ハイパフォーマンス
コンピューティング 54－9
（１９９４．１２．１４）

多倍長計算のＨＰＣ技術

太田昌孝　　前野年紀

東京工業大学　総合情報処理センター

多倍長計算の基礎となる整数多倍長乗算のプログラミングは、単純なアルゴリズムでは、桁数の二乗に比例した時間がかかる。このアルゴリズムのプログラミングに際して、行列の乗算に対して知られている高速コーディングの技法を各種適用してみた。多倍長の整数は、ビットごとに複数の６４ビット浮動小数点数に分割し表現した。キャリーの伝搬をまとめて行うことと、ループアンローリングによるレジスタブロッキングの技法を適用することにより、Ｃによるコーディングで１９８ＭＦＬＯＰＳの理論最大性能の約５０％を引きだすことができた。作成したプログラムは、公開鍵暗号の計算に必要な程度の桁数に対して極めて有効である。

HPC Technique on Multiple Precision Arithmetic

Masataka Ohta, Toshinori Maeno

Computer Center, Tokyo Institute of Technology

Efficient multiple precision integer multiply is implemented. as a basic tool for multiple precision arithmetic using the simple O(N^2) Algorithm. High performance computation techniques, that has been proven to be useful for efficient matrix multiply, is applied. Multiple precision integers are represented by a bit-sliced array of 64bit floating point numbers. By delayed carry propagation and by register blocking through loop unroling, about 50% of theoretical peak performance of 198MFLOPS as been extracted through C. The resulting program is practically useful for public key criptography.

1. はじめに

Ｎ桁の多倍長整数の四則演算に関して、加減算は桁数に比例したＯ（Ｎ）時間で行えることは明らかである。また、除算はニュートン法による近似を繰り返すことにより乗算と同じ計算量で行えることもよく知られている。乗算は、筆算同様の単純なアルゴリズムでは、Ｏ（Ｎ＾２）、高速フーリエ変換に基づいたアルゴリズムでは、Ｏ（ＮｌｏｇＮｌｏｇｌｏｇＮ）時間で行える。計算量のオーダーとしては明らかに後者が勝れているが、後者は、整数の剰余のような最近の計算機上では非常に遅い計算を必要とし、定数ファクターが大きいため、よほどＮが大きくないと実用的ではない。

ＲＳＡ方式に代表される公開鍵暗号の計算には、５００～２０００ビット程度の整数の多倍長計算を高速に行うことが重要であるので、計算機の浮動小数点演算能力を活用したＯ（Ｎ＾２）方式のプログラムを開発することは、実用上極めて有用となる。

そこで、［１］で、行列の乗算の高速化に有効であった各種技術を、多倍長整数乗算の高速化に適用してみた。

計測に利用したのは、ピーク性能１９８ＭＦＬＯＰＳの、ＨＰ９０００／７５５である。

2. データ構造

本稿では、多倍長整数は、ＬＭＩＮＴビットごとに区切り下の位から順に可変長の６４ビット浮動小数点数の配列に格納することとする。

ＩＥＥＥ標準の６４ビット浮動小数点数の仮数部は４８ビットあるので、２４ビットどうしの積まで計算しても誤差はでないが、後述する理由により、ＬＭＩＮＴとしては２０程度が適当である。とりあえずは１６としておく。

多倍長計算では、記憶領域は桁数に比例してしか消費しないので、計算結果を無理にコンパクトなかたちに直す必要はない。また、本稿の程度のビット数であれば、全体は楽にキャッシュにおさまる。

3. 素朴なプログラム

筆算の方式をそのままにプログラムすると、プログラム１が得られる。６４０ビット（１６ビット＊４０）の数二つに対して速度を計測したところ、一回あたり３．７ミリ秒かかった。浮動小数点乗算は４０＊４０＝１６００回、加算も同じ数だけ行われているので、これで換算すると、およそ０．８６ＭＦＬＯＰＳということになる。これはプログラム後半のキャリーの繰り上げに時間の大半をとられているためである。

```
#include <math.h>

#define MINT 65536
#define LMINT 16

mpmul(a,al,b,bl,c)
double *a,*b,*c;
int al,bl;
{int i,j;
double c0,c1,carry;
   for(i=0;i<al+bl;i++)
      c[i]=0;
   for(i=0;i<al;i++)
   {  for(j=0;j<bl;j++)
         c[i+j]+=a[i]*b[j];
      carry=0;
      for(j=0;j<bl;j++)
      {  c0=c[i+j]+carry;
         c1=fmod(c0,MINT.0);
         carry=(c0-c1)/MINT.0;
         c[i+j]=c1;
      }
      c[i+j]+=carry;
   }
}
```

プログラム１

4. キャリーの蓄積

そこで、キャリーを毎回繰り上げずにためておき、計算の最後にだけ伝搬するように、アルゴリズムを改良する。これが行えるためには、各回の浮動小数点乗算の結果をすべて足しても４８ビットを越えないようにする必要がある。

また、キャリーの計算をなるべく最適化する。ここでは、割り算系のライブラリ関数を使わず、逆数の乗算と、ＨＰのコンパイラが浮動小数点を整数に変換するとき小数部を切り捨てるという性質を利用した。

```
mpmul(a,al,b,bl,c)
double *a,*b,*c;
int al,bl;
{int i,j;
double c0,c1,carry;
   c1=ldexp(1.0,-LMINT);
   for(i=0;i<al+bl;i++)
      c[i]=0;
   for(i=0;i<al;i++)
      for(j=0;j<bl;j++)
         c[i+j]+=a[i]*b[j];
   carry=0;
   for(i=0;i<al+bl;i++)
   {  c0=c[i]+carry;
      carry=(int)(c0*c1);
      c[i]=c0-carry*MINT.0;
   }
}
```

プログラム２

このプログラムでは、配列ｃには浮動小数点の積がｍａｘ（ａｌ，ｂｌ）回、足される。そこで、配列ｃの要素には、ＬＭＩＮＴが１６であれば、４８－１６＊２＝１６ビットの余裕がある。ＬＭＩＮＴが２０であっても８ビットの余裕があるので、２＾８＊２０＝５１２０ビットまでの数どうしの乗算が可能で、公開鍵暗号への応用には十分である。

その結果、速度は２０．９ＭＦＬＯＰＳ換算にまで向上した。この時点で、計算時間の９０％は浮動小数点乗算／加算の部分に費やされていることがわかったので、この部分の改良にとりくむことになる。

5. ループアンロール

多倍長乗算においても、配列の乗算の場合と同様ｉとｊに関するループをそれぞれ４重にアンロールするとキャッシュからの１回の読みだしで何度も演算が行え多数のレジスタを有効につかった効率的な計算がおこなえる。

このような操作を施したのがプログラム３である。プログラム２では乗算／加算１回ごとにメモリ読み書きも１回ずつ必要であったが、プログラム３では、読みだし０．５回、書き込み０．２５回に減少している。

その結果、速度は６４ＭＦＬＯＰＳ換算にまで向上した。まだ、理論最大性能に比べて１／３程度の速度でしかないが、３７％の時間は桁上げが占めており、これをのぞくと実効１０３ＭＦＬＯＰＳと、まずまずの数値であり、これ以上の改良は難しそうである。

実際、［１］のソフトウェアパイプラインの技法も試してみたが、性能向上にはつながらなかった。

プログラム３で、ＬＭＩＮＴを２０として６４０ビットの二つの数値を掛けると３６マイクロ秒、２０００ビットだと、２２２マイクロ秒である。

6. おわりに

Ｈｉｇｈ　Ｐｅｒｆｏｒｍａｎｃｅ　Ｃｏｍｐｕｔｉｎｇの、ループアンローリングの手法により、数百〜数千ビットの多倍長計算を効率良く行うことができた。

参考文献

［１］前野年紀, 太田昌孝：最近のワークステーションアーキテクチャと行列乗算性能について，ＨＰＣ　４９－１, 情報処理学会, Ｏｃｔ．１９９３

```
mpmul(a,al,b,bl,c)
double *a,*b,*c;
int al,bl;
{int i,j;
double carry;
double a0,a1,a2,a3;
double b0,b1,b2,b3;
double c0,c1,c2,c3,c4,c5,c6,c7;
   for(i=0;i<al+bl;i++)
      c[i]=0;
   for(i=0;i<al;i+=4)
   {  a0=a[i];
      a1=a[i+1];
      a2=a[i+2];
      a3=a[i+3];
      c0=c[i];
      c1=c[i+1];
      c2=c[i+2];
      c3=c[i+3];
      for(j=0;j<bl;j+=4)
      {  c4=c[i+j+4];
         c5=c[i+j+5];
         c6=c[i+j+6];
         c7=c[i+j+7];
         b0=b[j];
         b1=b[j+1];
         b2=b[j+2];
         b3=b[j+3];
         c0+=a0*b0;
         c1+=a1*b0;
         c2+=a2*b0;
         c3+=a3*b0;
         c1+=a0*b1;
         c2+=a1*b1;
         c3+=a2*b1;
         c4+=a3*b1;
         c2+=a0*b2;
         c3+=a1*b2;
         c4+=a2*b2;
         c5+=a3*b2;
         c3+=a0*b3;
         c4+=a1*b3;
         c5+=a2*b3;
         c6+=a3*b3;
         c[i+j]=c0;
         c[i+j+1]=c1;
         c[i+j+2]=c2;
         c[i+j+3]=c3;
         c0=c4;
         c1=c5;
         c2=c6;
         c3=c7;
      }
      c[i+j]=c0;
      c[i+j+1]=c1;
      c[i+j+2]=c2;
      c[i+j+3]=c3;
   }
   carry=0;
   c1=ldexp(1.0,-LMINT);
   for(i=0;i<al+bl;i++)
   {  c0=c[i]+carry;
      carry=(int)(c0*c1);
      c[i]=c0-carry*MINT.0;
   }
}
```

プログラム 3