数理解析研究所講究録
920巻 1995年 161-164

17.

高速乗算システムの実現

元吉 文男
(電総研知能情報部)

17.1 はじめに

数式処理において任意精度の整数演算機能は必須であるが、普通の実装では乗算は桁数の二乗に比例するものがほとんどである。ここでは、小さな桁数の演算においても普通の実現法より高速に実行するプログラムを作成したので報告する。このプログラムによる乗算の計算量のオーダは $n^{log_2 3}$ である。

17.2 実現方法

高速乗算プログラムを作成するに当たり、汎用性を持たせるために以下の前提を置くことにした。

- 全部 C で書く。
  - 1 語 16 ビットで整数を表現
  - 小さな n でも高速に
- 乗算ハードを持っている/いない計算機に対応
  - 1 語の乗算はシステムのものを使用

乗算としてシステムの (C の) ものを使用するために、1 語を 16 ビットとした。これを符号無し整数として表現することにより、16 ビットの数同士の演算の繰り上げ計算も容易に実現できる。(C の整数は 32 ビット以上で表現されるものと仮定した。) もっと大きなビット数で表現して、シフトなどの論理演算だけで繰り上げも含めた乗算を記述することも考えたが、実験の結果 1 語 16 ビットの方

が高速であった。(アセンブラ等を使用すれば別の結果になる可能性もある。)

高速乗算プログラムで採用した方法は Karatsuba/Knuth によるものである。その計算原理は、同じ桁数の数 U, V の乗算を行なうときに、まず、b を桁数の半分程の数として

$$\begin{aligned} U &= bU_1 + U_2 \\ V &= bV_1 + V_2 \end{aligned}$$

と分解して、その乗算の式に

$$\begin{aligned} & (bU_1 + U_2)(bV_1 + V_2) \\ = \; & b^2 U_1 V_1 + b(U_1 V_2 + U_2 V_1) + U_2 V_2 \\ = \; & (b^2 + b)U_1 V_1 + b(U_1 - U_2)(V_2 - V_1) + (b+1)U_2 V_2 \end{aligned}$$

という変形を用いるものである。ここで、b の値を 2 のべき乗にとると、最後の行の $b^2 + b$ と $b + 1$ を掛ける部分はシフト演算で実現でき、しかもそのコストは桁数に比例するので桁数の 1 乗より大きいと予想される全体の計算量のオーダには影響がない。この式では、桁数が元の数の半分の乗算を用いているが、その使用回数が 2 番目の式では 4 回であるのに対し、3 番目では 3 回で済んでいる。これを再帰的に使用して乗算を実現するものである。

この方法による計算時間を考える。n 桁 $\times$ n 桁 に必要な計算時間を $M(n)$ とすると、c をある程度大きくとれば

$$M(2n) \leq 3M(n) + cn$$

となる。そこで

$$\begin{aligned} M(2^m) &\leq c2^{m-1} + 3M(2^{m-1}) \\ &\leq c2^{m-1} + 3(c2^{m-2} + 3M(2^{m-2})) \\ &\leq c(2^{m-1} + 3 \cdot 2^{m-2} + \cdots + 3^{m-1}) + 3^m M(1) \\ &= c2^{m-1}(1 + 3/2 + \cdots + (3/2)^{m-1}) + 3^m M(1) \\ &= c2^{m-1} 2 \cdot ((3/2)^m - 1) + 3^m M(1) \\ &= c(3^m - 2^m) + 3^m M(1) \end{aligned}$$

となり、

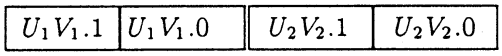
$$\begin{aligned} M(n) &\leq c(3^{\log_2 n} - n) + 3^{\log_2 n} M(1) \\ &\leq (c + M(1))3^{\log_2 n} \\ &= (c + M(1))n^{log_2 3} \\ &\simeq (c + M(1))n^{1.585} \end{aligned}$$

である。

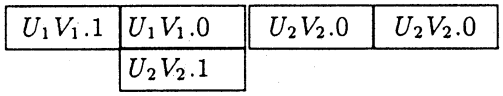
上の計算から分かるように、乗算の実行時間は各ステップでの桁数に比例する部分の係数 c に大きく依存している。そこで c を小さくするために

$$(b^2+b)U_1V_1+(b+1)U_2V_2$$

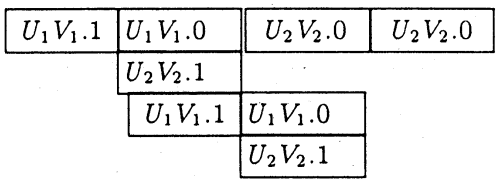
の部分の計算に以下のような方法を用いた。まず、U_1V_1 と U_2V_2 を再帰的に求めて STEP 1 のように並べておき、U_2V_2 の上位ブロックを U_1V_1 の下位ブロックに加え、U_2V_2 の下位ブロックを U_2V_2 部分に移す (STEP 2)。さらに、得られた結果の上位 2 ブロックを 1 つずつ下のブロックに加えると求める値が得られる (STEP 3)。この方法では、ブロックに対する操作が 4 回で済んでおり、普通に計算した場合の 6 回より少なくなっている。



STEP 1



STEP 2



STEP 3

このようにして計算した $(b^2+b)U_1V_1+(b+1)U_2V_2$ に $b(U_1-U_2)(V_2-V_1)$ を加えれば求める結果になるが、この後者の項は別の場所に求めておく必要がある。そのときに必要なメモリは桁数を n としたときに、$2\lceil n/2\rceil$ となる。そこで全体の乗算に必要なメモリ $S(n)$ は

$$S(n)=2\lceil n/2\rceil+S(\lceil n/2\rceil)$$

である。最悪の場合はいつも繰り上げになる場合で $n=2^m+1$ のときである。このときは

$$\begin{aligned} S(2^m+1) &= 2(2^{m-1}+1)+S(2^{m-1}+1) \\ &= 2^m+2+2^{m-1}+2+S(2^{m-2}+1) \\ &= 2^{m+1}+2(m+1)+S(2) \end{aligned}$$

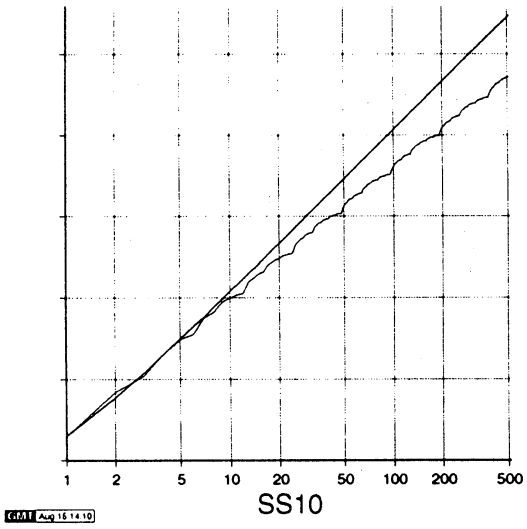
であるから、最悪の場合でも

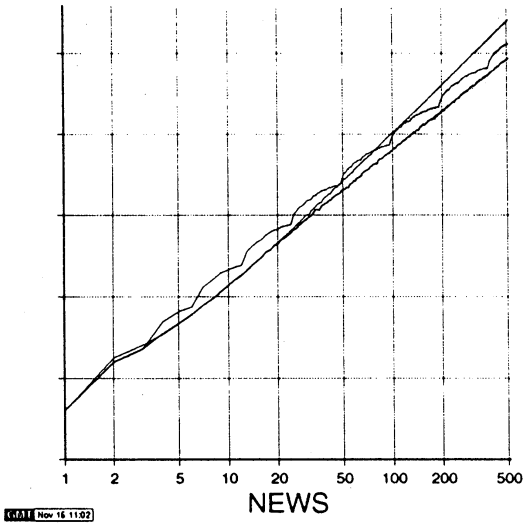
$$S(n)=2n+2\log_2(n-1)+2+S(2)$$

となる。

以上の方法を 2 つの異なるアーキテクチャの計算機で実現してみた結果を次に示す。CPU は SPARC と MIPS3000 である。(計算機は Sun SS2 と Sony NEWS である。) このうち SPARC ではハードウェアの乗算命令を使用せず、MIPS では使用している。MIPS の場合は、上記の方法だけの場合だと桁数が少ないときには、通常の方が早くなっているので、少ない桁数のときには通常の方法を使用する方法の結果も示してある。(図では MIPS のときの一番下のグラフがそれに相当する)。高速乗算

だけの場合には桁数が 2 のべき乗のところが早くなり、それ以外の桁のときには効率が悪くなっているのが分かる。





なおこの図で、縦軸の 1 目盛は 10 倍を示し、横軸の数字は 16 ビットを単位とした桁数である。

17.3 おわりに

以上のように当所の目的である高速乗算プログラムを作成したが、今後このプログラムを用いて以下のことを考えている。

- 除算の実現
- 浮動小数点数の四則
- 特殊関数の計算

これらの計算においては乗算の実行時間が支配的であり、乗算を高速に実行できることが全ての演算の高速化につながる。参考までに、n 桁の計算を実行するのに必要な時間を示しておく (by Brent)。

乗算の実行時間を $M(n)$ とし、π と $\log 2$ が前以って必要桁数だけ計算してあるとすると、

$$\begin{aligned} \text{除算} &: 4M(n) \\ \text{平方根} &: 11/2M(n) \\ \log, \exp &: 13M(n)\log_2 n \\ \tan^{-1}, \sin &: 34M(n)\log_2 n \end{aligned}$$

となる。なおπの計算時間は $15/2M(n)\log_2 n$ である。