Multiplication of signed-digit numbers

Ç.K. Koç and S. Johnson

Indexing terms: Digital arithmetic, Signed-digit numbers

A recently proposed technique for common-multiplicand multiplication of binary numbers is shown to be applicable to signed-digit numbers. The authors prove that multiplication of a single k-bit multiplicand by n k-bit multipliers can be performed using a 0.396n additions for canonically recoded signed-digit numbers, whereas the binary case requires 0.375n additions.

Multiplication of binary numbers: Let X and Y, Y2, ..., Yn be k-bit 2's complement or unsigned binary numbers such that n ≥ 2. To compute the products Pi, i = 1, ..., n, such that Pi = X × Yi for all 1 ≤ i ≤ n. Applications of this computation are found in cryptography; for example, the RSA algorithm [6] requires computation of modular exponentiations. The exponentiation operation is broken into a series of squaring and multiplication operations by the use of the binary method [3]. The right-to-left binary method performs a series of multiplication operations, in which a common multiplicand is multiplied by several multipliers.

The standard algorithm computes Pi = X × Yi, separately for each i, which takes n multiplications. Assuming that each Yi is a k-bit quantity, each multiplication requires on average k/2 additions, because randomly distributed k-bit binary numbers will have a Hamming weight of k/2. Thus, the standard algorithm requires nk/2 additions in the average case.

A more efficient method is given in [7]. Let i ≤ n. We first compute Yi = Yi × Yi × ... × Yi, where ⊕ is the bit-wise AND operation. We then compute Yi, for all i ≤ n, such that Yi = Yi ⊕ Yi, where ⊕ is the bit-wise XOR operation. It can be easily seen that Yi = Yi + Yi. Thus, Pi = X × Yi = X × (Yi + Yi) = (X × Yi) + (X × Yi). It was shown in [7] that, using this technique, only 3nk/8 = 0.375n additions will be required in the average case to perform n k-bit common-multiplicand multiplications.

Signed-digit numbers: Recording techniques (Booth recoding, bit-pair recoding, etc.) for sparse representations of binary numbers have been effectively used in multiplication algorithms [4]. For example, the original Booth recoding technique scans the bits of the multiplier one bit at a time, and adds or subtracts the multiplicand to or from the partial product, depending on the value of the current bit and the previous bit. The modified versions of the Booth algorithm scan the bits of the multiplier two bits at a time or three bits at a time. These techniques are equivalent in the sense that the identity 2n1 + 2n2 + ... + 2n1 + 2n2 + ... + 2n1 + 2n2 is used to collapse blocks of 1's appearing in a binary representation. In a signed-digit number with radix 2, three symbols {1, 0, 1} are allowed for the digit set, in which 1 and 1 in bit position i represent i+2 and -i, respectively.

The recoding is called canonical if it contains no adjacent nonzero digits. The canonical signed-digit vector can be constructed by the algorithm of Reitwiesner [5]. The Reitwiesner algorithm computes the recoded number starting from the least significant digit and proceeding to the left. First the auxiliary carry variable Ci is set to 0 and subsequently the binary number A is scanned two bits at a time. The canonically recoded digit Bi and the next value of the auxiliary binary variable Ci+1, for i = 0, 1, 2, ...

As an example, when A = 3038, we compute the canonical signed-digit vector B as A = (10101111011100) = (01100011000100) = B

Note that in this example the number A contains nine nonzero bits, and its canonically recoded version contains only four nonzero digits. It has been shown [1, 2] that the average Hamming weight of a k-bit canonically recoded binary number approaches k/3 when k → ∞.

Multiplication algorithm: Let X be a k-bit binary number, and Y1, Y2, ..., Yn be k-bit canonically recoded numbers. We will assume that k is sufficiently large so that the average Hamming weight of each Yi is approximately equal to k/3. Common-multiplicand multiplication using the standard method requires n multiplications, each of which requires k/3 additions on the average. Thus, a total of nk/3 additions will be required. To apply the technique of [7], we first define the following operators over the set {0, 1, -1} as shown in Table 2.

Table 2: Operators and

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

These operators are commutative and associative, which can be proven by checking all combinations. With these definitions, the multiplication of canonically recoded numbers is quite simple. First we compute Y1 = Y1 × Y1 × ... × Y1, using the new definition for a, and compute Yi = Yi ⊕ Yi, for all i ≤ n, using the new definition for a. As in the case of binary numbers, Yi = Yi + Yi. Thus, we can compute Pi = Pi ⊕ Pi = X × (Yi + Yi). We compute Pn for all 1 ≤ i ≤ n by breaking the set Y1, Y2, ..., Yn into [n/2] subsets with t elements each, and one subset with n mod t elements.

We assume that the two possible nonzero digits, 1 and -1, occur with equal probability. Furthermore, because Pr(0) = 2/3, we have Pr(1) = Pr(-1) = 1/6. Now, note the behaviour of the new operator, defined above. Given Q1, Q2, ..., Qn, with Qi ∈ {0, 1, -1} for all 1 ≤ i ≤ t, we compute Qi = Qi ⊕ Qi, Qi = Qi + Qi, where Qi will be equal to 1 if and only if Qi = 1 for all 1 ≤ i ≤ t. Similarly, Qi will be equal to -1 if and only if Qi = -1 for all 1 ≤ i ≤ t. In all other cases, Qi will be equal to zero. As a result, Pr(Qi = 1) = (Pr(1))/3 = 6/t and Pr(Qi = -1) = (Pr(-1))/3 = 6/t. Thus, the average Hamming weight of Yi is equal to 2 × 6/t, and the average Hamming weight for each of the Yi on average is equal to k/3.

Thus, the total number of additions needed to perform common-multiplicand multiplication on t numbers is found as

\[ 2 \times \frac{6}{t} \times t + (1 - \frac{6}{t+1}) \times (k/3) \times t \]

Ignoring the additions required to compute Pi = Pi ⊕ Pn, we compute the performance improvement over the standard algorithm as

\[ \frac{2 \times \frac{6}{t} + (1 - \frac{6}{t+1}) \times (k/3) \times t}{t} \]

By inserting appropriate values of t, we can determine the increase in performance for common-multiplicand multiplication of canonically recoded numbers. As was the case for binary numbers, it is...
Scheduling parallel tasks on hypercubes

J.-F. Lin and S.-J. Chen

The authors consider the problem of non-pre-emptively scheduling independent parallel tasks with communication overhead on a d-dimensional hypercube system. To find a schedule such that the schedule length is minimized is NP-hard. Therefore, a simple heuristic algorithm is investigated and its performance bound is derived as \((2 + \ln m - 1/m)\), where \(m = 2^d\).

Introduction: In the conventional scheduling problem, it is assumed that each task is processed in only one processor at a time. However, in the parallel task scheduling problem \([1, 2, 4, 5]\), each task is worked on by more than one processor at a time. Assume we have a set of \(n\) independent parallel tasks \(T = \{T_1, T_2, ..., T_n\}\) to be processed in a \(d\)-dimensional hypercube, and each task \(T_i\) has a computation requirement \(t_i\). Then assume that each task \(T_i\) is associated with a minimum parallelism dimension \(d_i\), that is, each task \(T_i\) can be processed at most on a \(d_i\)-dimensional subspace and this parallelism dimension, once decided for \(T_i\), will not be altered during its processing. If a task \(T_i\) is scheduled to run on a \(d_i\)-dimensional subspace, \(1 \leq d_i \leq d\), then \(d_i\) will be called the scheduled dimension of \(T_i\) and the execution time required by \(T_i\) will be \((t_i/2^{d_i})\). For this problem type, a schedule is feasible if the scheduled dimension \(d_i\) of each task \(T_i\) is no greater than its maximum parallelism dimension \(d\). A feasible schedule is called an optimal schedule if it has the shortest schedule length. A heuristic algorithm has a performance bound of \(O(\ln m/S_d)\) for all problem instances, where \(S_d\) and \(S_0\) denote the heuristic schedule length and the optimal schedule length, respectively.

Modified largest dimension first (MLDF) scheduling algorithm: Using the scheduled dimension \(d_i\) determined for each task \(T_i\) by the above decision-time-decision procedure, a given set of \(n\) independent parallel tasks \(T = \{T_1, T_2, ..., T_n\}\) can be decomposed into \((p + 1)\) subsets of tasks: \(T_p = \{T_{p,1}, T_{p,2}, ..., T_{p,n_p}\}\), \(P = \{T_1, T_2, ..., T_{p-1}, T_{p+1}, ..., T_n\}\), and \(T_p = \{T_{p,1}, T_{p,2}, ..., T_{p,n_p}\}\) such that each task \(T_i\) in \(T_p\) has a scheduled dimension of \(j\) for \(j = 0, 1, 2, ..., p\), and \(i = 1, 2, ..., n_p\) where \(p = \text{max}(d_1, d_2, ..., d_n, d_1)\), \(T_p = T_{p,1} \cup T_{p,2} \cup ... \cup T_{p,n_p}\), and \(n = n_1 + n_2 + ... + n_1 + n_n\).

MLDF algorithm: [Input task set \(T\); call decision-decision procedure to find \(d_i\) of each task \(T_i\); Divide task set \(T\) into \((p + 1)\) task subsets \(T_0, T_1, ..., T_p\); For each \(p\), choose the \(p\) processors with \(p = \text{max}(d_1, d_2, ..., d_n)\); Assign tasks to subtasks from task subset \(T_p\);]

Lemma 1: The number of free processors in the hypercube system will always be a multiple of the number of processors required by a task which is the next to be scheduled in the MLDF algorithm.

Lemma 2: If task \(T_j\), \(q < s < r\), is finished at time \(S_q\), there will be no processor idle before \((S_r - S_q)\), where \(S_q\) and \(S_r\) denote the schedule length of task set \(T\) scheduled by the MLDF algorithm and the processing time of task \(T_j\), respectively.

Theorem 1: The performance bound of the MLDF algorithm is bounded by \((2 + \ln m - 1/m)\), and this bound is almost tight, where \(m = 2^d\).