GPGPUによる RSA暗号ふるいの性能評価

2011年6月20日

後 保範 (早稲田大学)

目　次

1. はじめに

(1) 現在の暗号 (RSA暗号)や認証システムは多数桁数(1024ビット,10進309桁)の因数分解の困難さを利用している。

(2) 現在の多数桁数の因数分解の世界記録は、RSA-768(10進232桁)。2010年1月、NTTを含め5カ国共同、GNFSを使用。

(3) 「暗号の2010年問題」:RSA暗号を含む現在の暗号システムの変更が必要

1.1　暗号化方式

(1) 公開鍵暗号方式(非対称鍵)

公開鍵で暗号化、秘密鍵で復号化

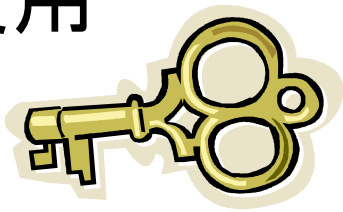
認証やネットワーク通信に都合が良い

RSA暗号：多数桁数因数分解の難しさを利用

(2) 秘密鍵暗号方式(共通鍵、対称鍵)

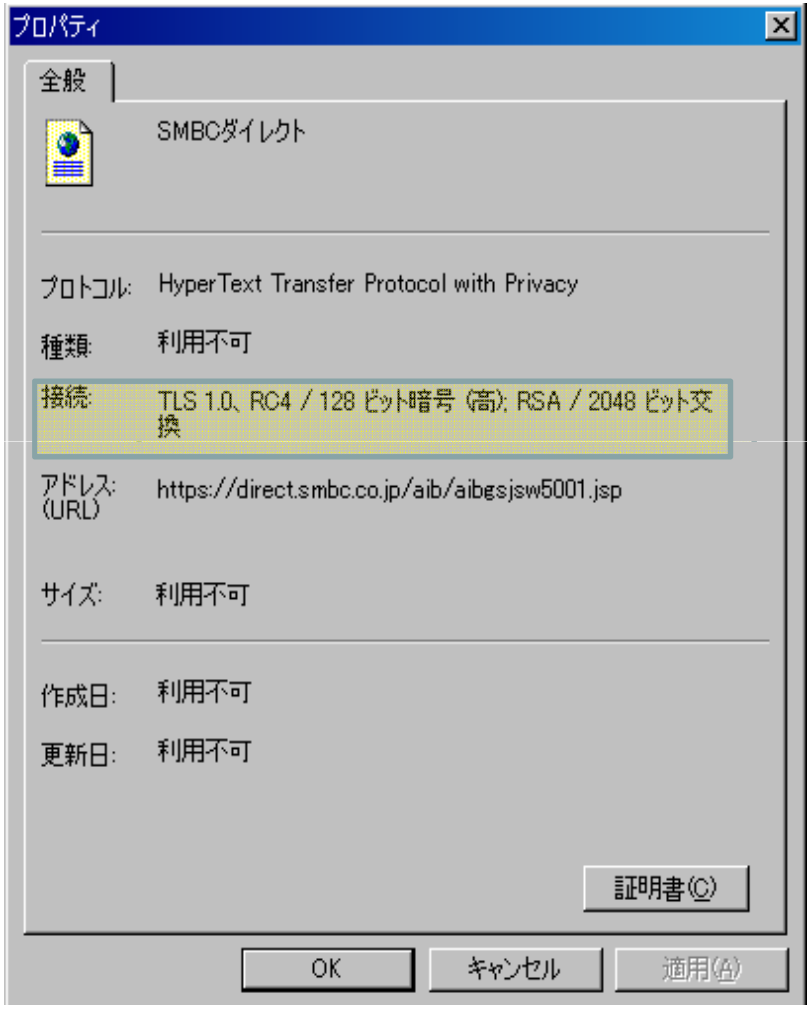
暗号化と復号化で共通の秘密鍵を使用

代表暗号例：AES, RC4(Netscape ）



1.2 インターネットの暗号例

暗号化なし(http)



暗号化あり(https)

1.3 RSA暗号の仕組み

- RSA暗号鍵の作成(数学的な方法)
  (1) ランダムに素数p, qを選ぶ
  (2) $n = p \times q$及び$f = (p-1) \times (q-1)$を計算
  (3) 素数eを選ぶ
  (4) $d = 1/e \pmod f$となるdを計算する

- (e,n)が公開暗号化鍵、(d,n)が復号鍵となる

1.4 RSA暗号化と復号化

- RSA暗号化

  (1) 情報をn以下の数Mに変換（公開方法)

  (2) $C=M^e \pmod n$で暗号Cを作成

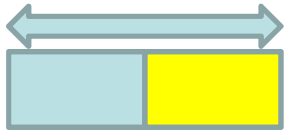
  $(n=pxq, f=(p-1)x(q-1))$

- RSA復号化　　オイラーの定理($M^f \equiv 1 \pmod n$)

  (1) $M=C^d \pmod n$で元の数Mに復号

  (2) 数Mを元の情報に変換（公開方法）

1.5　因数分解の方法



(1)　ふるい(Sieve)系解法

計算量は合成数の桁数に依存

RSA暗号の解読に都合が良い

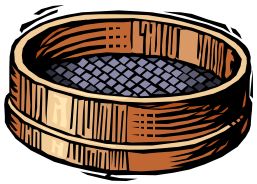
MPQS、GNFSが代表的解法

(2)　楕円曲線法(Elliptic Curve Method, ECM)

計算量は小さい因数の桁数に依存

RSA暗号の解読には不向き

2. ふるい法



(1) $A^2-B^2=(A-B)(A+B)\equiv 0 \pmod N$の関係を使用し、Nを因数分解

(2) $A_1^{l1}\cdot A_2^{l2}\cdots A_k^{lk}\equiv B_1^{m1}\cdot B_2^{m2}\cdots B_j^{mj} \pmod N$ なる関係を基底の数より多く集める

(3) 0-1行列を計算し、両辺が平方になるもの(従属関係)のデータを探す

(4) MPQS,GNFS等が代表的なふるい法

2.1　代表的なふるい法

(1)　QS (Quadratic Sieve、2次ふるい法)
MPQS (Multiple Polynomial QS,
複数多項式2次ふるい法)が代表的解法
100桁以下ではGNFSより高速な解法

(2)　GNFS (General Number Field Sieve、
一般数体ふるい法)。現在、100桁程度以上で最も高速な解法と言われている。

2.2 QS(2次ふるい法)

(1) QS (Quadratic Sieve, 2次ふるい法)

Nを分解、XはN$^{1/2}$に最も近い整数

$(X+k)^2 - N = A_k$, k=0,1,2, …

素数基底で分解できるA_kを集める

(2) MPQS (Multiple Polynomial QS)

複数の2次多項式を使用

代表例: $d^2 - N \equiv 0 \pmod{c}$なる(c,d)の組で

$(c \cdot x+d)^2 - N = c \cdot f(x)$ と変換しf(x)を分解

2.3 NFS(数体ふるい法)

(1) 分解の違いを利用 (SNFS,特殊数体ふるい法)

Nを分解、f(M)≡0 (mod N)なる多項式

f(x)=0の根の一つをθとする。

a+bMを素数基底で分解、

a+bθを生成元(素元と単元)で分解

(2) N=1333の例 (f(x)=x^3+2, M=11, θ^3=−2)

2+M=13, 2+θ = θ (1− θ)(1+ θ)= θ − θ^3

→ −10·11·12≡13 (mod N)

2.4 GNFS(一般数体ふるい法)

(1) SNFS→GNFS　　$f(x) \equiv (x-s)g(x) \pmod{p}$

一般には素元が求まらない。

→ 素元の代わりに素イデアルを使用

問題点：平方の形が明示的に現れない

→ イデアルの積が平方となるようにする

(2) GNFSで新たに必要なこと

(a) 平方剰余の追加：平方の確率を高める

(b) イデアルの平方根(代数平方根)を求める

3. ふるい処理(多数桁因数分解の)

- ふるい用4～7次関数の探査(GNFSだけ)
- 分解基底(素数、素イデアル)の選定
- ふるい処理で基底数以上のデータ収集

  基底ベキを要素とする行列(基底数xデータ数)作成

  基底ベキを(mod 2)して0-1行列に変更
- 0-1行列から従属となる行(データ)を計算
- 代数平方根の計算(GNFSだけ)
- $A^2 - B^2 \equiv 0 \pmod N$を構成し、因数分解

3.1 RSA-768(232桁)の計算時間

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

注) AMD64 (2.2Ghz, 1コア換算)

行列サイズ : 192,796,550 * 192,795,550

3.2 ふるいプログラム(中心部)

```
for (k=0; k<N; k++) : 基底の素数の数
  { for (i=Start[k]; i<LP; i+=Prime[k])
     { V[i] += LogP[k]; }   : 元は乗算(対数化で加算)
  }                                  LPは一回のふるいサイズ
for (i=0; i<LP; i++) : ふるいデータの採取
  { if(V[i] >= PS[i]) { Sive[No] = Pointer + i;
                        No++; }
  }                                  Noはふるいで得られたデータの数
次のふるいのためStart[0]～Start[N-1]を更新
```

4. ふるい処理の特徴

- PC(キャッシュ処理のパソコン)
  高速化のためにはキャッシュ内処理が必須
  LPのサイズはGPUより約千倍短い
  少し大きな素数(基底)ではLPをはみ出す
- GPU(GTX480)
  LPのサイズはPCの約千倍長く取れる
  本質的に非連続アクセスで、連続化は不可

注) LP: 1回のふるい処理での配列長

4.1 GPUプログラム(対策前)

```
no = gn*bn;                                  bn=512, gn=40を使用
for (k=0; k<LP; k+= no)                      LP=250*1024²を使用
   { i = (bn * blockIdx.x + threadIdx.x) + k;
     V[i] = 0; }                             Vの初期化
__syncthreads();
for (k=0; k<N; k++)                          区間LPで20480並列
   { for (i=Start[k]; i<LP; i+=Prime[k]*no)
       { ii = (bn*blockIdx.x+threadIdx.x)*Prime[k]+i;
         if(ii < LP) V[ii] += LogP[k]; }
     __syncthreads(); }
```

4.2 GPU並列化問題点

- LPのサイズはPCに比較して約1000倍大きくできるが、20480のスレッドで並列化すると、1スレッド当たりではPCよりLPは小さくなる
- このためk番目の素数の値Prime[k]が大きくなると、iiはPrime[k]のスレッド倍より大きくなり、下記のif文は空振りが多くなる

```
if(ii < LP) V[ii] += LogP[k];
```

4.3 GPU並列化対策

- スレッド並列化を区間LPから上位のN個の素数に変更する

  → データの依存性が発生し、V[ii]+=LogP[k];

  の加算処理が正しく動作しない

- ふるい処理 (1/1000以下のふるいデータの取得漏れはOK)なら許される並列化

  → N個の素数を3区分(小、中、大-特大)に

  分けて、異なるスレッド並列化を行う

4.4 GPUプログラム(対策後)

```
for (k=0; k<N1; k++)              1～5120(5K)番素数のふるい
   { for (i=Start[k]; i<LP; i+=Prime[k]*gn*bn)
      { ii = (bn*blockIdx.x + threadIdx.x)*Prime[k] + i;
        if(ii < LP) V[ii] += LogP[k];   __syncthreads(); }   }
for (k=N1; k<N2; k+=gn)           5K+1～40K番素数のふるい
   { kk = blockIdx.x + k;
     for (i=Start[kk]; i<LP; i+=Prime[kk]*bn)
      { ii = threadIdx.x*Prime[k] + i;
        if(ii < LP) V[ii] += LogP[kk];   __syncthreads(); }   }
for (k=N2; k<N; k+=gn*bn)         40K+1番素数以降のふるい
   { kk =(bn*blockIdx.x + threadIdx.x) + k;
     for (i=Start[kk]; i<LP; i+=Prime[kk])
      { if(i < LP) V[i] += LogP[kk];   __syncthreads(); }   }
```

5. ふるい処理高速化の評価法

- 10進m桁からの値を、小さい方からN個の素数基底でのふるい処理で評価

  MPQS：平方剰余となる素数を基底に使用(半分)

  GNFS：イデアル分解(処理の基本は素数基底と同じ)と10進s桁程度の素数基底分解

- ふるいでN個のデータが得られるまでの時間を測定

  通常のふるい処理では、同じ素数(イデアル)のデータが2件得られたら、基底に追加し、処理を短縮。評価を単純化するため、基底の追加はしない。

5.1 並列化対策によるふるい採取データ数の変動

- 10進60桁、LP=250*1024^2、利用素数の数N=10000*1024、反復数=5000で測定
- ふるいで採取されたデータ数(M)

  (1) 未対策:　　M=448425

  (2) 並列化対策後(12回測定)

  M=[448338, 448367], 平均M=448350

  未対策との最大差M= 87 = 0.02%

6. ふるい測定結果の比較

- PC(1コア)とGPU(1台)で比較
- PC

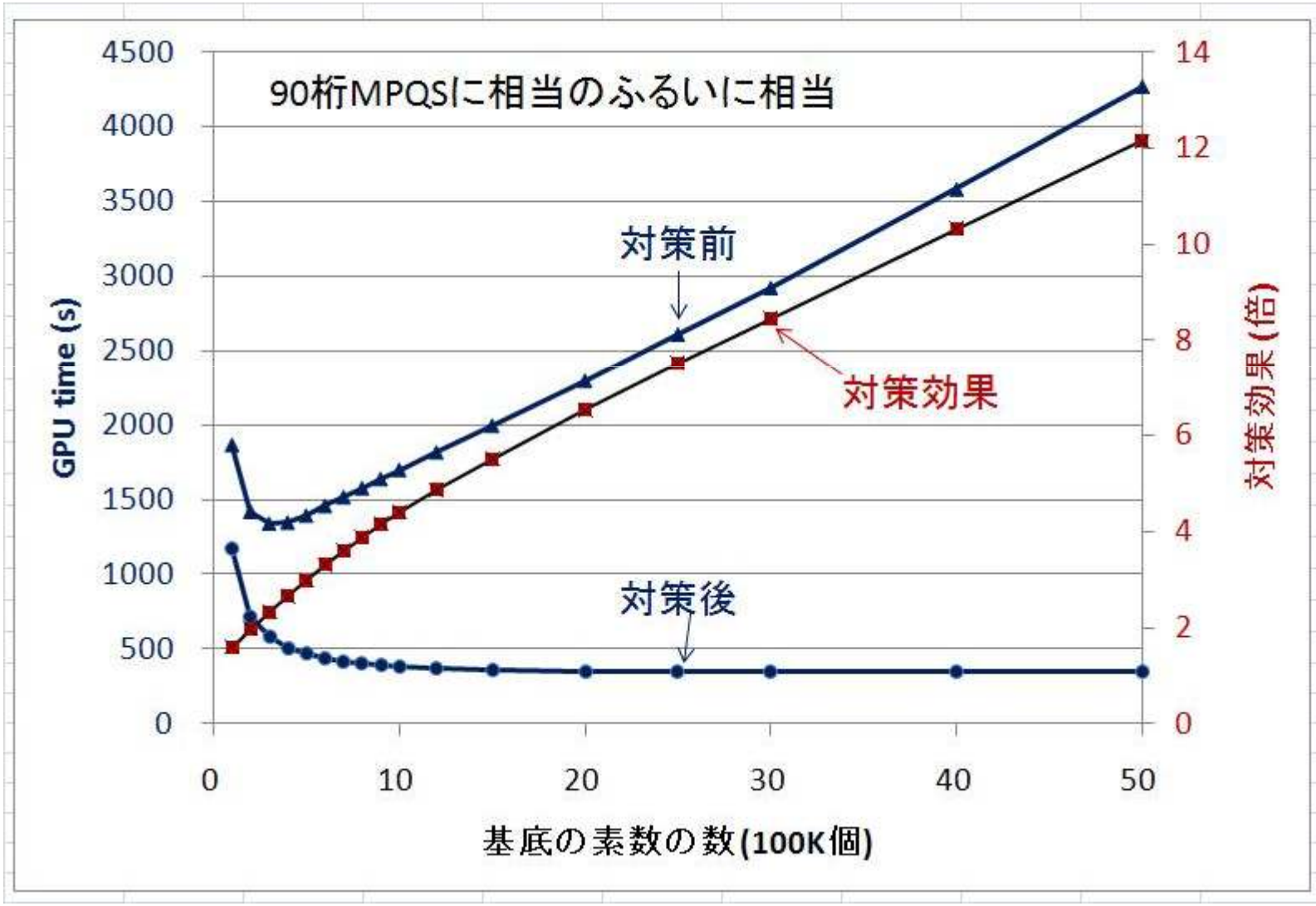
  Dell Vostro 200 (Intel Core 2, 2.33Ghz, 2GB)

  Windows Vista, gcc, -O3オプション
- GPU

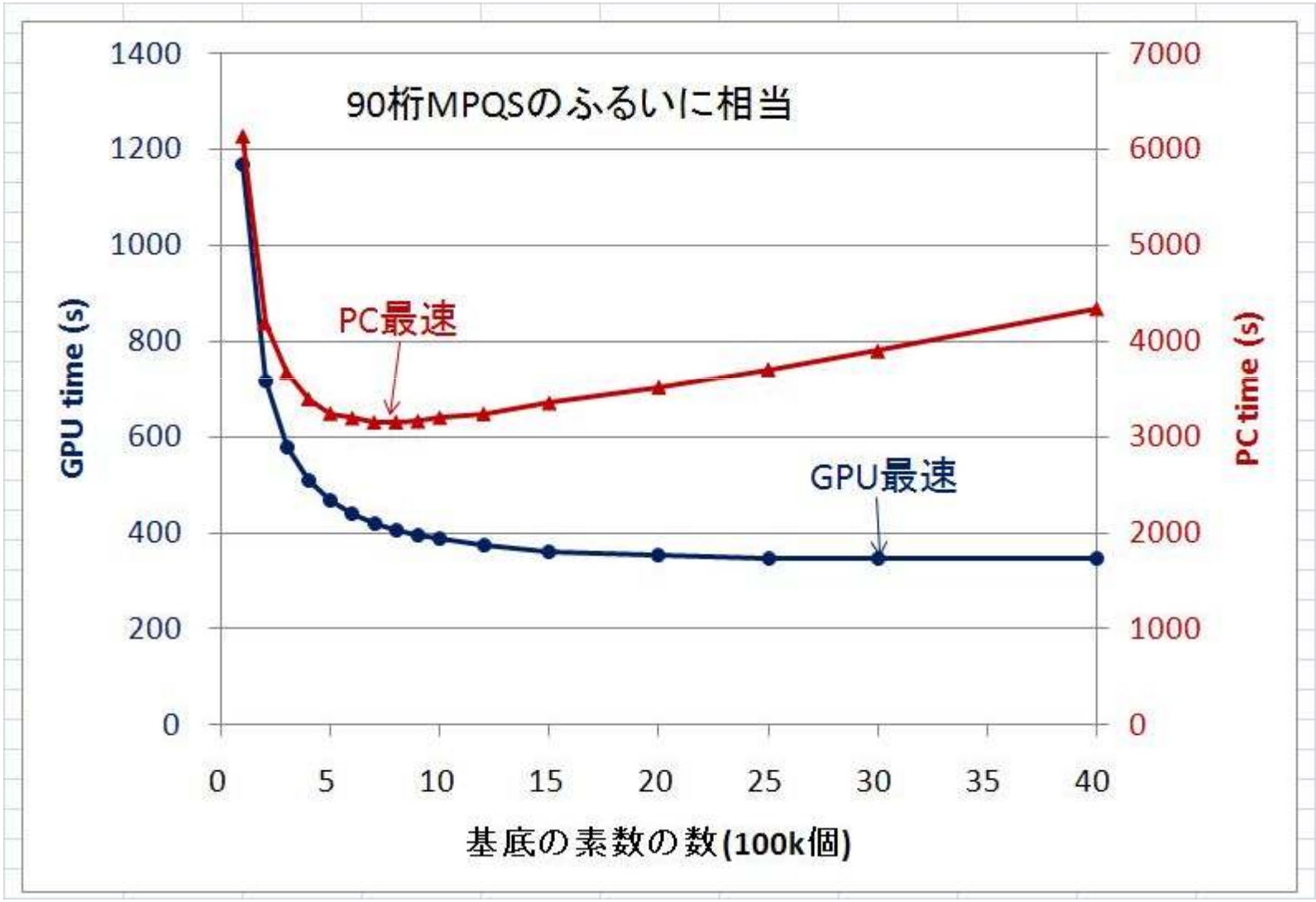
  NVIDIA GeForce GTX580 (1.544hz, 1.5GB)

  Unix, CUDA 3.2, -O3オプション

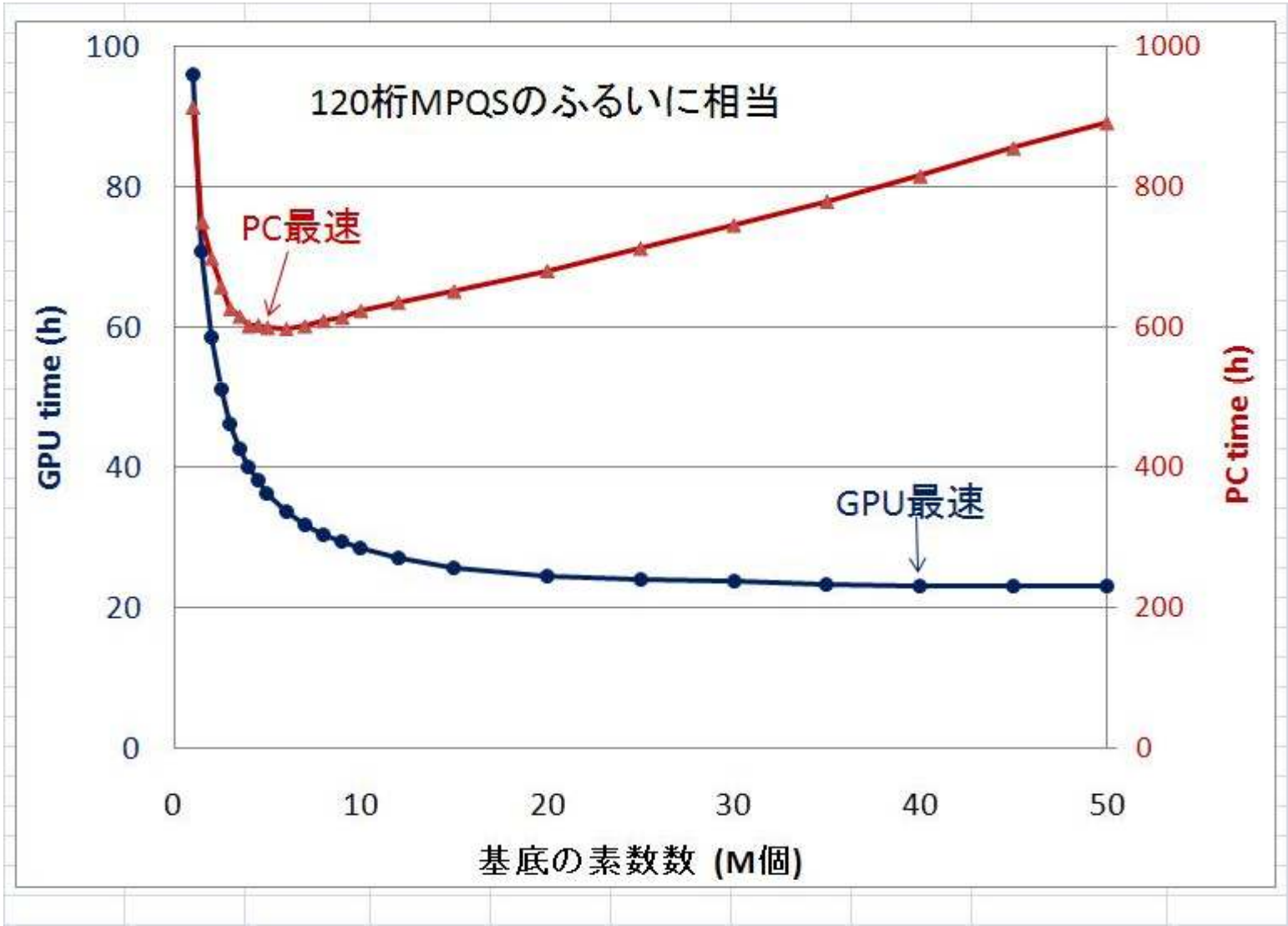
6.1 ふるい対策効果（10進45桁）



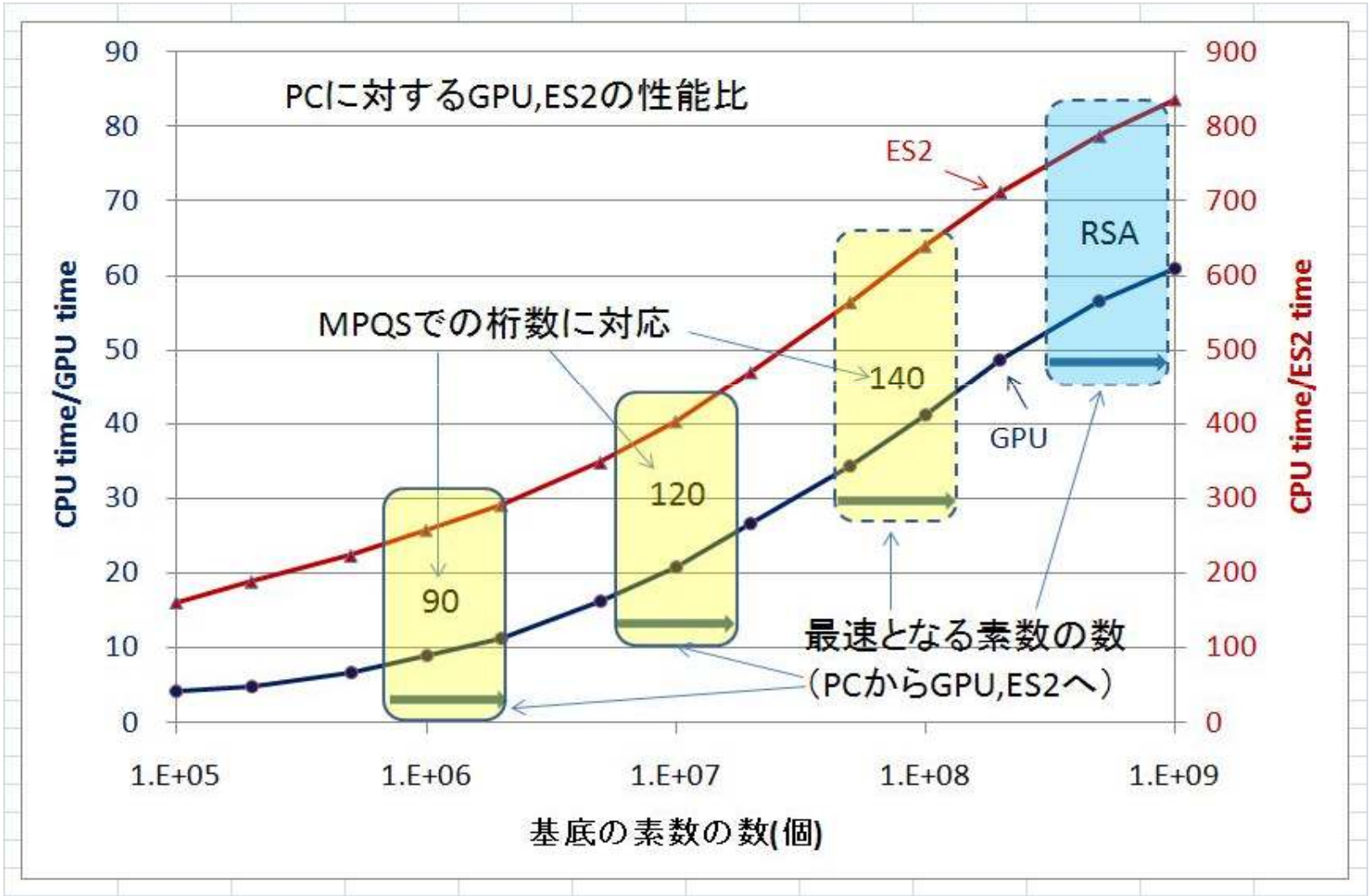
6.2 ふるい計算時間(10進45桁)



6.3 ふるい計算時間(10進60桁)



6.4 ふるいの性能比較


PCに対するGPU,ES2の性能比
CPU time/GPU time
CPU time/ES2 time
基底の素数の数(個)
ES2
GPU
RSA
MPQSでの桁数に対応
90
120
140
最速となる素数の数
（PCからGPU,ES2へ）
0
10
20
30
40
50
60
70
80
90
0
100
200
300
400
500
600
700
800
900
1.E+05
1.E+06
1.E+07
1.E+08
1.E+09

7. GNFSふるいでの考察

- GPUではPCより最速となる、基底の素数(素イデアル)の数は5～10倍多く使用できる
- GNFSのふるいは素数と素イデアルの両基底で、共に分解できるものを採取する
- GNFSでは 数値実験(MPQSを想定)よりGPUがPCに比較して更に有利になる可能性大(基底だけ大きく)

素数ふるい	素イデアルふるい	採取データ
100,000,000	100,000,000	100,000
200,000,000	200,000,000	400,000
400,000,000	400,000,000	1,600,000

8. おわりに

- ふるい処理においてGPU(GXT580)の性能は PC(2.33Ghz)の5倍～60倍程度
- GPUの性能は140桁のMPQSで約40倍、RSA暗号解読の規模で約60倍と推定されるGNFSでは、GPUの効果はより大きいと予想
- スレッド並列化は基底の素数の大きさで3段階に分け、大きい素数は素数で並列化
- 対策後は、データ依存性で約0.02%の採取不足があるが、ふるい処理では十分である

謝辞

GPU (NVIDIA GeForce GTX580)の
利用環境をご用意頂いた、

筑波大学　長谷川 秀彦　教授

に謹んで感謝の意を表します