ベクトル計算機による RSA 暗号ふるいの高速化

海洋研究開発機構・地球シミュレータセンター　後　保範(Yasunori Ushiro)
Earth Simulator Center
Japan Agency for Marine-Earth Science and Technology

1. はじめに

RSA 暗号はインターネットを使ううえで欠かせない技術である。だが、現在使用している 1024 ビットの RSA 暗号が近いうちに安全性の問題で使えなくなるという「2010 年問題」[2)]が懸念されている。RSA 暗号は多数桁の因数分解の困難性を利用している。現在、知られている手法では、1024 ビットの RSA 暗号の解読はスーパーコンでも数十年かかると予想されるが、計算機の高速化により数年先には、解読される危険性が高くなっている。RSA 暗号の因数分解には 2 次ふるい法(QS)と一般数体ふるい法(GNFS)がある。10 進 100 桁程度までは QS を拡張した複数多項式 2 次ふるい法(MPQS)が高速で、それ以上は GNFS[1)]が高速になる。2010 年1月に NTT 他 5 カ国の共同により、GNFS を用い Opteron2.2GHz 換算で約 1700 コア・年の演算量を使用して RSA-768(10 進 232 桁)が解読[5)]された。これまで、RSA 暗号解読はほとんど PC でおこなわれ、RSA-150, RSA-160, RSA-200, RSA-567, RSA-640 および RSA-768 の世界記録[8)]もすべて PC クラスタで実施されている。RSA 暗号解読の主力計算はふるい処理である。この計算は、PC で高速計算するため、キャッシュに収まる範囲の領域で行われる。ふるいは素数のサイズでメモリをアクセスする必要があり、大きな素数になると速度低下の問題が発生する。そこで、メモリアクセス性能がキャッシュサイズに依存しないベクトル計算機で、ふるいの高速化を試みた。

2. RSA 暗号

インターネットで情報を安全に送るために、暗号化が用いられている。暗号化する方式には、共通鍵方式(秘密鍵方式)と公開鍵方式があり、両方の利点を用いたハイブリッド方式が使用されている。共通鍵方式は高速処理できる利点があるが、暗号鍵と復号鍵が同じため、鍵が安全に送れない。一方、公開鍵方式は、暗号鍵と復号鍵が異なるため、暗号鍵を送付しても安全性が保たれる利点があるが、共通鍵方式に比べて処理速度が遅い。そのため、公開鍵方式を使用して、共通鍵方式の鍵を送付し、情

報本体は共通鍵方式で暗号化して送信する方法が使用されている。

現在使用されている公開鍵方式は、1978 年に R. L. Rivest, A. Shamir, L. M. Adlman の 3 名により発見され、RSA 暗号と呼ばれている。これは多数桁の因数分解は非常に時間がかかることを利用している。RSA 暗号は二つの大きな素数 P,Q 及び e をランダムに作成し、$n=P\times Q$、$F=(P-1)\times(Q-1)$を計算し、$D=e^{-1} \pmod F$を求め、n と e を暗号鍵として使用し、n と D を復号鍵に使用する。n と e は秘密にする必要はなく、公開鍵として自由に送信できる。一方、D は復号するための秘密鍵で、この鍵は送信しない。送付したい情報を n 以下の数値の列に分けて、それぞれを暗号化して送信し、受け取り側で復号して元にもどす。分けられた n 以下の数値を m とし、暗号文 C を $C=m^e \pmod n$で計算し送付する。受信側で元の数値 m に $m=C^D \pmod n$でもどす。元の数値への復号は二つの素数 P,Q にオイラーの定理、$M^{(P-1)(Q-1)}=1 \pmod n$を適用することで得られる。現在 n は 1024 ビット(10 進 309 桁)の値が使用され、P,Q はそれぞれ 510～514 ビット程度のランダムな素数が使用されている。

RSA 暗号を解読するには、合成数 n を二つの素数 P と Q に因数分解すればよい。多数桁の合成数を効率よく因数分解する方法には、ふるい法及び楕円曲線法が知られている。楕円曲線法による合成数の因数分解の計算量は、分解される因数の桁数に依存する。一方、ふるい法による因数分解の計算量は、合成数の桁数に依存する。RSA 暗号に使用される二つの素数の桁数はほぼ同じため、この因数分解には楕円曲線法よりふるい法の方が向いている。そのため、RSA 暗号解読に用いられる因数分解はふるい法が利用されている。

3. ふるい法

合成数を n とするとき、整数 a,b に対して $a^2 - b^2 = (a-b)(a+b) = 0 \pmod n$の関係を導いて n を因数分解する方法である。この関係式から a+b と n の最大公約数 $\gcd(a+b,n)$を求めると、n の因数となる。しかし、$\gcd(a+b,n)$は半分の確率で 1 や n となる場合があり、求める n の因数となる確率は半分である。そのため、より確かに n を因数分解するためにはこの関係式を多く求めておく必要がある。整数 a,b はふるいで得られたデータ(関係式)から、各基底(素数など)のベキ数を偶数にするデータを抽出し、これらを乗算することで得られる。

3.1　2 次ふるい法(QS)

合成数 n に対して、S を n^2 - S が正で $n^{1/2}$ に一番近く整数とし、$(S+k)^2-n=A_k$, k=0,1,…とする。また、p を指定数以下の素数とし、平方剰余 $x^2=n \pmod p$となる整数 x が存在する素数を求め、その集合を素数基底 B とする。次に、k=0,1,2,…と順に増やし、A_k が素数基底 B だけで分解できるものを、素数基底 B を構成する素数の数

より多く集める。この方法は 2 次ふるい法(QS, Quadratic Sieve)と呼ばれる。集めたデータは素数基底 B を構成する素数のベキ数の乗算で表わされる。構成する各素数のベキ数が全て偶数となるようなデータの組み合せを行列計算により求める。このときの行列の要素は法 2 のベキ数で構成され、全て 0 か 1 となる。データ数の方が、素数基底 B の素数の数より多いため、必ず偶数となる組み合わせが求められる。この組み合わせによる A_k の乗算は素数の乗算の二乗となる。従って、$a^2 - b^2 = 0 \pmod n$が得られ、確率 1/2 で n が因数分解される。より確実に因数分解するため、ふるいで集めるデータ数は、素数基底 B の素数の数より数百個多くする。ふるい処理は A_k を素数基底 B の各素数 p で割る代わりに、各 p に対し$(S+c)^2=n \pmod p$, $(0 =< c < p)$なる c(重根でない限り各 p に対し 2 個)を求め、$k=c+jp$, $j=0,1,2$ となる A_k が素数 p で割れることを利用して行う。この方法では k が大きくなると、A_k がほぼ比例して大きくなり、素数基底 B での分解が難しくなる。

2 次ふるい法での A_k の値をできるだけ小さくする方法として、複数多項式 2 次ふるい法(MPQS, Multiple Polynomial Quadratic Sieve)がある。適当な変換を行い複数の多項式を作る方法は数種類ある。代表的な変換として、$e^2 - n = 0 \pmod d$となる複数組の整数(d, e)と整数変数 x の 2 次式 f(x)を使用して、関係式$(dx+e)^2 - n=df(x)$を導出する方法である。現在、10 進 100 桁までの因数分解では最も高速な解法と言われている。

3.2 一般数体ふるい法(GNFS)

合成数 n に対して、$f(M)=0 \pmod n$を満たす整数多項式 f(x)と整数 M を求め、f(x)の根の一つを θ とする。このとき、$\alpha + \beta M$ を素数基底で分解し、$\alpha + \beta\ \theta$ を素元と単元で分解したものを集め、その分解の違いを利用する。例えば n=1333, $f(x)=x^3+2$, $f(M)=n$, M=11 の例では $2+M=13$ と $2+\theta = \theta \cdot \theta^3 = \theta(1-\theta)(1+\theta) \pmod{f(\theta)}$から $-10 \cdot 11 \cdot 12=13 \pmod n$の関係が得られる。これを、分解に利用して素数基底と生成元(素元と単元)の合計数以上集める。しかし、実際に生成元が得られるのは f(x)が特別な場合だけである。一般数体ふるい法(GNFS, General Number Field Sieve)では、素元で分解する代わりに対応する素イデアルで分解する。素イデアルの分解の原理は、d 次の多項式 f(x)において、イデアル $\alpha + \beta\ \theta$ に対し多項式ノルム $N(\theta)=|b^d f(-\alpha/\beta)|$ を定義し、これを素イデアル基底に対応する素数で分解する。素イデアル基底は素数 p に対して f(x)が $f(x)=(x-s)h(x) \pmod p$と分解できる場合に、p と s の組(p:s)で求める。素数基底の p と s の組(p:s)を使用して、$N(\theta)$が p で割れることと、$\alpha + \beta s$ が p で割れることは同じことが知られている。そのため、イデアル $\alpha + \beta\ \theta$ に対して、$\alpha + \beta M$ を素数基底の素数で分解でき、素イデアル基底の(p:s)の組を使用して $\alpha + \beta s$ が p で分解できるものを選ぶ。素イデアル側は更に、$\alpha + \beta\ \theta$ の平方剰余を複数個追加する。ふるいで集めるデータ数は、素数基底+イデアル基底+平

方剰余の合計数より数百多くする。これにより、素数基底側も、イデアル基底側も共にベキ数が偶数になる組み合わせを、行列計算により得られる。素数側はその組み合わせで単純に平方数となり、平方根 a が求められる。一方、イデアル側は f(x)の法の基に、d 次方程式の平方式になっているが、その代数平方根は別に求める必要がある。求めた代数平方根の変数に M を入れると平方根 b が得られる。両方から $a^2-b^2=0 \pmod n$が得られて、n が因数分解できる。

GNFS をより効率的に行うには、d 次多項式 f(x)の各係数の絶対値及び $f(M)=0 \pmod n$となる、M の値が小さい方が良い。d 次多項式で $f(M)=0 \pmod n$にするためには、M は $O(N^{1/d})$の大きさになる。最近、M を $O(N^{1/d})$の値にせず、t,u を $O(N^{1/d})$の値にして、tM+u=n になるようにし、d 次の多項式 f(x)と 1 次式 g(x)=tx+u の両方を素イデアル分解する方法が利用されてきている。このとき、$f(M)=0 \pmod n$で $g(M)=0 \pmod n$である。現在、10 進 100 桁以上になるとこの GNFS が最も高速になると言われている。

4. ふるい処理

ふるい処理は、MPQS においても、GNFS の素数ふるいでも、さらに GNFS の素イデアルふるいでも、すべて素数 p に対して C+k・p が素数 p で割れることを利用することに帰着する。言い換えると、多数の素数 p に対して、各初期値 C から素数 p 間隔で配列に、各素数で決まる値を乗算又は加算する処理が、ほとんどを占める。配列に入れる値は、普通に処理すれば、倍精度浮動小数点で表わした素数の乗算となり、その結果を評価値と比較し、評価値以上ならふるいデータとして採用することになる。しかし、大小を比較して採否を決めるだけなので、各素数及び評価値共に対数を取ることにすれば、乗算は加算に代わり、記憶する配列は倍精度浮動小数点から単精度浮動小数点又は固定小数点に変更可能である。そのため、ふるい処理の心臓部は FORTRAN で記載すると下記のようになる。

(a) キャッシュマシン用ふるい処理

```
      SUBROUTINE   SIEVE1(LP,JL,Base,LogP,N,PS,V,Sive,NS)
      IMPLICIT   INTEGER*4(A-Z)
      INTEGER*8 Sive, LLP
      DIMENSION   Base(2,N), LogP(N), Sive(N), PS(LP), V(LP)
C   Clear V                                        ＜ふるい領域の初期化処理＞
      LLP = LP
      LLP = LLP*JL                                 JL はサブルーチンが呼ばれた回数
      do 10 i=1,LP                                 LP は 1 回のふるいの長さ
        V(i) = 0                                   ふるい領域のゼロクリア
```

```
   10 continue
C  Sieve                                        ＜ふるい処理＞
      do k=1,N                                  N は基底の数
         IP   = Base(2,k)                       IP は増分値で素数
         IST = Base(1,k) + 1                    開始番地(IP で関数値が割れる)
         do 20 i=IST,LP,IP
            V(i) = V(i) + LogP(k)               対数処理された値、原理は乗算
   20    continue
      end do
C  Reset Base                                   ＜次のふるいの開始番地更新＞
      do 30 k=1,N
        Base(1,k) = mod(Base(1,k)-LP, Base(2,k))
   30 continue
C  Select
      do 40 i=1,LP                              ＜ふるいデータの採取＞
         if(V(i) .ge. PS(i)) then               基底値の累計が大きければ採用
            NS = NS + 1                         ふるいで得られたデータの合計
            Sive(NS) = i + LLP                  ふるいで得られたデータの番号
         end if
   40 continue
      RETURN
      END
```

ここで、LP は 1 回の処理におけるふるいの長さで、JL はふるい処理(サブルーチン)が呼ばれた回数を示し、N は基底の数を示す。基底は、小さい順に並べた素数か又はその中から選んだものである。Base は基底に対応した配列で、Base(1,k)は k 番基底の開始番地、Base(2,k)は k 番基底の値(素数)である。また LogP(k)は k 番基底の対数値(固定小数点用にスケール化)で、PS(i)は i 番目の評価値(固定小数点用にスケール化)である。V はワーク配列で、配列 Sive にはふるいで得られた累計番号が入り、NS はふるいで得られたデータの総数が入る。Base(1,k)は適用する解法に依存するが、いずれの解法でも k に依存する関数値が Base(2,k)で割れるようになっている。高速化のために、キャッシュマシンでは LP*4 バイトがキャッシュ(1MB 程度)に完全に収まるように LP の長さを決める。一方、使用する基底数(N)は 10 進 200 桁程度の分解で数千万になり、ストライド（基底）の大きいものは数億になる。そのため、大きい基底(素数)でのふるいはできるだけ大きい LP が利用したいという矛盾がキャッシュマシンでは発生する。また、ストライドは素数になっているため、ベクトルマシンではバンク競合が自動的に避けられる。

キャッシュマシンでは、ストライド（基底）が大きくなりすぎると、do 20 のループは実行されることが少なくなり、キャッシュの効果より空振りの影響が大きくなる。そのため、小

さな基底に対してはキャッシュ内で完全にふるいを行い、大きな基底では LP の数をまとめてキャッシュを無視してふるいを行う。そのように対策した、ふるい処理の心臓部は FORTRAN で記載すると下記のようになる。

(b) 大きい基底用に対策したキャッシュマシン用ふるい処理

```
      SUBROUTINE   SIEVE2(LP,NLP,JL,Base,LogP,N,N1,PS,V,Sive,NS)
      IMPLICIT   INTEGER*4(A-Z)
      INTEGER*8   Sive, LLP
      DIMENSION   Base(2,N), LogP(N), Sive(N), PS(LP*NLP), V(LP*NLP)
C   Clear V
      LLP = LP*NLP
      LLP = LLP*JL
      LS = 0
      do 100 l=1,NLP                          NLP 回繰り返す(キャッシュ内処理)
        do 10 i=1,LP
          V(i+LS) = 0                         ふるい領域の初期化処理
   10   continue
C   Sieve NO1                                 <小さい基底でのふるい処理>
      do k=1,N1                               N1 は小さい基底の数
        IP   = Base(2,k)
        IST = Base(1,k) + 1
        do 20 i=IST,LP,IP                      ふるい処理はキャッシュ内で行う
          V(i+LS) = V(i+LS) + LogP(k)
   20   continue
      end do
C   Reset Base                                <小さい基底の開始番地更新>
      do 30 k=1,N1
        Base(1,k) = mod(Base(1,k)-LP, Base(2,k))
   30   continue
      LS = LS + LP
  100 continue
C   Sieve NO2                                 <大きい基底でのふるい処理>
      do k=N1+1,N                             N1 番基底より大きい基底の処理
        IP   = Base(2,k)
        IST = Base(1,k) + 1
        do 25 i=IST,LP*NLP,IP                  キャッシュ外で処理(NLP 倍)
          V(i) = V(i) + LogP(k)
   25   continue
      end do
C   Reset Base                                <大きい基底の開始番地更新>
```

```
      do 35 k=N1+1,N
        Base(1,k) = mod(Base(1,k)-LP*NLP, Base(2,k))
   35 continue
C    Select                                          ＜ふるいデータの採取＞
      do 40 i=1,LP*NLP
        if(V(i) .ge. PS(i)) then
          NS = NS + 1
          Sive(NS) = i + LLP
        end if
   40 continue
      RETURN
      END
```

ここで、基本となるSIEVE1ルーチンに追加した引数は、NLPとN1の二つである。N 個の基底を、小さい方から N1 個までの基底と、N1+1～N 個までの基底に分けてふるいを行う。LP は SIEVE1 と同様にキャッシュ内でふるいが行える1回のふるい長である。NLP は N1+1～N 個までの大きい基底のふるいを LP*NLP の長さに拡大して行うものである。これは、大きい基底では do ループの空振りによる効率の低下が、キャッシュミスによる効率の低下以上になるのを避けるためである。そのため、N1 は効率が逆転する大きさを事前に予測して与える。この予測値は、簡単なテスト実行で決めることができる。N1 個までの小さな基底に対するふるいは、Sieve1 と同じくLPのふるい長で行い、このルーチン内で、NLP 回繰り返す。Base(1,k)なる k 番基底のふるい開始番地更新は、1回のふるい処理に同期して行う必要があるため、小さい基底においては NLP 回行い、大きい基底に対しては1回で処理する。ふるいデータの採取は、N 番までのすべての基底におけるふるい処理が完了して行う必要があるため、最後に行う。

5. ベクトル計算機でのふるい処理

ベクトル計算機として今回は地球シミュレータ(ES2)での高速化を目標とした。ES2 はベクトルキャッシュが使える仕様になっているが、ふるい処理にはサイズが小さく、ベクトル長が長く保てないため、利用しない方針とした。このため、ES2 ではキャッシュサイズに依存しないで、1 回のふるい長(LP)が長く取れるので都合がよく、ベクトル化も十分である。しかし、ふるい本体(do 20 を含む do ループ)に比べ、演算量は非常に少ない「ふるいデータ採取」(do 30 のループ)が遅く、期待した性能が得られないことが判明した。この部分もベクトル化はするが、圧縮処理という本来の要素単位のベクトル処理とは異なり、性能が悪い。しかし、「ふるいデータの採取」で選ばれるデータは

もともと稀で、分解桁数が大きくなると数百万～数億に 1 回と極端に少なくなる。
また、分解桁数が小さい場合は、パソコンで数秒もかからず分解可能なので、今回の高速化の対象ではない。この性質を利用し、数万単位で採用データの有無を判定し、その区間に採用データが存在する場合だけ当該ループでふるい処理を行うことで高速化できた。この対策により、心臓部のふるい処理全体で約 3 倍の高速化ができた。

ふるい処理は、各プロセッサに担当させるふるい担当区間を決めれば、通信オーバーヘッドはほとんど発生せず、負荷も均等に分けられるため、並列化には非常に向いた計算である。そのため、多数の台数の並列化でもほぼ 100%の並列化率で並列計算可能である。ベクトル用に対策した、ふるい処理の心臓部は FORTRAN で記載すると下記のようになる。本ふるい計算は MPI を使用し並列計算できるようにしたものである。

(a) ベクトル計算機用ふるい処理

```
      SUBROUTINE   SIEVE3(LP,JL,Base,LogP,N,PS,V,Sive,NS)
      IMPLICIT   INTEGER*4(A-Z)
      INTEGER*8   Sive, LLP, LNP, LW
      PARAMETER (lb=1024*64)          lbはふるいデータ採取判定のブロック長
      COMMON /MPP/np,nr               npは利用プロセッサ数、nrは自分の番号
      DIMENSION Base(2,N), LogP(N), Sive(N), PS(LP), V(LP)
C   Clear V
      LW   = LP
      LNP = LW*np
      LLP = LNP*JL + LW*nr                呼ばれた回数と各プロセッサで異なる
      do 10 i=1,LP
         V(i) = 0                         ふるい領域の初期化処理
   10 continue
C   Sieve                                 ＜Sieve1と同じふるい処理＞
      do k=1,N
         IP   = Base(2,k)
         IST = Base(1,k) + 1
         do 20 i=IST,LP,IP
            V(i) = V(i) + LogP(k)
   20 continue
      end do
C   Select                                  ＜ふるいデータの採取＞
      do j=1,LP,lb                          ＜lbの長さ単位で採否の判定＞
        wk = -100
        do i = j,min(j+lb-1,LP)
          wk = max(wk, V(i)-PS(i))          wkは差の最大値
```

```
        end do
        if (wk .ge. 0) then                    このブロックに採取データあり
          do 40 i = j,min(j+lb-1,LP)           ブロック内で採取データ特定
            if (V(i) .ge. PS(i)) then
              NS = NS + 1
              Sive(NS) = i + LLP
            end if
 40       continue
        end if
      end do
C   Reset Base                                 ＜次のふるいの開始番地更新＞
      do 30 k=1,N
        Base(1,k) = mod(Base(1,k)-LNP, Base(2,k))
 30   continue
      RETURN
      END
```

ここで、ベクトル用のふるい処理Sieve3の引数と、キャッシュマシン用の基本ふるい処理Sieve1の引数は同じである。ただし、LPの大きさは桁違いに大きくなる。本ルーチンはMPIによる並列処理ができるように、COMMON変数でnpとnrを定義している。npは利用するプロセッサ数で、nrは自プログラムが動作するプロセッサ番号である。このため、k番基底における開始番地Base(1,k)はnrに依存して異なる値が与えられる。ふるいデータの採取処理が、ベクトル用に対策してあり、Sieve1と大幅に異なる。

LPが十分大きいので、本ふるい処理Sieve3で十分大きな基底のふるいに対応できそうに思われるが、超大基底に対してはdo 20のベクトル長が短くなり性能が低下する。そのため、超大基底に対しては、ブロック化した基底数をループの最内側にし、リストベクトルを使用してベクトル長を長くする工夫が必要となる。ベクトル計算機を使用したふるい処理では、キャッシュマシンに比較してより多くの基底を使用した方が高速化できるため、この工夫は重要となる。超大基底用に対策した、ふるい処理の心臓部はFORTRANで記載すると下記のようになる。

(b) 超大基底用に対策したベクトル用ふるい処理

```
      SUBROUTINE   SIEVE(LP,JL,BaseL,LogP,N,N2,PS,V,Sive,NS,NW)
      INTEGER*8   Sive, LLP, LNP, LW
      PARAMETER (lb=1024*64)
      COMMON /MPP/np,nr
      DIMENSION   Base(2,N), LogP(N), Sive(N), V(LP), NW(N)
C   Clear V
      LW   = LP
```

```
      LNP = LW*np
      LLP = LNP*JL + LW*nr
      do 10 i=1,LP
        V(i) = 0
   10 continue
C    Sieve NO1 (stride sieve)                    ＜ふるい処理(超大基底以外)＞
      do k=1,N2
        IP  = Base(2,k)
        IST = Base(1,k) + 1
        do 20 i=IST,LP,IP
          V(i) = V(i) + LogP(k)
   20   continue
      end do
C    Sieve NO2 (list sieve)                      ＜超大基底のふるい処理＞
      do j=N2+1,N
        NW(j) = Base(1,j)                       開始番地をワークにコピー
      end do
      do 60 k=N2+1,N,lb                         lbブロック単位にふるい
        je = min(k+lb-1,N)
        lg = LP/Base(2,k) + 1
        do i=1,lg
!CDIR NODEP                                     強制ベクトル化(ES2)
          do 50 j=k,je
            IST = NW(j)
            if(IST .lt. LP) then
              V(IST+1) = V(IST+1) + LogP(j)     リスト処理でふるい
              NW(j) = IST + Base(2,j)           ワークの開始番地更新
            end if
   50     continue
        end do
   60 continue
C    Select                                      ＜ふるいデータの採取＞
      do j=1,LP,lb                              ＜lbの長さ単位で採否の判定＞
        wk = -100
        do i = j,min(j+lb-1,LP)
          wk = max(wk, V(i)-PS(i))              wkは差の最大値
        end do
        if (wk .ge. 0) then                     このブロックに採取データあり
          do 40 i = j,min(j+lb-1,LP)            ブロック内で採取データ特定
            if (V(i) .ge. PS(i)) then
              NS = NS + 1
```

```
                Sive(NS) = i + LLP
              end if
   40       continue
          end if
        end do
C   Reset Base                                    ＜次のふるいの開始番地更新＞
        do 30 k=1,N
          Base(1,k) = mod(Base(1,k)-LNP, Base(2,k))
   30 continue
        RETURN
        END
```

ここで、ベクトル用ふるいSieve3ルーチンに追加する引数は、N2とNWの二つである。N2はN個の基底の中で、N2+1番以降の基底を超大として、doループの順序を入れ替えて、リスト処理するものである。N2番以下の基底のふるい処理は変更しない。配列NWはワーク配列で、リスト処理する間、N2+1番以降の基底に対する開始番地を更新しながら、保持するものである。配列Baseの方の開始番地の更新は基底の番号に関係なく一括して行う。また、ふるいデータの採取もベクトル用ふるいルーチンSieve3と同じである。do 50のループの前には、リストベクトルを使用してベクトル化するため、ベクトル計算機（地球シミュレータ、ES2）で強制的にベクトル化する指示文を追加する。

6. ふるい処理計算量の比較

現在最も長い桁数の RSA 暗号の因数分解は、2010 年 1 月に日本の NTT 他 5 カ国が共同して行い発表 [5]した、RSA-768(10 進 232 桁である)。この因数分解は 1 次式と 6 次式の二つの多項式を使用して、GNFS(一般数体ふるい法)で実施された。一般に GNFS で合成数 n を因数分解するには、下記の 6 ステップが必要である。

(a) 合成数 n から多項式の係数を求める（利用関数の探査）
(b) ふるいに利用する基底（素数及び素イデアル）の選定
(c) ふるい処理で基底数以上のふるいデータの採取（ふるい処理）
(d) ふるいデータより 0-1 行列の作成
(e) 0-1 行列から従属行の組を求める(0-1 行列計算)
(f) 多項式を法とする代数平方根の計算（代数平方根の計算）
(g) $a^2-b^2=0 \pmod n$の形に変形し、合成数 n を因数分解

MPQS(複数多項式 2 次ふるい法)の場合は上記、(a),(f)が不要で、基底は平方剰余となる素数から選ぶ。GNFS の素イデアルもふるいプログラムの観点でみると素数と

同じになる。全体の計算の中で、(b),(d),(g)の計算量は微量であり、計算量で比較する場合はその他扱いする。NTT 他[5]で実施した、RSA-768 の計算量を表 1 に示す。

表 1. RSA-768(10 進 232 桁)の計算量

処理項目	計算量(台数・年)	構成比率(%)
利用関数の探査	20	1
ふるい処理	1500	90
0-1 行列計算	155	9
代数平方根の計算	1	0
その他	1	0

注)計算量は ADM64(2.2Gh)の 1 コアで 1 年の計算が 1 台数・年

RSA-768 の分解に使用された、0-1 行列サイズは、192,796,550 ×192,795,550 である。約 2 億次元の行列で、列サイズが行サイズより 1000 だけ大きい少し縦長の行列である。0-1 行列の従属行を求めるのは、ベキ数が偶数にして平方数にするためである。表 1.より大きい合成数 n の因数分解の計算量の大半はふるい処理が占めることが分かる。次に 0-1 行列計算の計算量が多く、それ以外の計算量は少ない。

ふるいの原理では、ふるいで得られたデータ数が列サイズ(行数)になり、基底の数の合計が行サイズ(列数)となる。しかし、ふるい処理でより多数のふるいデータを得るため、基底だけでは分解されないデータも基準を設けて集め、基底外の素数(素イデアル)を 1 つ加えると、2 件以上のデータが得られるデータも利用している。この処理はふるい心臓部に比較し、プログラム処理は非常に複雑であるが、計算量は少ない。そのため、本来はふるい処理の計算量は、この追加基底処理まで考えて比較すべきである。しかし、この追加基底処理まで考えた評価をすると、ふるい本体の高速化評価の基準がぼやける。また、追加基底処理を含めて高速化評価すると、この処理方法と追加基準の設定により高速化が大きく左右されてしまう。今回の目的は、RSA 暗号のふるい処理で、ベクトル計算機が PC に比較してどれだけ高速にできるかを見定めることである。そのため、ふるい処理は最初に選定した基底だけで集めるふるい処理に限定して比較する。更に、本目的をより純粋に比較できるように、種々の方法が提案されている GNFS や MPQS のふるい処理ではなく、これらと同じ処理で、余分なパラメータが入らないふるい処理を使用して評価する。それは、10 進 m 桁の値を与え、その値から先の数値を、N 個の素数基底でふるい処理することにした。ここで、N 個の素数基底は小さい方から順に選ぶ。この方法は、MPQS のふるいに対応させると、約 10 進 2m 桁のふるいに対応する。また、GNFS のふるいに対応させると、分解桁数が多い場合はおおよそ 10 進 3m 桁のふるいに対応する。MPQS は素数の約半分が基底になり、GNFS は 2 種類のふるいがあり、一種類は全ての素数が基底になり、もう一方

(素イデアル基底)は素数の数分の一が基底に対応する。また、ふるいは、基底の数の選び方に大きく左右される。その左右のされ方は、PC とベクトル計算機では異なるものと推定される。そのため、ふるい処理の高速化の評価は、10 進 m 桁の値を与え、その先からの数値を、N 個の素数基底で行うふるいを対象とする。その場合、同じ m 桁の値に対し、N をパラメータとして変化させる。

7. ふるい処理時間の比較

ベクトル計算機がふるい処理で、PC に比べどのくらい高速化できるかを比較する。比較には表 2.に示す、PC とベクトル計算機を使用した。

表 2. 使用した PC とベクトル計算機

項目	PC	ベクトル計算機
使用ハード	Dell Vostro 200 Intel Core 2 2.3Ghz, 2GB メモリ	地球シミュレータ (ES2) 3.2Ghz 1ノード: 819Gflops,128GB
測定単位	1 コア cpu 時間で測定	1 ノード (8cpu) 経過時間で測定
使用ソフト	Windows Vist, g77 -O3 オプション	NEC SUPER-UX 自動ベクトル化 FORTRAN + MPI

計算速度の比較には、10 進 45 桁と 10 進 60 桁からの値のふるい処理を用いた。これは、MPQS(複数多項式 2 次ふるい法)では、90 桁と 120 桁程度のふるい処理の比較に相当する。GNFS では、ほぼ 130 桁と 170 桁程度のふるい処理の比較に相当すると考えられる。現在の RSA 暗号は 1024 ビット(10 進 309 桁)の合成数を使用しており、分解されている世界記録は RSA-768(768 ビット、10 進 232 桁)である。そのため、本来は更に大きい桁数で比較すべきであるが、計算時間を考慮して選定した。ふるいの計算速度の比較では、ふるいに使用する基底の数も大きく影響する。今回の比較に使用する基底は全て素数であり、小さい方から N 個の素数を選び基底とした。

測定に使用したプログラムは、Sieve2 を PC 用に、Sieve4 をベクトル計算機用に使用した。また、1 回のふるいの長さ LP は、PC ではキャッシュに入るように LP=512K とし、ベクトル計算機では LP=1G とした。ここで、K,M,G は K=1024、$M=1024^2$、$G=1024^3$ としている。Sieve2 も Sieve4 も基底数 N を二つに分けて、異なるふるいを実施している。そのため、分けるための基準値が必要である。この基準値は事前に測定で求め、PC 用の Sieve2 の N1 は 100K とし、ベクトル用の Sieve4 の N2 は 1M と

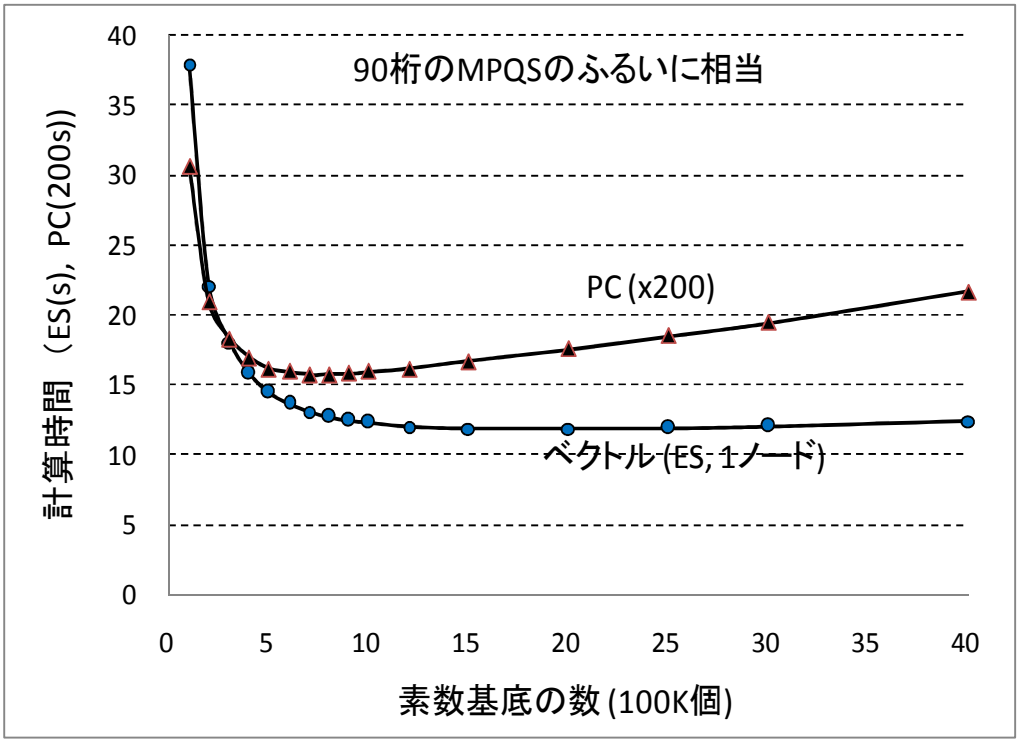

図 1.　ふるい処理時間の比較（10 進 45 桁）

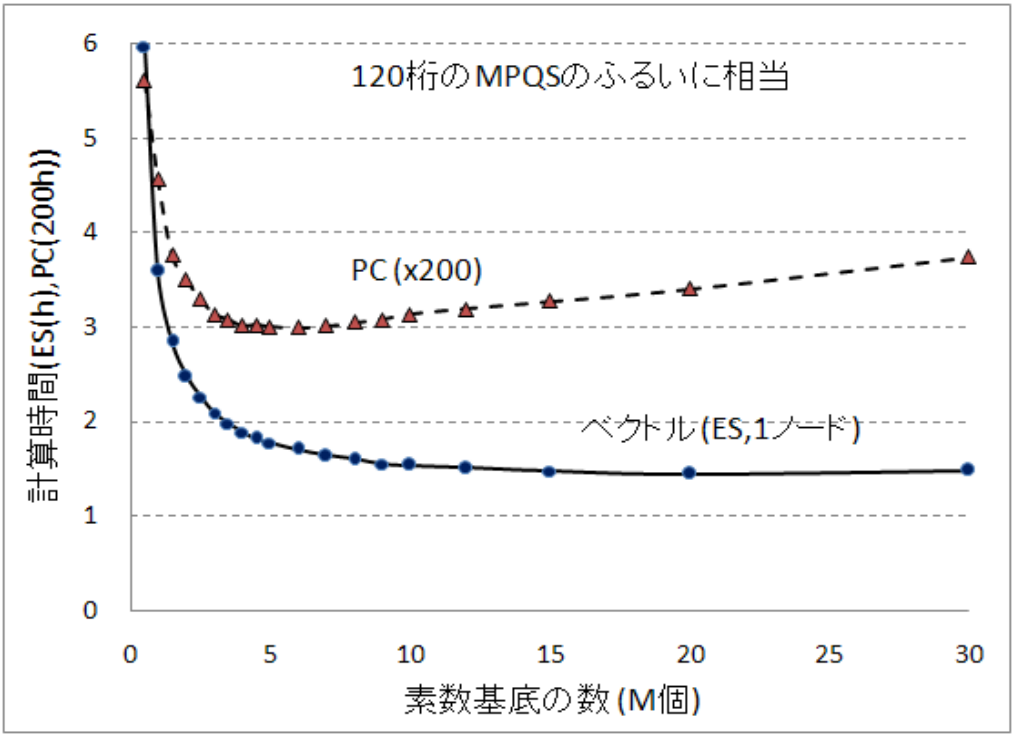

図 2. ふるい処理時間の比較(10 進 60 桁)

した。図 1 に 10 進 45 桁からの値のふるい処理時間の比較を示す。図 2 に 10 進 60 桁からの値の処理時間の比較を示す。どちらも、PC の処理時間は縦軸の 200 倍の時間で表示している。

図 2 の PC の処理時間を点線で示しているのは、1 ケースの測定で 600 時間を超える測定が必要なため、1/100 のふるいデータが得られるまでの時間を測定し、処理時間はそれを 100 倍したためである。その妥当性(ふるいはデータはほぼ均等に得られる)はベクトル計算機で検証した。全体測定した場合と 1/100 測定での推定は数%の差であり、グラフに表示すると線の太さにかくれる。図 1,図 2 ともに、素数基底の数により処理時間が変化することが分かる。図 1 より PC では 700K 個程度の基底数でふるい処理時間は最短の 3200 秒となる。一方、ベクトル計算機では 2M 個程度の基底数で最短の 12 秒となる。その比率は約 270 倍である。図 2 より PC では 6M 個程度の基底数でふるい処理時間は最短の 600 時間となる。また、ベクトル計算機では 20M 程度で最短の 1.45 時間となる。その比率は約 410 倍である。これから、分解対象の桁数が長くなると、ベクトル計算機の PC に対する高速化の比率は大きくなることが分かる。また、同じ桁数で最短になる基底の数は、PC よりベクトル計算機の方が多くなることも分かる。ふるい処理において、もし基底(素数)の大きさにかかわらずワークへの加算処理(V(i) = V(i) + LogP(k))1 回の処理時間が同じなら、基底の数を大きくする方がふるい処理時間全体は小さくなる。しかし、この加算処理1回のコストが基底の大きさにより変わるため、特定の基底の数で最短の処理時間となる。基底の大きさによるこのコストの増加が、PC よりベクトル計算機の方が緩やかなため、時間最短の基底数が大きくなる。また、図 1,図 2 より最短に近い基底数の範囲は PC よりベクトル計算機の方がはるかに広いことが分かる。この範囲が広いと、使用する基底の数の推定が多少ずれても処理時間への影響が少なく、安全に推定できる。

実際に知りたいのは更に大きな桁数の場合である。基底の数は分解する対象数の桁数に応じて大きくする必要がある。そこで、素数基底の数を横軸にして、PC とベクトル計算機の性能比がどのようになるかを検討した。図 3 は素数基底の数によるベクトル計算機と PC の性能比較である。図 1 及び図 2 のデータを基に作成し、MPQS(複数多項式 2 次ふるい法)の 150 桁相当の部分は、そのまま延長して推定したものである。そのため、150 桁相当の部分は点線で示してある。同じ桁数でも最速となる素数基底の数は、PC とベクトル計算機で異なり、ベクトル計算機の方がより大きくなる。RSA 暗号解読のために、MPQS で 150 桁に相当するふるい処理より更に、1 段上の規模のもので、GNFS(数体ふるい法)によるふるい処理である。このときの基底は素数基底だけでなく素イデアル基底も必要になる。しかし、素イデアル基底によるふるい処理は、処理プログラムの観点からは素数基底と同等である。これらから、分解対象桁数が長くなるに従い、ベクトル計算機の PC に対する性能比は増大し、150 桁の MPQS では 600 倍程度で、実際に適用する規模のふるい処理では 800 倍程度と推

定される。

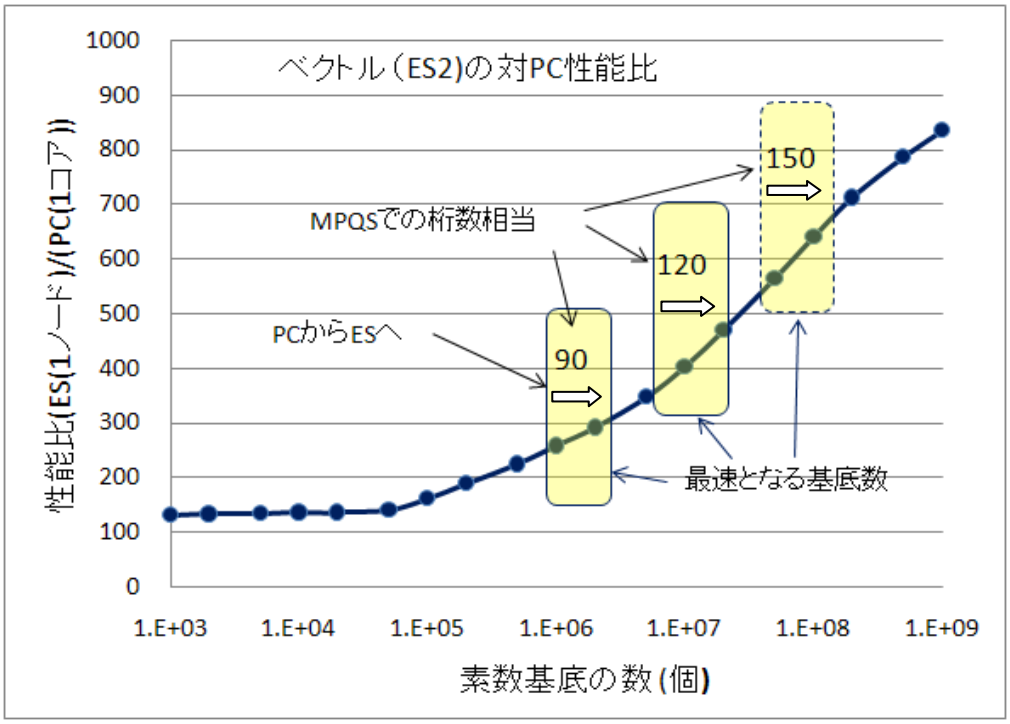

図 3. 素数基底の数によるベクトル計算機と PC の性能比較

8. おわりに

RSA 暗号のふるい処理は、多数桁の因数分解を MPQS や GNFS で行う場合に発生する。ふるい処理では、基底(MPQS では素数、GNFS では素数と素イデアル)の数以上のデータをふるい処理で求める。実際のふるい処理では、基底だけで得られるデータだけでなく、一定の基準で基底をオーバーしたデータも採取し、基底外でも 2 件以上のデータが揃えば、それを新たに基底と採取データに追加して高速化している。しかし、追加する基底とふるいデータまで考えると、高速化評価の基準がぼやけるため今回の評価は基底の数だけのふるいデータを得るまでの処理時間で評価した。その結果下記のことが判明した。

(a) PC(2.33Hz,1 コア)とベクトル計算機(ES2,1 ノード)のふるい処理の性能比は 200 倍～800 倍となる。

(b) 分解対象桁数が大きくなると、性能比も大きくなり、実用的な規模では性能比は 800 倍程度となる。

(c) 同じ桁数の分解では、最速となる基底の数は PC よりベクトル計算機の方が大きくなる。また、最速に近い基底の数の範囲が PC より拡大し、ふるい処理で使用する基底の数の推定がより容易になる。

今後は、本結果をふまえて、GNFS に適用し 10 進 100 桁から 200 桁近くまで、一定の桁数単位で計算し、GNFSの桁数に対する計算時間を推定するつもりである。その結果で暗号の「2010 年問題」と言われる、1024 ビット RSA 暗号の解読にかかる時間を推定したいと考えている。

＜謝辞＞

地球シミュレーター(ES2)の高速化について、貴重なご意見を頂いた海洋研究開発機構・地球シミュレータセンターの福井義成氏及び浅野俊幸氏に謹んで感謝の意を表する。

＜参考文献＞

1) 木田 祐司：数体ふるい法による素因数分解、2003 年、
http://www.rkmath.rikkyo.ac.jp/~kida/nfs_intro.pdf
2) 山崎 潤一、釜池 聡太：暗号の 2010 年問題、ASCⅡ, Technologies, 2010 年 9 月、 特集 2、pp75～93
3) Richard A. Mollin: RSA and PUBLIC-KEY CRYPTOGRAPHY, CRC Press, 2003 年
4) N. コブリッツ、桜井 幸一訳：数論アルゴリズムと楕円暗号理論入門、Springer, 1998 年
5) NTT 情報流通基盤総合研究所：公開鍵暗号の安全性の根拠である「素因数分解問題」で世界記録を更新, NTT News Release 100108a, 2010 年 1 月,
http://www.ntt.co.jp/news2010/1001/100108a.html
6) 今井 秀樹他：暗号技術検討会 2009 年度報告書、2010 年 3 月、
http://www.cryptrec.go.jp/report/c09_kentou_final.pdf
7) 地球シミュレータ講習会テキスト一式、海洋研究開発機構地球シミュレータセンター & NEC、2010 年
8) Dr. Ari Juels 他： The RSA Challenge Numbers, RSA Laboratories, 2010,
http://www.rsa.com/rsalabs/node.asp?id=2093