Short Notes

On the Addition of Binary Numbers

R. BRENT

Abstract—An upper bound is derived for the time required to add numbers modulo 2^n, using circuit elements with a limited fan-in and unit delay, and assuming that all numbers have the usual binary encoding. The upper bound is within a factor $(1+\varepsilon)$ of Winograd's lower bound (which holds for all encodings), where $\varepsilon \to 0$ as $n \to \infty$, and only $0(n \log n)$ circuit elements are required.

Index Terms—Addition, binary numbers, computational complexity, group multiplication, logic circuits, logical design.

I. Introduction

Winograd [3] and Spira [2] have considered the time required to perform group multiplication, using logical circuits consisting of elements which have a limited number $r \geq 2$ of input lines, and which compute a logical function of their inputs with unit delay. For a precise definition of the mathematical model, see [3]. We consider the special case (also considered by Ofman [1]) of addition of nonnegative integers modulo 2^n, and we assume a 2-valued logic. Winograd [3] has given the lower bound

$$\tau(n) \geq \lceil \log_r (2n) \rceil \qquad (1)$$

for the time $\tau(n)$ required for this addition, and Spira [2] has shown that

$$\tau(n) \leq 1 + \left\lceil \log_r \left(\frac{n}{\lfloor r/2 \rfloor} \right) \right\rceil. \qquad (2)$$

Here $\lceil x \rceil$ denotes the smallest integer $y \geq x$, and $\lfloor x \rfloor$ denotes the largest integer $y \leq x$.

Since the lower and upper bounds (1) and (2) differ by at most 1, the problem might be regarded as essentially solved. Unfortunately, though, the circuit which Spira constructs to prove (2) is not at all practical, for it has 2^n output lines and at least 2^n circuit elements. Hence, it is interesting to impose the restriction that all numbers (both input and output) have the usual binary representation. In this paper we shall show that, even with this restriction, it is possible to get within a factor $(1+\varepsilon)$ of the lower bound (1), where $\varepsilon \to 0$ as $n \to \infty$, and the circuit which does this requires only a modest number, $0(n \log n)$, of circuit elements. We shall only consider addition, although it would be interesting to consider the computation of other functions, for example multiplication, under the same restrictions. For a discussion of multiplication, see [4].

II. The "Carry" Function

If $a_n a_{n-1} \cdots a_2 a_1$ and $b_n b_{n-1} \cdots b_2 b_1$ are n-bit binary numbers with sum (mod 2^n) $d_n d_{n-1} \cdots d_2 d_1$, all numbers having the usual encodings, then

$$d_i = a_i \oplus b_i \oplus c_{i-1}$$

where

$$c_0 = 0 \quad \text{(i.e., false)},$$

$$c_i = x_i \wedge (y_i \vee c_{i-1}),$$

$$x_i = a_i \vee b_i,$$

and

$$y_i = a_i \wedge b_i \qquad (1 \leq i \leq n).$$

Here $\oplus$ means the sum mod 2, and in the usual terminology c_i is the carry from bit position i, x_i is a "carry propagate" condition, and y_i is a "carry generate" condition. From this formulation we see that the time $\tau(n)$ for addition mod 2^n is bounded by

$$\tau(n) \leq t(n-1) + K_r, \qquad (3)$$

where $t(n)$ is the time required to compute

$$c_n = x_n \wedge (y_n \vee (x_{n-1} \wedge (y_{n-1} \vee \cdots \vee (x_1 \wedge y_1) \cdots))), \qquad (4)$$

and

$$K_r = \begin{cases} 3 & \text{if } r = 2 \\ 2 & \text{if } r > 2. \end{cases}$$

Because of the inequality (3), we shall concentrate on finding a good upper bound for $t(n)$. We never need to use the fact that $x_i = 0$ implies $y_i = 0$ when x_i and y_i arise as above. Since c_n of (4) depends on each of $x_n, \cdots, x_1$ and $y_n, \cdots, y_1$, a simple fan-in argument (see [3]) gives the lower bound

$$t(n) \geq \lceil \log_r (2n) \rceil, \qquad (5)$$

and it is also easy to see that $t(n) = 1$ if and only if $r \geq 2n$.

III. An Upper Bound for $t(n)$

The main result of this paper is the following theorem, which gives an upper bound for $t(n)$ and hence, by (3), for $\tau(n)$.

Theorem 1: For any integers $k \geq 1$ and $r \geq 2$,

$$t(r^{k(k-1)/2}) \leq k(k+1)/2. \qquad (6)$$

Before proving Theorem 1, we need some preliminary results. Lemma 1 shows how we may compute $c_n(x_n, y_n, \cdots, x_1, y_1)$ if the inputs $x_n, y_n, \cdots, x_1, y_1$ are split into p groups.

Lemma 1: If

$$n = \sum_{j=1}^{p} q_j, \qquad p \geq 1, \quad q_j \geq 1,$$

$$s_i = \sum_{j=1}^{i} q_j, \qquad (s_0 = 0),$$

$$X_i = x_{s_i} \wedge \cdots \wedge x_{s_{i-1}+1},$$

$$D_i = X_p \wedge \cdots \wedge X_{i+1}, \qquad (D_p = 1, \text{i.e., true}),$$

$$E_i = x_{s_i} \wedge (y_{s_i} \vee \cdots \vee (x_{s_{i-1}+1} \wedge y_{s_{i-1}+1}) \cdots),$$

Manuscript received November 26, 1969. This work was supported by CSIRO, Australia.

The author is with the Department of Computer Science, Stanford University, Stanford, Calif. 94305.

and

$$F_i = D_i \wedge E_i \quad \text{for } 1 \le i \le p,$$

then

$$x_n \wedge (y_n \vee \cdots \vee (x_1 \wedge y_1) \cdots) = F_1 \vee \cdots \vee F_p. \qquad (7)$$

Proof: If the left side of (7) is true, then there is some j, $1 \le j \le n$, such that $y_n = \cdots = y_{j+1} = 0$, $y_j = 1$, $x_n = \cdots = x_j = 1$. Choose i such that $s_i \ge j > s_{i-1}$. Then $D_i = 1$ and $E_i = 1$, so $F_i = 1$, and the right side of (7) is true. Conversely, suppose the right side of (7) is true. Then some F_i is true, so $D_i = E_i = 1$. Since $E_i = 1$, there is a j, $s_i \ge j > s_{i-1}$, such that $y_j = 1$ and $x_{s_i} = \cdots = x_j = 1$. Since $D_i = 1$, we also have $x_n = \cdots = x_{s_i+1} = 1$, so the left side of (7) is true.

Lemma 2 is a simple application of Lemma 1. For simplicity we state it in the case where each of the p groups has the same size q.

Lemma 2: If

$$n = pq \qquad (p, q \ge 1)$$

and

$$L(x) = \begin{cases} 0 & \text{if } x < 1 \\ \lceil \log_r x \rceil & \text{if } x \ge 1 \end{cases}$$

then

$$t(n) \le 1 + L(p) + \max(t(q), L(q) + L(p-1)). \qquad (8)$$

Proof: We may compute the E_i of Lemma 1 in time $t(q)$, and simultaneously compute the X_i, followed by the D_i, in time $L(q) + L(p-1)$ (the worst case is the computation of D_1), when each $q_j = q$. Thus, in time $\max(t(q), L(q) + L(p-1))$ we are ready to compute the F_i (in unit time), and then $F_1 \vee \cdots \vee F_p$ in time $L(p)$.

If $q = p'q'$, (8) gives an upper bound for $t(q)$ in terms of $t(q')$, and this may be substituted into (8) to give another bound for $t(n)$, and so on. Lemma 3 is the strongest result that we can get in this way when n is a power of r.

Lemma 3: If a and k are nonnegative integers and $T(x) = t(r^x) - x$, then

$$T(a + kT(a) + k(k-1)/2) \le k + T(a). \qquad (9)$$

Proof: From Lemma 2 with $p = r^b$, $q = r^c$, we have $t(r^{b+c}) \le 1 + b + \max(t(r^c), b + c)$, so

$$T(b + c) \le 1 + \max(T(c), b). \qquad (10)$$

Now (9) is trivially true if $k = 0$, and putting

$$b = k + T(a)$$

and

$$c = a + kT(a) + k(k-1)/2$$

in (10) gives

$$T(a + (k+1)T(a) + k(k+1)/2)$$
$$\le 1 + \max(T(a + kT(a) + k(k-1)/2), \quad k + T(a)),$$

so the result follows by induction on k.

We are now ready to prove Theorem 1. Since $T(0) = t(1) = 1$, Lemma 3 with $a = 0$ gives $T(k(k+1)/2) \le k + 1$. Replacing k by $k - 1$, the theorem follows by the definition of T. Note that if we knew a good upper bound for $t(r^a)$ for some positive a, Lemma 3 could be applied in the same way to give a result like Theorem 1, but perhaps stronger. In the next section, Theorem 1 is used to obtain an upper bound for $t(n)$, even if n is not a power of r. While this is enough for our purposes, we could probably get better results for a particular n by going back to Lemma 2 or Lemma 1.

IV. Concluding Remarks

Given $n \ge 1$, we may choose $k \ge 1$ such that $r^{k(k-1)/2} \ge n \ge r^{(k-1)(k-2)/2}$. Clearly then

$$t(n) \le k(k+1)/2$$

and so, by (3),

$$\tau(n) \le 3 + k(k+1)/2. \qquad (11)$$

An easy corollary is the following.

Theorem 2: For any given $\varepsilon > 0$ and $r \ge 2$,

$$\tau(n) \le (1 + \varepsilon) \log_r n \qquad (12)$$

for all sufficiently large n, even if all numbers have the usual binary representation.

Theorem 2 shows that the lower bound (1) is almost achievable, for sufficiently large n, even with our restriction to binary encodings. The circuit constructed by following the proof of Theorem 1 looks rather like a conventional multistage carry look-ahead adder, with the number of levels of look-ahead depending on k. The number $s(n)$ of circuit elements required to compute c_n with $n = r^{k(k-1)/2}$ in time $k(k+1)/2$ can be estimated from the proof of Theorem 1. If the X_i and D_i of Lemma 1 are computed in an economical way, we find that $s(n) < 7n$. Even if n is not of the form $r^{k(k-1)/2}$, we can show that $s(n) = 0(n)$, and the number of elements required to add n-bit numbers in time satisfying (12) is 0 ($n \log n$). The argument is straightforward, but rather tedious, so it is omitted.

Finally, we note that Theorem 1 does not give the best possible upper bound for all $k \ge 1$ and $r \ge 2$. If $k = 2$ and $r = 3$, we may compute $z_1 = y_3 \vee (x_2 \wedge y_2)$ and $z_2 = x_2 \wedge x_1 \wedge y_1$, both in unit time, and then $c_3 = x_3 \wedge (z_1 \vee z_2)$ in unit time, so $t = 2$ rather than 3. The computation of $t(n, r)$ and $\tau(n, r)$ for moderate n and r seems to be a difficult combinatorial problem.

Acknowledgment

The author wishes to thank R. Floyd for suggesting the problem.

References

[1] Yu. Ofman, "On the algorithmic complexity of discrete functions," *Dokl. Akad. Nauk SSSR*, vol. 145, pp. 48–51, January 1962.

[2] P. Spira, "The time required for group multiplication," *J. ACM*, vol. 16, pp. 235–243, April 1969.

[3] S. Winograd, "On the time required to perform addition," *J. ACM*, vol. 12, pp. 277–285, April 1965.

[4] ——, "On the time required to perform multiplication," *J. ACM*, vol. 14, pp. 793–802, October 1967.