他の命令の実行時間は無視できるとしても、2560回の加算に必要なサイクルは25.9μ秒であり、単純な方法の20μ秒に及ばない。

同様に2000ビットの場合を考えるが、FFTは要素数が2 のべきでないと都合が悪いので、5120ビット(256要素)に0拡張し計算すると、演算量は、バタフライ演算2048回、浮動小数点加算13312回、134.5μ秒となる。2000ビットの単純な場合の実測値177μ秒より速く、2560ビットの場合の実測値276μ秒より約2倍速い。

とはいえ、理論最大浮動小数点演算性能の50%をだすことは、配列の乗算や多倍長整数計算のように極めて単純で高度に規則的な場合も困難を極め、FFTの場合にはまず不可能であろう。実際にある程度高速であろうと思われるコードでの実測値は、256要素の FFT一回あたり660μ秒と、理論最良値より10倍ほど遅かった。

5. 終りに

現状のCPU構成では、FFTの高速化に多少の余地はあるものの、公開鍵暗号系に必要な程度の桁数の多倍長整数計算のためには、単純なアルゴリズムを高速にコーディングした[1]の方式が最適である。

参考文献

[1] 「多倍長計算のHPC技術」, 太田　昌孝, 前野年紀 HPC 研究会 54-9, 情報処理学会, pp. 61-65, Dec. 1994.