FAST MODULAR MULTIPLICATION USING 2-POWER RADIX

Colin D. Walter

Department of Computation
UMIST
PO Box 88
Manchester M60 1QD, UK
www.co.umist.ac.uk

Abstract. Details are given here of how to generalise Brickell's fast modular multiplication algorithm to when the number representations have a general 2-power radix. Correct action depends upon the satisfaction of a complicated inequality and speed upon the use of a redundant number system to enable parallel digit operations. The effect of varying the radix on the efficiency of hardware implementations is considered. Improved efficiency has repercussions in public key cryptography where the RSA encryption scheme may use this type of algorithm for its modular exponentiations. However, it is shown that there is no advantage in taking a very large radix, although a small increase above 2 is beneficial.

Key words: Modular Multiplication, Fast Computer Arithmetic, Digital Arithmetic, Integer Division, Parallel Bit Operations, Redundant Number Systems, RSA Algorithm, Cryptography

C.R.Categories F.2.1, B.7.1, E.3.

0 Introduction

We investigate here a parallel digit algorithm for the calculation of $A \times B \bmod M$. A modular product R of B with part of A is created iteratively by using the successive digits of A to add a multiple of B and predicting the right multiple of M to subtract at the next cycle. So the whole process takes a time proportional to the number of digits in A. There is an apparent advantage in throughput speed from increasing the radix of the representation in order to decrease the number of digits in A. Here we prove a very general algorithm and look closely at the major terms describing hardware area and clock speed in an implementation in order to clarify the best choice of radix.

The history of the subject includes the use of redundant number systems (e.g. [2]) applied to division ([1]). Arising from this, modular multiplication using these techniques is discussed by Simmons and Norris [7], and described for base 2 by Brickell [3], with proofs and further detail provided by Gibson [4] and Walter and Eldridge [8]. The recent rash of publications in this area also includes notable work by Selby and Mitchell [6]. The main application of the results here is in hardware implementations of the RSA algorithm [5] where the modular exponentiation of encryption and decryption is performed by repeated modular multiplications.

1 Notation

Numbers here are represented with 2-power radix $r = 2^\rho$, $\rho \geq 1$, making translation to or from a binary form fairly trivial. The digits of a number X are written x_i and lie in the range $0..x_{\max}$, say, so that X has the value $\sum_{i\geq 0} x_i r^i$. The $i{+}1$st digit of X is x_i, the coefficient of r^i, because the numbering starts at 0. In an irredundant system $x_{\max} = r{-}1$, which makes the choice of digits unique, and in a redundant system $x_{\max} \geq r$, which allows alternative representations. A typical choice might be $x_{\max} = 2r{-}1$ because full use is then made of the extra bits needed for a minimal, properly redundant system. Since hardware area is crucial $x_{\max}$ is kept below $2r$ so that a digit requires ρ bits if the representation is not redundant, and $\rho{+}1$ if it is redundant.

Digit ranges are chosen to suit applications. Here, we are interested in repeated modular multiplication to perform the modular exponentiation of the RSA two-key cryptosystem. Thus the modulus M is often fixed for many successive modular multiplications. Since M is usually in binary form, we obtain M in a non-redundant form to base r by taking its bits in groups of ρ. So we assume its digits are at most $m_{\max} = r{-}1$.

The algorithm presented here calculates a residue R and an integer quotient Q satisfying $A{\times}B = M{\times}Q{+}R$ where R is either the smallest non-negative residue of $A{\times}B \bmod M$ or differs by at most M from it. This occasional extra M can be subtracted separately if required. Suppose M has N digits. If the output R has more than N digits it is larger than M and can be reduced at virtually no cost. So we may assume R has at most N digits. Since A and B are initially in the range $0..M$ in the RSA algorithm, and the output is used as subsequent input, we can always assume A and B have at most N digits. Normally the hardware fixes N, whilst M is shifted to put it in the right range, resulting in appropriate shifts for the other quantities initially and finally. So we assume M lies in the range $r^N/2..r^N{-}1$ and also that each of A, B, R, Q and M is non-negative.

The multiplication is performed by repeated addition, but to perform addition in a single clock cycle requires each output digit to be dependent on only a small number of input digits, not on all lower-indexed digits as in the case of the usual (serial) hand calculation. So carries must propagate at most only a small fixed distance. This can only be done using a redundant number system for the output. Since in our application, output from one modular multiplication becomes input for the next, we expect A, B, and any intermediate partial product R to have a common redundant number system, say with digits $0..r_{\max}$. Later on, sensible values for $r_{\max}$ will be considered. The integer quotient Q also turns out to need a redundant representation, say with digits in the range $0..q_{\max}$.

One of the ideas tacit in Brickell's paper [3], which we make explicit here, is to calculate $A{\times}B \bmod M$ as $(AS{\times}B \bmod MS)/S$ for a fixed r-power $S = r^E$. So a shift up by E places is performed. Here the number E of extra digits needed at the top of the registers is large enough to make the contribution of B to the partial product R insignificant in a certain precise way. Henceforth, we therefore work with RS, MS, etc., which we consider as individual numbers, not products, with digits denoted rs_i, ms_i, etc.

To predict the right multiple of MS to subtract from intermediate partial products RS in order to reduce them $\bmod MS$ we look only at the top few digits of the numbers involved. The notation is that $\underline{X}$ denotes the number obtained by setting the lowest bits of a number X to 0 and E_X is the greatest number with the same representation as X which is zero when $\underline{\ \ }$ is applied. The error involved in using $\underline{X}$ as an approximation for X is expressed in $\underline{X} \le X \le \underline{X} + E_X$. If the lowest δ digits and β bits from the next digit are set to zero in X, then $E_X = x_{\max}(r^\delta-1)/(r-1) + r^\delta(2^\beta-1)$. So, in the non-redundant case, $E_X = r^\delta 2^\beta - 1$.

2 Software Algorithm

Next is a pseudo-Pascal version of the modular multiplication algorithm under study here. Missing detail and restrictions are derived later. In particular, L is the main constant yet to be determined, suggestions are needed for defining the function *ApproxQuot*, and the register sizes for R and Q are outstanding. It is an easy step subsequently to obtain $(A{\times}B) \bmod M$ and $(A{\times}B) \operatorname{div} M$ precisely, because we will find that $L < 2M$, so that the property $R < L$ for the output residue makes it either the least non-negative one, or the second least.

```
Const    S = r^E  ;
Type     Mdigits = 0..r-1;  Qdigits = 0..q_max; Rdigits = 0..r_max;
Procedure ModMult(A,  B  : Array[0..N-1] of Rdigits ;
                      M  : Array[0..N-1] of Mdigits ;
                  Var R  : Array[0..N+?] of Rdigits ;
                  Var Q  : Array[0..N+?] of Qdigits ) ;
{ Pre-Condition:   r^N/2 ≤ M < r^N,  L = ?   }
{ Post-Conditions: A*B = Q*M + R,  R ∈ 0..L }

Var      J   : 0..N+E-1 ;
         MS  : Array[0..N+E-1] of Mdigits ;
         AS  : Array[0..N+E-1] of Rdigits ;
         RS  : Array[0..N+E+?] of Rdigits ;

         Function ApproxQuot(rRS,MS : Array...) : Qdigits ;
         { Post-Condition : ApproxQuot ≈ rRS div MS }
         Begin ... End ;
```

```
Begin { ModMult }
        MS := Shift(M,E);  AS := Shift(A,E);  RS := 0;  Q := 0;
        For J := N+E-1 DownTo 0 do
        Begin
            { Invariant: RS ∈ 0..LS }
            Q[J] := ApproxQuot(rRS,MS) ;
            RS   := Shift(RS,1) + AS[J]*B - Q[J]*MS ;
        End ;
        R := Shift(RS,-E)
End ; { ModMult }
```

Using the subscripts of our notation so far rather than the indexing [_] of Pascal, if $AS = \sum_i as_i r^i$ then $RS_J \equiv (\sum_{i=J}^{N+E-1} as_i r^{i-J})B - (\sum_{i=J}^{N+E-1} q_i r^{i-J})MS$ satisfies $RS_{N+E} = 0$ and $RS_J \equiv rRS_{J+1} + as_J B - q_J MS$. Hence RS_J is the value of RS at the end of the iteration with J as the value of the control variable. Therefore the final iterate $RS = RS_0$ satisfies $RS \equiv AS{\times}B - Q{\times}MS$, and the output satisfies $A{\times}B = Q{\times}M + R$, with $R \leq L$ if the claimed loop invariant holds. When a suitable L is determined, the outstanding register sizes can be found easily, assuming representations of the numbers which require the maximum number of digits. In particular, we will find that R can be squashed into N digits.

3 Convergence

If one partial product RS is in the range $0..LS$, then we require that the next partial product, say $rRS{+}aB{-}qMS$, be in the same range so that the process converges – otherwise the output R may become too large. In this section we establish the property required of L for this to be the case.

Ideally, the choice of q should make the expression for the next value of RS minimally non-negative, allowing L to be chosen equal to M. However, this would require checking every digit of RS, aB and MS. Time is saved by using only the topmost bits of these inputs to generate an approximation for q. This forces a wider range $0..LS$ for the next RS, and possible extra final subtractions of M to achieve R in the interval $0..M$.

Since S will be chosen large enough to make the term aB relatively small, q will satisfy $rRS{-}qMS \approx 0$, yielding $q \approx rRS$ div MS, an approximate quotient. As the value of q is predicted by looking at only the top few digits of MS and RS, we define $q = \mathit{ApproxQuot}(r\underline{RS}, \underline{MS})$ for a suitable function. Here $\mathit{ApproxQuot}$ may be obtained from a pre-computed look-up table, parametrised only by $\underline{RS}$, which is re-set each time M is changed (cf Selby & Mitchell [6]). The table can be computed easily from the values of $\underline{qMS}$ as q varies, these values being uniquely determined because the representation of MS is not redundant.

Although the choice of $\mathit{ApproxQuot}$ is in the hands of the user, one possibility is $q = r\underline{RS}$ div $(\underline{MS} + E_{MS}{+}1)$. This is slightly less accurate than the tabulated values of q suggested above, but easier to prove properties about. The inequalities established from this definition of q will work even better for a more accurate approximation to $r\underline{RS}$ div $\underline{MS}$. Here E_{MS}+1, $r\underline{RS}$ and $\underline{MS}$ are all multiples of a very large power of 2, so that shifting them down leaves a simple definition of q. Because $MS \le \underline{MS} + E_{MS}+1$, we know

$$\begin{aligned} & rRS + aB - qMS \\ \ge\ & rRS - \{r\underline{RS} \text{ div } (\underline{MS} + E_{MS}+1)\}\times MS \\ \ge\ & rRS - \{rRS \text{ div } MS\}\times MS > 0 \end{aligned}$$

which is the desired lower bound. Similarly, an upper bound is obtained from

$$\begin{aligned} & rRS + aB - qMS \\ =\ & rRS + aB - q(MS+E_{MS}+1) + q(E_{MS}+1) \\ =\ & rRS + aB - \{r\underline{RS} \text{ div } (\underline{MS} + E_{MS}+1)\}\times(MS+E_{MS}+1) \\ & \qquad +\{r\underline{RS} \text{ div } (\underline{MS} + E_{MS}+1)\}\times(E_{MS}+1) \\ \le\ & rRS + aB - \{(rRS - rE_{RS}) \text{ div } (MS+E_{MS}+1)\}\times(MS+E_{MS}+1) \\ & \qquad +(rRS \text{ div } MS)\times(E_{MS}+1) \\ =\ & X + rE_{RS} + aB - (X \text{ div } Y)\times Y + (rRS \text{ div } MS)\times(E_{MS}+1) \\ & \qquad \text{where } X = rRS - rE_{RS} \text{ and } Y = MS+E_{MS}+1 \\ =\ & X \bmod Y + rE_{RS} + aB + (rRS \text{ div } MS)\times(E_{MS}+1) \\ <\ & Y + rE_{RS} + aB + (rRS \text{ div } MS)\times(E_{MS}+1) \\ \le\ & MS + E_{MS} + 1 + rE_{RS} + {r_{\max}}^2(r^N-1)/(r-1) + (rLS \text{ div } MS)\times(E_{MS}+1). \end{aligned}$$

This last expression is larger than MS, but only by a little if S is large enough and sufficiently many top bits are used. It is almost the value to choose for LS, which we need to choose slightly larger.

The other constraint required for the convergence of the algorithm is that *ApproxQuot* produces a result in the range $0..q_{\max}$. Certainly $q \ge 0$ in this case. Because here, as in any definition of q, $rRS - qMS \ge 0$, we know $q \le rRS \text{ div } MS \le rLS \text{ div } MS$ if we pick LS as intended. Hence $q \le q_{\max}$ if $rLS \text{ div } MS \le q_{\max}$. Now, with this assumption, define

$$LS = MS + E_{MS} + 1 + rE_{RS} + {r_{\max}}^2(r^N-1)/(r-1) + q_{\max}(E_{MS}+1).$$

Then the working of the last paragraph proves $rRS + aB - qMS < LS$, which is the required upper bound to ensure that the loop invariant is satisfied. So the process will converge as long as $rLS \text{ div } MS \le q_{\max}$.

Inserting the value for LS in this assumption yields the sufficient condition

$$r\{MS + rE_{RS} + {r_{\max}}^2(r^N-1)/(r-1) + (q_{\max}+1)(E_{MS}+1)\} \text{ div } MS \le q_{\max}$$

for everything to work, or, equivalently,

$$rE_{RS} + {r_{\max}}^2(r^N-1)/(r-1) + (q_{\max}+1)(E_{MS}+1) < (q_{\max}-r+1)MS/r.$$

This is clearly sharpest when MS is at a minimum and illustrates that Q must have a redundant representation, i.e. there is only a solution if at least $q_{\max} \geq r$ holds. It is certainly solved by taking S large enough and using sufficiently many digits to approximate MS and RS for any fixed such $q_{\max}$. Since the general upper bound $q_{\max} < 2r$ was an initial hypothesis, $rLS \operatorname{div} MS \leq q_{\max} < 2r$ so that $L < 2M$, proving the contention that the output is at most M larger than the minimum non-negative residue.

4 A Solution

Assume that the lowest $\alpha+E$ digits and the lowest β bits of the next digit are set to 0 in MS by $\_\_$, and that $\sigma+E$ digits and τ bits are annihilated similarly in the case of RS. Moreover, suppose as usual that $M \geq M_{\min} = r^N/2$. The final inequality of the last section may now be re-written as

$$r(r_{\max}(r^{\sigma+E}-1)/(r-1) + r^{\sigma+E}(2^\tau-1)) + {r_{\max}}^2(r^N-1)/(r-1) + (q_{\max}+1)r^{\alpha+E}2^\beta$$
$$< (q_{\max}-r+1)r^{N+E}/2r$$

This is what we need to satisfy for the algorithm to behave.

If q is obtained from a look-up table pre-computed from $\underline{MS}$ and indexed by $\underline{RS}$, then we want to minimise the number of significant bits in $\underline{RS}$, that is, maximise $\sigma(\rho+1)+\tau$. To do this, take $q_{\max}$ as large as possible, viz $2r-1$. Then, from choosing just the two most significant terms on the left in the above, at least

$$r^{\sigma+E+1}2^\tau + r^{\alpha+E+1}2^{\beta+1} < r^{N+E}/2$$

must hold, giving $\sigma = N-2$, $\tau = \rho-2$ and $\alpha = N-2$, $\beta = \rho-4$ as the most optimistic solution possible.

How many bits of RS are needed? Using $RS < LS < 2MS < 2r^{N+E}$, there may be 1 significant bit in the $N+E+1$st digit of RS. Also $\rho+1$ bits come from the $N+E$th digit, and 3 from the $N+E-1$st, since $\rho+1$ are needed for each digit. This is a total of $\rho+5$ significant bits in $\underline{RS}$. However, it is easy to OR the only bit of the $N+E+1$st digit with the $\rho+1$st bit of the $N+E$th digit to reduce this by 1 (so the $N+E+1$st digit of RS is not needed). For $\underline{\text{MS}}$ only $\rho+4$ bits are needed if the above solution works because MS has a non-redundant form with only $N+E$ digits. In this case, however, MS will have been shifted up initially so that its most significant bit is the top one of its $N+E$th digit. So only $\rho+3$ bits of MS affect the computation of *ApproxQuot*. Thus, direct calculation of that function requires a total input of $2\rho+8$ bits if the above choices satisfy our inequality.

When does the proposed solution above not work? Substituting the values into the inequality yields

$$r(r_{\max}(r^{N-2+E}-1)/(r-1) + r^{N-2+E}(r/4-1)) + {r_{\max}}^2(r^N-1)/(r-1) + r^{N+E}/8$$
$$< r^{N+E}/2$$

Suppose we insist, reasonably, that $r_{\max} \leq 2(r-1)$. Then this holds if

$$2r(r^{N-2+E}-1) + r^{N-1+E}(r/4-1) + 4(r^N-1)(r-1) < 3r^{N+E}/8$$

i.e. if

$$r^{N-1+E} - 2r + 4(r^N-1)(r-1) < r^{N+E}/8$$

For large enough E, this is therefore satisfied if $r > 8$, i.e. $r \geq 16$, or $\rho \geq 4$, which, of course, is the only range of values for which the choice of $\beta(\geq 0)$ makes sense.

The largest value needed for E is when r is least, i.e. $r = 16$. The last inequality is satisfied if

$$r^{E-1} + 4(r-1) \leq r^E/8$$

i.e. if $4(r-1) \leq r^{E-1}(r/8-1)$, and so if $4r \leq r^{E-1}$. Thus $E = 3$ suffices for $r \geq 16$, with $E = 2$ easily if $r \geq 64$.

5 Summary

The general radix version of Brickell's modular multiplication algorithm given in section 2 works when L is given by

$$LS = MS + E_{MS} + 1 + rE_{RS} + {r_{\max}}^2(r^N-1)/(r-1) + q_{\max}(E_{MS}+1)$$

and $q_{\max} = 2r-1$, $r_{\max} < 2r-1, M_{\min} = r^N/2$, $\underline{MS}$ is obtained by setting the lowest $N+E-2$ digits and the lowest $\rho-4$ bits of the next digit to 0, $\underline{RS}$ is obtained similarly by setting the lowest $N+E-2$ digits and next $\rho-2$ bits to 0, $E = 3$ for $r \geq 16$, $E = 2$ for $r \geq 64$, and $r \geq 16$.

We do not need to work out LS explicitly, but we know it is less than $2MS$, so that the output R is bounded by $2M$. The least value $M_{\min}$ of M can be justified by shifting M up by a power of 2 to make full use of the Nth digit.

Solutions exist to the inequality for the values 2, 4 and 8 of r and can be obtained in exactly the same way as above. This is also true for different choices of $r_{\max}$ and $q_{\max}$ provided they each allow redundancy.

6 Hardware and Efficiency

In the loop each addition and the generation of the next q can be done together in a single clock cycle, with the result that $N+E$ cycles are required for the algorithm to execute. Clock speed is bound by the number of gates on the longest or critical path, which is found by adding the lengths of the critical paths needed to compute q and to perform the addition $rRS+aB-qMS$. The addition, which includes the digit multiplications, is like adding about $2\rho+6$ binary numbers, and so has a depth in the order of $log_2(2\rho+6)$. However, binary carries must then propagate over the length of the digit, so that the critical path length has an effective length of order ρ.

The number of bits for determining q has order ρ and so a look-up table would require a critical path of the order $log_2\rho$, leaving the adder as the main source of length in the critical path.

Thus the clock cycle time is asymptotically proportional to ρ. Also, the number of iterations in the loop is asymptotically proportional to the number of digits, and so to $1/\rho$. This means that the throughput time is asymptotically constant as r is increased, and no real benefit accrues from having a large base.

However, for small r, the various switching circuits etc. independent of r will dominate critical path lengths, so that a modest increase in r beyond 2 should initially reap a reward, but that return will decrease every time r is further increased.

The area of the chip is dominated by that of the adder, and the registers containing A, B, M and R. Again, the common hardware for any choice of r dominates for very small r. The adder can be made from around $3(2\rho+6)$ gates for each bit position, giving an area $3N\rho(2\rho+6)$ for the adder. This is proportional to ρ as r is increased. Hence chip efficiency measured as the inverse of the product of time by area actually decreases eventually as r is increased. However, one immediate advantage of increasing the base is that the registers contain fewer bits: for a given natural number, the number of bits required in our redundant systems is proportional to $(\rho+1)/\rho$, which makes the step from base 2 to $r = 4$ or 8 quite attractive.

7 Conclusion

A very general modular multiplication algorithm has been presented and verified and the efficiency of hardware implementations discussed. Although the order estimates supply an accurate view of large-scale behaviour, lower order terms dominate for small values of the base r of the number representations. It seems best to choose the base $r > 2$, but not much larger.

References

1. Atkins, D.E., *Higher-radix division using estimates of the divisor and partial remainders*, IEEE Trans. Comp. **C17** (1968) 925-934.
2. Avizienis, A., *Signed Digit Number Representations for Fast Parallel Arithmetic*, IEEE Trans. Elec. Comp. **EC10** (1961) 389-400.
3. Brickell, E.F., *A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography*, Advances in Cryptology (Proc. of CRYPTO 82), Chaum et al. eds., Plenum, 1983, 51-60.
4. Gibson, J.K., *A generalisation of Brickell's algorithm for fast modular multiplication*, Bit **28** (1988) 755-763.
5. Rivest, R.L., Shamir, A., & Adleman, L., *A Method of obtaining Digital Signatures and Public Key Cryptosystems*, Comm. ACM **21** (1978) 120-126.
6. Selby, A., & Mitchell, C., *Algorithms for software implementations of RSA*, IEE Proc. **136**, part E, (1989) 166-170.

7. Simmons, G.J., & Norris, M.J., *High speed arithmetic utilizing redundant number systems*, Nat. Telecom. Conf. 1980, Houston, Texas, IEEE, New York, 1980, 49.3/1-2.
8. Walter, C.D., & Eldridge, S.E., *A Verification of Brickell's Fast Modular Multiplication Algorithm*, Intern. J. Computer Math. **33** (1990) 153-169.