Electronic Journal: Southwest Journal of Pure and Applied Mathematics
Internet: http://rattler.cameron.edu/swjpam.html
ISBN 1083-0464
Issue 1 July 2000, pp. 82 – 83
Submitted: July 2, 1999. Published: July 15, 2000

Fast Multiplication of Small Numbers

Jan-Christoph Puchta

Abstract

We give an algorithm to multiply two n-digit numbers with running time $O\left(\frac{n^2}{\log n}\right)$. Different from asymptotically better algorithms this algorithm outnumbers the standard algorithm allready for 11-digit numbers.

Subject Classification Codes: 68Q25
Keywords: Multiplication of Integers, Algorithm

Let f be a function with the following two properties: If $f(n)$ is known, $f(2n)$ can be computed at cost a, and if $f(n)$ and $f(m)$ are known, $f(n+m)$ can be computed at cost b, where obviously $a \leq b$. Then one can compute $f(n)$ recursively by computing $f\left(\lfloor \frac{n}{2} \rfloor\right)$, and then computing $f(n)$ with cost a resp. $a+b$ depending on whether n is even or odd. The expected running time to compute $f(n)$ is $a \log_2 n + \frac{b}{2} \log_2 n$. We will refer to this procedure as the standard algorithm.

If a and b are large compared to the cost of bitoperations, planing the computation can reduce the total costs, e.g. a^n can be computed with $(1+o(1))\log_2 n$ multiplications instead of $(\frac{3}{2}+o(1))\log_2 n$ multiplications by exploiting addition chains[2]. The aim of this note is to show that similar methods can be used to improve easier computations, too. The most important example is certainly the multiplication of integers.

Jan-Christoph Puchta, Albert-Ludwigs-Universität Freiburg
Mathematisches Institut, Eckerstr. 1, 79104 Freiburg Germany

Choose an integer $k \geq 2$ and compute $f(a)$ for all odd integers $< 2^k$. To compute $f(n)$ for odd n, compute $f\left(\left\lfloor \frac{n}{2^k} \right\rfloor\right)$. If n is odd, compute $f\left(2^k \left\lfloor \frac{n}{2^k} \right\rfloor\right)$ with cost $k \cdot a$, then $f(n)$ with additional cost b. For even n compute $\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$, then $f(n)$ with cost a.

Proposition 1

The total cost of the algorithm described above is $< a \log_2 n + \frac{b}{k} \log_2 n + 2^{k-1} b + a$. *Set* $f(n) = n \cdot m$ *and assume that* n *and* m *have* d *binary digits. Then the total cost of the algorithm above is* $\ll \frac{d^2}{\log d}$. *Moreover with* $k = 2$ *the expected running time of the algorithm above is less then the running time of the standard algorithm if* n *has more then 11 binary digits.*

Proof:

1. During the computation of $f(n)$ we have to do $\log_2 n)$ shifts of cost a, and at most $\frac{\log_2 n}{k}$ additions of cost b. Finally we can compute $f(n)$ for all $n \leq 2^k$ at total cost $2^{k-1} b + a$ by first computing $f(2)$ at cost a, then $f(2k+1)$ out of $f(2k-1)$ at cost b. □

2. For this special choice of f, b is $\sim \log_2 m$, however a is neglectable. With $k = \frac{1}{2} \log_2 d$ and the estimate in 1. we obtain total costs of $\frac{2d^2}{\log_2 d} + d^{3/2} \ll \frac{d^2}{\log_2 d}$. Now for $k = 2$ a saving of b is achieved for every pair of consecutive 1's in the binary representation of n. On the other hand we have to compute $f(3)$ with cost b. Thus in average a saving is achieved as soon as the expected number of consecutive pairs of 1's in the binary representation of n is > 2. Now a simple computation shows that this number equals $\frac{d}{6} + \frac{1}{6} + \frac{(-1)^{d-1}}{3 \cdot 2^{d-1}}$, which for $d \geq 11$ is indead > 2. □

Note that there is an algorithms by Karatsuba and Ofman with running time $O(d^{1.585})$ and even an algorithm with running time $O(n \log n \log \log n)$ by Schonhagen and Strasser[1], however, the constants implied by the O-symbols are so large that these algorithms are useless for computations with numbers with less than 1000 binary digits.

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The design and Analysis of Computer Algorithms*, Reading, Mass., Addison – Wesley 1974

[2] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Reading, Mass., Addison – Wesley 1969