6 YUNIK, M., and BOYANOV, B.: 'Method for evaluation of the noise-to-harmonic component ratios in pathological and normal voices', *Acustica*, 1990, **70**, pp. 89–91

7 KOHONEN, T.: 'The self-organizing map', *Proc. IEEE*, 1990, **78**, pp. 1464–1480

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 $0.306nk$ additions for canonically recoded signed-digit numbers, whereas the binary case requires $0.375nk$ additions.

Multiplication of binary numbers: Let X and $Y_1, Y_2, \ldots, Y_n$ be k-bit 2's complement or unsigned binary numbers such that $n \geq 2$. We want to compute the numbers $P_1, P_2, \ldots, P_n$ such that $P_i = X \times Y_i$ for all $1 \leq i \leq n$. 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 $P_i = X \times Y_i$ separately for each i, which takes n multiplications. Assuming that each Y_i 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 $t \leq n$. We first compute $Y_c = Y_1 \wedge Y_2 \wedge \ldots \wedge Y_t$, where $\wedge$ is the bit-wise AND operation. We then compute $Y_{i,c}$ for all $i \leq t$, such that $Y_{i,c} = Y_i \oplus Y_c$ where $\oplus$ is the bit-wise XOR operation. It can be easily seen that $Y_i = Y_{i,c} + Y_c$. Thus, $P_i = X \times Y_i = X \times (Y_{i,c} + Y_c) = (X \times Y_{i,c}) + (X \times Y_c)$. It was shown in [7] that, using this technique, only $3nk/8 = 0.375nk$ additions will be required in the average case to perform n k-bit common-multiplicand multiplications.

Signed-digit numbers: Recoding 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 $2^{i+j} - 2^i = 2^{i+j-1} + 2^{i+j-2} \ldots + 2^{i+1} + 2^i$ is used to collapse blocks of 1's appearing in a binary representation. In a signed-digit number with radix 2, three symbols $\{\bar{1}, 0, 1\}$ are allowed for the digit set, in which 1 and $\bar{1}$ in bit position i represent $+2^i$ and -2^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 C_0 is set to 0 and subsequently the binary number A is scanned two bits at a time. The canonically recoded digit B_i and the next value of the auxiliary binary variable C_{i+1} for $i = 0, 1, 2, \ldots, n$ are generated using Table 1.

Table 1: Canonical recoding

A_{i+1}	A_i	C_i	B_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	$\bar{1}$	1
1	1	0	$\bar{1}$	1
1	1	1	0	1

As an example, when $A = 3038$, we compute the canonical signed-digit vector B as

$$A = (010111011110) = (10\bar{1}0000\bar{1}000\bar{1}0) = B$$

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 $k/3$ as $k \to \infty$.

Multiplication algorithm: Let X be a k-bit binary number, and $Y_1, Y_2, \ldots, Y_n$ be k-digit canonically recoded numbers. We will assume that k is sufficiently large so that the average Hamming weight of Y_i 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 $\wedge$ and $\oplus$ operators over the set $\{0, 1, -1\}$ as shown in Table 2

Table 2: Operators $\wedge$ and $\oplus$

$\wedge$	0	1	–1
0	0	0	0
1	0	1	0
–1	0	0	–1

$\oplus$	0	1	–1
0	0	1	–1
1	1	0	0
–1	–1	0	0

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 $Y_c = Y_1 \wedge Y_2 \wedge \ldots \wedge Y_t$, using the new definition for $\wedge$, and compute $Y_{i,c} = Y_i \oplus Y_c$ for all $i \leq t$, using the new definition for $\oplus$. As in the case of binary numbers, $Y_i = Y_{i,c} + Y_c$. Thus, we can compute $P_i = P_{i,c} \times P_c = X \times (Y_{i,c} + Y_c)$. We compute P_i for all $1 \leq i \leq n$ by breaking the set $Y_1, Y_2, \ldots, Y_n$ up into $[n/t]$ 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 $\wedge$ operator, as defined above. Given $Q_1, Q_2, \ldots, Q_t$ with $Q_i \in \{0, 1, -1\}$ for all $1 \leq i \leq t$, we compute $Q_c = Q_1 \wedge Q_2 \wedge \ldots \wedge Q_t$. Q_c will be equal to 1 if and only if $Q_i = 1$ for all $1 \leq i \leq t$. Similarly, Q_c will be equal to –1 if and only if $Q_i = -1$ for all $1 \leq i \leq t$. In all other cases, Q_c will be equal to zero. As a result, $Pr(Q_c = 1) = (Pr(1))^t = 6^{-t}$ and $Pr(Q_c = -1) = (Pr(-1))^t = 6^{-t}$. Thus, the average Hamming weight of Y_c is equal to $2 \times 6^{-t} \times k$, and the average Hamming weight for each of the $Y_{i,c}$ terms is equal to

$$k/3 - 2 \times 6^{-t} \times k = (1 - 6^{-t+1}) \times (k/3)$$

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

$$2 \times 6^{-t} \times k + (1 - 6^{-t+1}) \times (k/3) \times t$$

Ignoring the additions required to compute $P_i = P_{i,c} + P_c$, we compute the performance improvement over the standard algorithm as

$$\frac{t/3}{2 \times 6^{-t} + (1 - 6^{-t+1}) \times (1/3) \times t}$$

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 easily shown that the performance improvement is maximised when $t = 2$. Larger arrays can be dealt with by breaking the array up into pairs. By substituting 2 for t into the above formula, we calculate that the performance improvement for the canonically recoded numbers is 12/11. Thus, the common-multiplicand multiplication of n k-digit canonically recoded numbers takes $nk \times (1/3) \div (12/11) = 11nk/36 \simeq 0.306nk$ additions.

14 March 1994
Electronics Letters Online No: 19940623

Ç. K. Koç and S. Johnson (*Department of Electrical & Computer Engineering, Oregon State University Corvallis, Oregon 97331, USA*)

References

1 ARNO, S., and WHEELER, F.S.: 'Signed digit representations of minimal Hamming weight', *IEEE Trans.*, 1993, **C-42**, (8), pp. 1007–1010
2 EĞECIOĞLU, Ö, and KOÇ, Ç.K.: 'Exponentiation using canonical recoding', to be published in *Theoretical Computer Science*, 1994
3 KNUTH, D.E.: 'The art of computer programming: Seminumerical algorithms. Vol. 2' (Reading, MA: Addison-Wesley, Second edition, 1981)
4 KOREN, I.: 'Computer arithmetic algorithms' (Englewood Cliffs, NJ: Prentice-Hall, 1993)
5 REITWIESNER, G.W.: 'Binary arithmetic', *Advances in Computers*, 1960, **1**, pp. 231–308
6 RIVEST, R.L., SHAMIR, A., and ADLEMAN, L.: 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, 1978, **21**, (2), pp. 120–126
7 YEN, S.-M., and LAIH, C.-S.: 'Common-multiplicand multiplication and its applications to public key cryptography', *Electron. Lett.*, 1993, **29**, (17), pp. 1583–1584

Scheduling parallel tasks on hypercubes

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

Indexing terms: Computer architecture, Parallel architectures, Scheduling

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 minimised 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 $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ 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 Δ_i, that is, each task T_i can be processed at most on a Δ_i-dimensional subcube 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 subcube, $1 \leq d_i \leq \Delta_i \leq d$, 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 Δ_i. A feasible schedule is called an optimal schedule if it has the shortest schedule length. A heuristic algorithm has a performance bound of β if $(S_H/S_O) \leq \beta$ for all problem instances, where S_H and S_O denote the heuristic schedule length and the optimal schedule length, respectively.

Under the linear speedup assumption, Wang and Cheng [4] proposed an ECT (earliest completion time) algorithm for scheduling parallel tasks and derived a performance bound of $(3 - 2/m)$ in which the number of processors required by each task and the number of processors m in systems are arbitrary. Zhu and Ahuja [5] proposed an LDF (largest dimension first) algorithm for scheduling parallel tasks on hypercube systems and derived a performance bound of $(2 - 1/m)$, where m is the number of processors in the hypercube system.

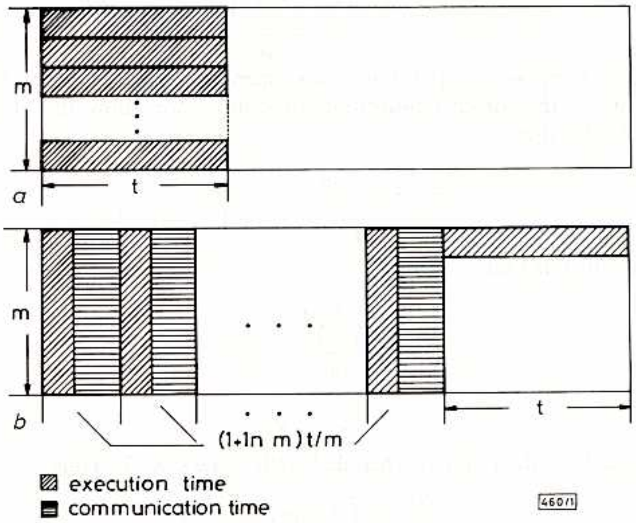

Fig. 1 *Optimal schedule and MLDF schedule*

a Optimal
b MLDF

Decision of scheduled dimension: It is obvious that while a task T is processed in parallel on a parallel machine, communication between processors is unavoidable. As a consequence, a linear speedup can never be achieved. In a hypercube system, because the average communication overhead is proportional to the dimension x of a subcube on which a task T is processed [3], the communication time required by each processor can be assumed to be Cx, where C is a given positive rational number. Thus, the total processing time $P(x,t)$ required for processing a task T is equal to $((t/2^x) + Cx)$, where t is the computational requirement of task T. By simple calculus, we then have: $\partial P(x,t)/\partial x = -(\ln 2/2^x)t + C$ and $\partial^2 P(x,t)/\partial x^2 = [(\ln 2)^2/2^x]t$. Because $\partial P(x,t)/\partial x = 0$ when $x = \log_2[(\ln 2/C)t]$ and $\partial^2 P(x,t)/\partial x^2 > 0$, $P(x,t)$ reaches its minimum value at $x = \log_2[(\ln 2/C)t]$. Based on the derived result, our strategy is to make the scheduled dimension of task T equal to $\lfloor \log_2[(\ln 2/C)t] \rfloor$.

Dimension-decision procedure: {Assign $\min\{\Delta_i, \lfloor \log_2[(\ln 2/C)t_i] \rfloor\}$ to the scheduled dimension d_i of task T_i, for $i = 1, 2, \ldots, n$.}

Modified largest dimension first (MLDF) scheduling algorithm: Using the scheduled dimension d_i determined for each task T_i by the above dimension-decision procedure, a given set of n independent parallel tasks $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ can be decomposed into $(\rho + 1)$ subsets of tasks: $T^\rho = \{T^\rho_1, T^\rho_2, \ldots, T^\rho_{n\rho}\}$, $T^{\rho-1} = \{T_1^{\rho-1}, T_1^{\rho-1}, \ldots, T_{n\rho-1}^{\rho-1}\}$, $\ldots$, $T^1 = \{T^1_1, T^1_2, \ldots, T^1_{n1}\}$ and $T^0 = \{T^0_1, T^0_2, \ldots, T^0_{n0}\}$ such that each task T^j_i in T_j has a scheduled dimension of j, for $j = 0, 1, 2, \ldots, \rho$ and $i = 1, 2, \ldots, n_j$, where $\rho = \max\{d_1, d_2, \ldots, d_n\}$, $\mathbf{T} = T^\rho \cup T^{\rho-1} \cup \ldots \cup T^1 \cup T^0$, and $n = n_\rho + n_{\rho-1} + \ldots + n_1 + n_0$.

MLDF algorithm: {Input task set $\mathbf{T}$; call dimension-decision procedure to find d_i of each task T_i; Divide task set $\mathbf{T}$ into $(\rho + 1)$ task subsets $T^\rho, T^{\rho-1}, \ldots, T^1$ and T^0, where $\rho = \max\{d_1, d_2, \ldots, d_n\}$; Assign tasks to subcubes from task subset T^ρ to T^0;}

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^q_w, $q \leq d$, $1 \leq w \leq n_q$, is finished at time S_H, then there will be no processor idle before $(S_H - S^q_w)$, where S_H and S^q_w denote the schedule length of task set $\mathbf{T}$ scheduled by the MLDF algorithm and the processing time of task T^q_w, 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 $m = 2^d$.