Parabola Volume 37, Issue 3 (2001)

UNLIMITED PRECISION

John D. Steele[1]

Introduction

Everyone reading this article will be familiar with calculators. One of the problems we face using them is that they have a limited precision: usually 8 digits, sometimes 10. If you need to know that π to 50 places is

$$3.1415\,92653\,58979\,32384\,62643\,38327\,95028\,84197\,16939\,93751,$$

then a calculator is of no use.

However, as some of you may be aware, a computer armed with a type of software called a **computer algebra system** can easily cope with this, and can in fact do arithmetic to **arbitrary precision**, that is, to whatever accuracy you demand of it. Of course, you still have to be careful not to demand unreasonable accuracy: if you start with a collection of numbers accurate to 5 decimal places, then it is pointless asking for the average to 20 places.

There are several commercially available computer algebra systems, the two major ones being Mathematica (www.wolfram.com) and Maple (www.maplesoft.com). Such systems can of course do a lot more than just arithmetic: they are computer *algebra* systems, and so can perform such tasks as polynomial manipulation (addition, multiplication, factorization), solving equations, plotting graphs, differential and integral calculus and even geometry.

How do we get the computer to do these things? More importantly, how can we get the computer to do them quickly and efficiently?

Representing numbers

The first problem we face in answering the initial question is to ask how a computer algebra system on a finite computer storing data in small chunks of a fixed, finite size (e.g. 32 binary digits per word) can cope with holding and calculating with integers that have no pre-defined limit.

There are two ways of doing this. The simplest is to borrow the technique that we humans use. We can write and manipulate integers of arbitrary size by using only 10 symbols (or only two if we use base 2, as computers do internally). Computer algebra systems choose as a base for arithmetic a larger number, such as 10^4, and work in that base. The size of the base depends on the actual computer used. In most cases 10^4 is

[1]Dr John D. Steele is a Lecturer in the Department of Pure Mathematics at UNSW

chosen, and this is because $(10^4)^2 = 10^8$ will fit into a 32-bit word and leave one digit spare for carrying ($2^{32} \approx 4 \times 10^9$). In a computer with 16-bit words, the system would be set up to use base 10^2. The sign of the integer is kept separately as that is easy to keep track of.

For example, Maple on my PC uses base 10^4 and so stores the number $1\,234\,567\,890$ as

$$(12) \times 10^8 + (3456) \times 10^4 + (7890),$$

needing 3 words to do it.

The second way of storing large integers is to rely on a result known as the Chinese Remainder Theorem. Rather than storing and manipulating the integer itself you store and manipulate its remainders after division by each member of a collection of other numbers known as the **moduli**. These moduli must have the property that no two of them have any common factors — you could do this by picking only prime numbers, but this is not necessary. Suppose the collection of numbers you've chosen is n_1, n_2, ..., n_k: you would pick these numbers so that they all fitted inside one word, and that means the remainders would all fit inside one word. In fact, you can pick the moduli so that $\frac{1}{2}n_i$ fits inside one word for each i and store remainders between $-\frac{1}{2}n_i$ and $+\frac{1}{2}n_i$, storing the actual sign in a separate bit.

Let $N = n_1 n_2 \ldots n_k$ be the product of all the moduli. The Chinese Remainder Theorem says that if you know all the remainders then there is one and only one number with this collection of remainders between 0 and N, and also gives you a way of restoring the number from its remainders.

For example, with the moduli 16, 17 and 19 we can store numbers between -5168 and $+5168$ with three one-digit remainders and a bit for the sign. In this case -52 would be stored as $(-)[4, 1, -5]$, 45 as $(+)[-3, -6, 7]$ and so on.

Multiplying big numbers

Addition and subtraction are straightforward with either of these two methods. You have to take care of carrying digits sometimes in the first method, and of course you have to stick within the range $-N$ to N for the second.

For example, with the Chinese Remainder technique we can find $45 - 52$ just by subtracting the remainders for each modulus seprately, and then (if necessary) taking remainders in the sum. So $45 - 52$ can be stored as $[-7, -7, -7]$, since $7 - (-5) = 12$ has remainder -7 when divided by 19. We hardly need the Chinese Remainder Theorem to tell us that the difference is -7.

Multiplication would work the same way in the Chinese Remainder method: just multiply the remainders. Then we get -52×45 stored as $(-)[4, -6, 3]$, which the Chinese Reminder Theorem tells us is -2430, and (apart from the using the Chinese Remainder Theorem itself) we never need to calculate any product larger than 9×9.

With the first method, storing to base 10^4, we see the reason for using this number as base: when we multiply two numbers less than 10^4 we get a number less than 10^8. If a carry over is added to the product it will still never get beyond 10^9 *and so can still fit in a single word*. In practice this means that the software can rely on the internal

hardware of the actual computer to do the multiplications, which are therefore very fast, and only have to deal with the shifting and adding.

This is conceptually the same idea behind the times tables, which those of us of a certain vintage were drilled in at school. You only need to know how to multiply every possible pair of digits (and be able to add) in order to be able to multiply any two integers of any size. Thus you only need to know 45 different products, from 1×1 to 9×9, to do long multiplication. The computer algebra systems replace knowledge of the times table with the hardware arithmetic, and the software does the rest – the shifting and adding.

Karatsuba's Method

For integers with less than about 20 digits in base 10^4 (that is numbers up to about 10^{80}), the "classical method" of multiplication works about as efficiently as one could get. However, it suffers from the problem that an n-digit number needs about n^2 multiplications (as well as all the adding and shifting), so it will take four times as long to multiply two 40-digit integers as it does to multiply two 20-digit integers.

You might be happy enough with integers with 80 decimal digits, but this is small potatoes really — there are primes known with over $300\,000$ decimal digits. If two 80-digit integers could be multiplied in a second, it would take the classical method over 5 months to multiply two $300\,000$ digit integers, which is not good enough.

Happily, there are several clever ways of multiplying very large integers together. The simplest one is due to A. Karatsuba of the Soviet Union and was published in 1962. The trick here is to split the integers being multiplied into parts with a smaller number of digits, and use a bit of algebra. Suppose we have integers x and y and we write them in base B as

$$x = aB^{n/2} + b \qquad y = cB^{n/2} + d,$$

so a, b, c and d are $\frac{1}{2}n$-digit numbers. Then

$$xy = acB^n + (bc + ad)B^{n/2} + bd.$$

Here we have four products each of half the number of digits. If two n-digit numbers can be multiplied in time Kn^2 for a fixed K, then (ignoring the extra additions) this split up will take time $4 \times K(n/2)^2 = Kn^2$. It looks like we have saved nothing, but we do not need to find all four products. Since

$$bc + ad = (a + b)(c + d) - ac - bd,$$

we only need three products of $\frac{1}{2}n$-digit integers, not four.

If we then do the $\frac{1}{2}n$-digit products the same way, we can show that the time it takes to multiply two n-digit numbers by Karatsuba's method is proportional to $n^{\log_2(3)} \approx n^{1.6}$. To compare this to the classical method, under the same conditions as above, Karatsuba's method would multiply two $300\,000$ decimal digit numbers in only 5 days.