MATHEMATICS OF COMPUTATION
VOLUME 44, NUMBER 169
JANUARY 1985, PAGES 251-260

Implementing the Continued Fraction Factoring Algorithm on Parallel Machines

By Marvin C. Wunderlich*

Abstract. An implementation is described of the continued fraction factoring algorithm on the DAP parallel processor located in Queen Mary College in London. The DAP has 4096 parallel processors each containing 16K bits of memory and the suggested implementation incorporates the early abort strategy and the large prime variation.

Introduction. There are several good general factoring methods (i.e., methods which do not require factors of a certain form) for finding the factors of large composite numbers. The one which has been implemented and analyzed the most is the continued fraction algorithm of M. Morrison and J. Brillhart [4] which in 1970 factored $F_7 = 2^{128} + 1$ on the IBM 360/91 at U.C.L.A. An important variation on the continued fraction method (henceforth called CFRAC) is the early abort strategy (EAS) which has been mentioned by several factorizers including Brillhart but was finally analyzed and implemented by Pomerance and Wagstaff [7]. An essentially new method called the quadratic sieve (QS) was popularized by Pomerance and implemented by J. L. Gerver [1] at Rutgers University. QS should be superior to CFRAC even with EAS for numbers in excess of 60 decimal digits and appears to be ideally suited to fast pipe line machines like the CRAY I. All of these algorithms along with several other important variations have been analyzed from an asymptotic point of view by C. Pomerance [6].

It is the author's belief that CFRAC with or without EAS is ideally suited for implementation on highly parallel array processors such as the ICL DAP in London or the Goodyear MPP, which has been installed this year at Goddard Space Flight Center. This paper describes a suggested implementation of CFRAC on such a machine and presents a running time analysis which optimizes the input parameters for numbers which are in the 60 decimal digit range. Array processors are not readily available at this time. The ICL DAP is located in London and is only available to the author through an international telecommunications link such as TELENET. The MPP has been recently installed at NASA and is not easily available. Thus the implementation described herein has not been completely written and the assumptions used in the analysis have yet to be tested on large numbers.

Received March 2, 1983; revised November 28, 1983 and May 16, 1984.
1980 *Mathematics Subject Classification*. Primary 10A25.
*Research supported in part by grants from NSF and NSA.

0025-5718/85 $1.00 + $.25 per page

I. Description of the Algorithm. All the methods discussed in the introduction are based on the observation that if two squares X^2, Y^2, can be found such that $X^2 \equiv Y^2 \pmod{M}$ and $X \equiv \pm Y \pmod{M}$, then M can be factored by computing $\mathrm{GCD}(X - Y, M)$, since $M \mid X^2 - Y^2 = (X - Y)(X + Y)$. All the methods find the two squares by generating and factoring a large collection of quadratic residues, mod M. They differ only in the method of computing the quadratic residues and the procedures used in factoring them. Good descriptions of CFRAC can be found in Morrison and Brillhart [4] and in Knuth [2]. We will repeat the salient portions of the algorithm for this discussion.

Step 1. If M is the number to be factored, compute all the primes $p \leqslant x$ for which $(M/p) = 1$. x is an input parameter whose value will be discussed later and (M/p) is the usual Jacobi symbol. Assume there are F such primes and refer to them as the *factor base* and denote the set by FB.

Step 2. Compute a set of integers $Q = (Q_1, Q_2, \ldots, Q_{\mathrm{NQ}})$ and $A = (A_1, A_2, \ldots, A_{\mathrm{NQ}})$ having the properties

$$(-1)^i Q_i \equiv A_i^2 \bmod M, \tag{1}$$

$$Q_i \leqslant 2\sqrt{M}. \tag{2}$$

The Q's and A's are generated by making use of the continued fraction expansion of $\sqrt{M}$. (See Remark 2.) The parameter NQ depends on the size of M and is discussed later.

Step 3. Attempt to factor each Q_i in Q by dividing it by each p in the factor base. Either Q_i will factor completely over FB so that

$$Q_i = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_F^{\alpha_F} \tag{3}$$

or Q_j will factor having one large prime p in the factorization with $p < (p_F)^2$. Therefore,

$$Q_i = p_1^{\beta_1} p_2^{\beta_2} \cdots p_F^{\beta_F} \cdot p. \tag{4}$$

Step 4. Construct a 0, 1 matrix $T = \{m_{i,j}\}$ having F columns and R rows using the factorizations obtained in Step 3. T will have one column for each prime in the factor base. R, the number of rows, can be partitioned as $R = t_1 + t_2$, where t_1 is the number of Q's which factor as in (4) for which their largest prime p are equal. If a largest p occurs n times, it counts as $n - 1$ pairs. In the first case, the matrix row consists of the 0, 1 vector $(a_1, a_2, \ldots, a_F)$, where $a_i \equiv \alpha_i \pmod 2$ in (3). In the second case, suppose the pair is Q_j as in (4) and

$$Q_1 = p_1^{\gamma_1} p_2^{\gamma_2} \cdots p_F^{\gamma_F} p.$$

We then let the matrix row be the vector $(c_1, c_2, \ldots, c_F)$, where $c_i \equiv \alpha_i + \gamma_i \pmod 2$. This is the 0, 1 vector corresponding to the quadratic residue $Q_i \cdot Q_1$.

If $R = F + D$, then applying Gaussian elimination to T will produce at least D zero rows and each represents a collection of Q's whose product contains only even powers in its unique factorization and is a square. If X^2 is one of these squares, and Y^2 is the product of the A^2 in (1) corresponding to these Q_i's, then $X^2 - Y^2 \equiv 0 \pmod M$ and $\mathrm{GCD}(X - Y, M)$ will likely be a proper factor of M. If none of the squares succeed in factoring M, one goes back to Step 2 to compute and factor some more values of Q.

Remarks. 1. The computation of the Jacobi symbols in Step 1 consumes little time in the algorithm and could be computed sequentially on a standard computer without severely impacting on the total execution time. However, the implementation described herein uses a parallel algorithm to compute 4096 Jacobi symbols simultaneously. This is described in [7].

2. The Q's are generated recursively using the formulas

$$\begin{aligned} Q_n &= Q_{n-2} + q_{n-1}(r_{n-1} - r_{n-2}), \\ G_n &= 2g - r_{n-1}, \\ q_n &= [G_n/Q_n], \\ r_n &= G_n - q_n q_n, \\ A_n &\equiv q_n A_{n-1} + A_{n-2} \bmod N, \end{aligned}$$

where the initial values are

$$G = [\sqrt{N}], \quad Q_{-1} = N, \quad Q_0 = 1, \quad q_0 = g, \quad r_0 = 0, \quad A_{-1} = 1, \quad A_0 = g.$$

Although parallel methods are known for generating values of Q and even possibly the A's, in all current implementations of this algorithm, these are computed sequentially.

3. Step 4 requires that the factorization satisfying (4) be sorted on the largest prime factor p so that those with equal p can be identified. Steps 2 and 3 must be executed until it is reasonably certain that enough factored Q are generated so that Step 4 will produce a few dependent rows. This is generally accomplished by computing the ratio LEVEL = (F + LP)/NF where F is the number of primes in the factor base, LP is the number of Q which factored with a large prime as in (4) and NF is the total number of factored Q. Experience has shown that when LEVEL reaches the value .96 for numbers around 40–50 digits in length, there are a suitable number of dependent rows in Step 4 to obtain a factorization. For larger numbers (60–70 digits) it appears that a value of LEVEL = .98 is more suitable. More will be said later in this paper about this strategy.

4. An algorithm which performs Gaussian elimination on a 0, 1 matrix in GF(2) using minimal storage is discussed in a separate paper by Parkinson and Wunderlich [5]. This step uses little computer time compared to the factoring in Step 3 and is not included in the subsequent analysis.

II. An Analysis. We will first examine a simplified version of CFRAC in which the large prime variation is not employed. This version only accepts complete factorizations of the Q over the factor base as in (3). We must determine the optimum value of $x = p_F$, the largest prime in the factor base, to minimize the running time. We will assume that sufficient Q must be factored so that NF, the number of factored Q, will equal F, the number of elements in the factor base. To do this, we must attempt to factor

$$\text{NQ} = F/r \tag{5}$$

values of Q, where r is the fraction of Q's which factor over the factor base and this will require ND division operations where

$$\text{ND} = F^2/r. \tag{6}$$

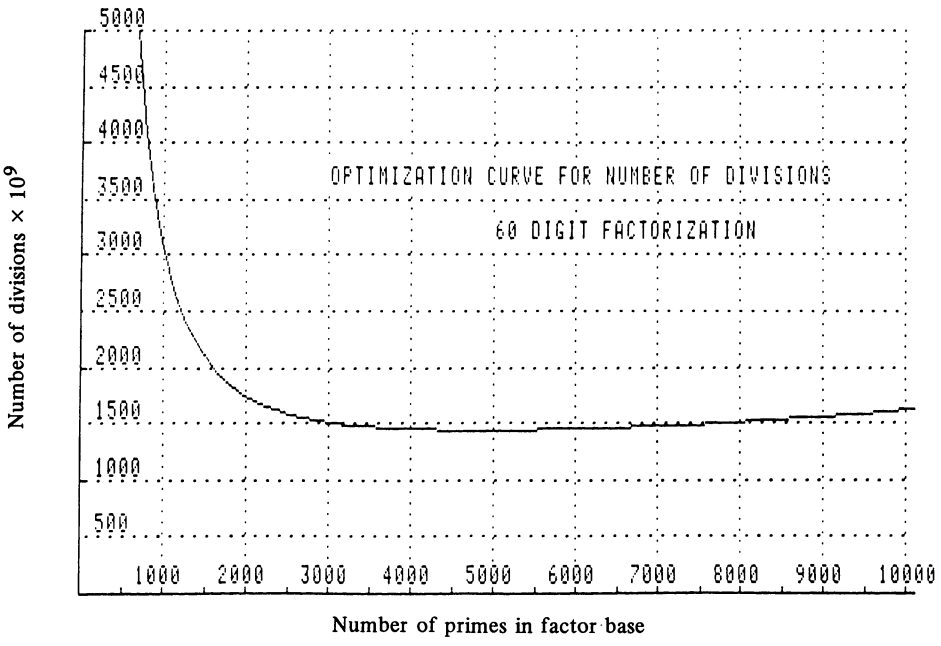

FIGURE 1

The ratio r can be approximated by Dickman's function $r(\alpha)$ which is the limiting fraction of integers n for which all prime factors of n are less than n^{α}. To this end, we let $\alpha = (\log x)/\log\sqrt{M}$ and using the prime number theorem and the fact that roughly half the prime numbers p have the property that the Legendre symbol $(M/p) = 1$, we obtain

$$\mathrm{F} = \frac{M^{\alpha/2}}{\alpha \log M}$$

which gives a running time estimate

$$\mathrm{ND} = \frac{M^{\alpha}}{r(\alpha)\alpha^2 \log^2 M}. \tag{7}$$

Dickman's function $r(\alpha)$ has been tabulated by D. E. Knuth and L. Trabb Pardo [3, p. 340]. Interpolating geometrically in these tables, we can choose the optimum value of α, and hence of F, which minimizes ND.

The effect of using the large prime variation can be analyzed in a similar fashion. If we let L be the value LEVEL = .98 discussed in Remark 3, let r be defined as in (6) and let g be the fraction of Q's which factor admitting a large prime, then a little calculation shows that the large prime variation affects the estimate of ND given in (6) by the factor

$$\mathrm{LPC} = \frac{r}{g}\,\frac{L}{1 - L(1 - (r/g))} \tag{8}$$

which reduces to unity when $L = 1$. This value can be computed using the tabulated values of $g = G(\alpha)$ given in Knuth and Trabb Pardo; for $N = 10^{60}$, $L = .98$ and for values of α in the "useful range" around .15, the value of LPF is about .413. For this

TABLE 1

F	α	NQ	.4ND	r(α)
20,202	0.1800	0.294167E+09	0.229628E+13	0.000071091
10,720	0.1700	0.388178E+09	0.166458E+13	0.000027617
5,709	0.1600	0.632527E+09	0.144438E+13	0.000009025
5,361	0.1590	0.671318E+09	0.143964E+13	0.000007986
5,035	0.1580	0.713623E+09	0.143726E+13	0.000007056
5,004	0.1579	0.718061E+09	0.143715E+13	0.000006968
4,972	0.1578	0.722538E+09	0.143707E+13	0.000006882
4,941	0.1577	0.727054E+09	0.143700E+13	0.000006796
4,910	0.1576	0.731612E+09	0.143697E+13	0.000006712
4,880	0.1575	0.736209E+09	0.143695E+13	0.000006628
4,849	0.1574	0.740849E+09	0.143697E+13	0.000006545
4,819	0.1573	0.745528E+09	0.143700E+13	0.000006464
4,789	0.1572	0.750250E+09	0.143706E+13	0.000006383
4,759	0.1571	0.755015E+09	0.143715E+13	0.000006303
4,729	0.1570	0.759821E+09	0.143726E+13	0.000006224
4,442	0.1560	0.810345E+09	0.143969E+13	0.000005481
4,172	0.1550	0.865683E+09	0.144461E+13	0.000004819
3,052	0.1500	0.123668E+10	0.150970E+13	0.000002468
1,639	0.1400	0.300163E+10	0.196767E+13	0.000000546
885	0.1300	0.992030E+10	0.350998E+13	0.000000089

reason, and in view of the shaky evidence for determining the correct value of $L = .98$, we will simply introduce the factor .4 in all of our tabulation of ND in order to allow for the use of the large prime variation. It is clear that as our ability emerges to factor numbers significantly larger than 60 digits, the usefulness of this variation will certainly diminish. This is consistent with Pomerance's asymptotic analysis of the variation given in [6]. For a more thorough analysis of this variation, see [9].

Figure 1 and Table 1 demonstrate the result of this optimization. For various values of α shown in column 2 of the table, the values of F, NQ, NDx.4 and the interpolated value $r(\alpha)$ obtained from Knuth's tables are tabulated. It shows that CFRAC should optimally perform on a 60-digit factorization if about 4880 primes are used in the factor base and 7.36×10^8 values of Q are computed. About 1.44×10^{12} division instructions will be required which, at a rate of one micro-second per division on a sequential machine, would take about 400 hours of computing.

III. The Implementation. The DAP array processor consists of 4096 individual processors each having a memory consisting of 16,384 bits. Data can be organized in almost any way in the processor memories since the lowest level assembly language treats each single bit as a separate addressable storage location. Instruction acts on all processors simultaneously, although any subset of the processors can be masked out so that data is not affected in the masked processors. We will describe a suggested implementation for Step 3 of the algorithm: the factoring of the Q_i's. One can either put 4096 different Q's in the processors and divide them individually by the p's in the factor base, or one can put one prime in each processor and, with a single instruction, divide a single Q by all the primes in the factor base. This implementation uses a combination of the two methods. We will use the latter

method in Stage 1 of the algorithm in order to identify, for each Q, the p's which divide Q. Then in Stage 2, we use the first-mentioned method to actually factor the Q's.

Set up. Put a different prime p from the factor base in each processor so that in the algorithm below, P will always refer to a collection of up to 4096 primes, one in each processor memory.

Stage 1. In *one processor*, compute the next (Q, A) pair using the equations in Remark 2. In each processor, compute REM $= (Q/P) * P$ where truncated integer division is meant by the / sign. In each processor for which REM $= 0$, store P on a stack contained in the processor containing the current Q and A. Repeat Stage 1 using another processor for the (Q, A) pair until all processors are used.

Remark. Each processor memory now contains a value of Q, A and a set of primes from the factor base which divide that particular value of Q. The average number of P in each stack is about log log Q or 4.24 for a 60-digit factorization. The maximum number for 4096 Q's should not exceed ten.

Stage 2. In all processors, beginning with $I = 1$, divide Q by the Ith element of the prime stack giving QUOT and REM so that

$$Q = P(I) * \text{QUOT} + \text{REM}.$$

Where REM $= 0$, replace Q by REM and repeat until all REM $\neq 0$. Replace I by $I + 1$ and repeat the stage for those processors whose prime stack contains an Ith prime.

Test. Store on a file those Q's which have factored. If not enough Q's have been interrogated, go back to Stage 1.

Note that a great deal of inefficiency is introduced in Stage 2, since for most divisions of higher powers, most processors will have been masked out. But this entire stage processes at most 10 primes and uses up a small amount of processor time compared to Stage 1 which represents the bottleneck in this implementation. Since the fundamental operation in Stage 1 is to find a remainder rather than determine a quotient and remainder, we can save processor time by programming the "division" in Stage 1 by using repeated additions. For example, if N is a 60-digit number and each Q contains about 100 bits, then in the *Setup* portion of the algorithm, compute in each processor memory, the numbers $T(1), T(2), \ldots, T(100)$ where $T(i) \equiv 2^i \bmod P$ for the prime P assigned to that processor. Now, let $B(1), B(2), \ldots, B(100)$ represent the contents of bit 1, bit 2, ..., bit 100 of the current value of Q in Stage 1. Then

$$\text{REM} \equiv \sum_{\substack{i=1 \\ B(i)=1}}^{100} T(i) \bmod P,$$

and finding a remainder mod P in each processor amounts to k additions, where k is the number of binary locations in Q containing 1; generally about 100/2. Such a programming trick is easy to implement on an array processor which has general bit manipulation instructions. This procedure for finding a remainder is about $3\frac{1}{2}$ times faster than ordinary division.

Most of the individual parts of this implementation have been written on the DAP but, at the time of this writing, the total program has not been written. It is expected that a 60-digit factorization should take about 5 hours of DAP time.

IV. The Early Abort Strategy. The Early Abort Strategy (EAS) consists of giving up on the trial division of a particular Q_n if division by the early primes in the factor base produces relatively few prime factors of Q_n. This strategy has been analyzed by Pomerance [6] and an implementation on a sequential computer has been described by Pomerance and Wagstaff [7]. The results of their investigations show that this variation can effect an 8- to 10-fold acceleration of CFRAC. We will describe their method and show how it could be incorporated in the DAP implementation.

To implement EAS with three aborts, we would choose three integers m_1, m_2, and m_3 which satisfy

$$0 < m_1 < m_2 < m_3 < F,$$

and three bounds B_i satisfying

$$2\sqrt{N} > B_1 > B_2 > B_3 > 1.$$

If, after dividing out of Q_i all primes p in the factor base satisfying $0 < p \leqslant m_i$, the cofactor CF_i satisfies

$$CF_i > B_i,$$

then we abandon the Q_i, compute, and begin processing the next value Q_{i+1}. There are three tests or "cuts" where such an abort is contemplated and each test reduces substantially the number of Q's which are interrogated. This is an idea well suited to a sequential processor where the values of Q are investigated one at a time and it does not seem immediately adaptable to our *Stage* 1, in which a single Q is divided by all F primes at once. We will describe an EAS scheme which will perform efficiently on a parallel processor as long as the integers m_1, $m_2 - m_1$, and $m_3 - m_2$ are divisors of the number of processors. For a 60-digit factorization on the DAP which has 4096 processors, will select the cut points

$$m_1 = 16, \quad m_2 = 80, \quad m_3 = 592.$$

We can use the table in [7, p. 108] to determine the best values for the B's and for this implementation we would use

$$B_1 = 4 \times 10^{27}, \quad B_2 = 7 \times 10^{24}, \quad B_3 = 10^{21}.$$

Stage 1 would now be expanded into three nested stages in the following way.

Pass 1. *Attempt to Factor the Q's With* 16 *Primes.*

Stage 1.1. We partition the 4096 processors into 256 partitions consisting of 16 processors each. We regard these as separate parallel processors each consisting of 16 processors and Stage 1 is implemented with multiplicity 256 in that 256 values of Q are computed, one in each partition, and the first 16 primes in the factor base are placed in each partition. With one parallel instruction, the value REM = $(Q/P) * P$ is computed in each processor of each partition. Within each partition, the primes corresponding to REM = 0 are placed in a stack in the processor containing the particular Q belonging to the partition. The processor location of this value of Q is incremented by one with each execution of this stage until, after 16 executions, 4096 values of Q have been tested, and each is associated with its stack of primes dividing Q.

Stage 1.2. This executes exactly as Stage 2 of the previously described method. The elements of the stack are used as trial divisors in order to divide out all possible

powers of the primes which divide the Q. Then we save those values of Q for which $Q < B_1$ which will be used as input into the next pass. If there is sufficient memory, the values of Q which pass the test could be "parked" in a higher portion of memory to be recalled when a sufficient number of "good" Q's have been collected to execute a stage of the next pass.

Pass 2. *Divide by the Next* 64 *Primes.*

Stage 2.1. We now partition the 4096 processors into 64 partitions of 64 processors each and proceed exactly as in Stage 1.1. This is a natural mode of operation for the DAP since high-level software exists for the machine when it is regarded as a 64 processor parallel computer operating with a word length of 64 bits. At the conclusion of 64 executions of this stage, 4096 Q's are determined with their stack of primes carried in from Pass 1 with an additional stack of primes which were added in Stage 2.1.

Stage 2.2. The new primes are divided out of the Q's in parallel and the resulting Q's which satisfy $Q < B_2$ are again saved on a file or "parked" in an even higher portion of memory.

Pass 3. *Divide by the Next* 512 *Primes.* Here the 4096 processors are partitioned into 8 partitions consisting of 512 processors each, and the 4096 good-good Q's which have survived Passes 1 and 2 and are parked in memory or stored on a file and are retrieved eight at a time and processed exactly like Pass 2. The survivors of this pass, those for which $Q < B_3$, are parked or stored to be used as input for the final pass.

Pass 4. *The Final Set of Primes.* We now finish off those values of Q which have survived all the earlier cuts with one instruction in parallel which divides the Q's one at a time by up to 4096 primes, one in each processor. The algorithm concludes exactly like the original algorithm without EAS. The primes are parked one by one into a region of high core along with the stack of primes which divide them. Stage 4.2 divides all the prime factors and those Q which factor completely are saved for the Gaussian elimination step. If the *Large Prime* variation is employed, the Q's whose cofactor is less than the square of p_F are saved on some external storage device until enough are accumulated so that the scan program will generate a row of zeros in the Gaussian elimination step.

Pomerance and Wagstaff claim that EAS with three aborts should increase the running speed of the program by a factor of 10 and this would produce an algorithm which would factor a 60-digit number in about a half hour. However, for three of the four "Stage 1" executions, it will not be possible to obtain full advantage of the remaindering operation described in Section III in which the sum (9) is taken over all i whose ith bit of Q contains 1. Since the computer contains many Q's, the sum will have to be computed for each i and utilized in those processors whose Q has its ith bit 1. This will probably reduce our gain from using this table remaindering from $3\frac{1}{2}$ to 2, giving us a 60-digit average running time of about an hour. This section concludes with a set of remarks pertaining to a suggested DAP implementation of this algorithm.

1. The majority of this algorithm can be written in the higher-level language DAP FORTRAN. For an elementary discussion of DAP FORTRAN and an example of its application to a problem in number theory, see [8]. The most difficult part of the

algorithm will be the collection in a stack of the primes which divided the Q in its particular processor partition. (See Stage 1.1.) This essentially consists of identifying all elements of a portion of a bit plane containing a one and recording the cardinal number of its processor. This would involve programming at the microcode level. Routines do exist when the partition size is 64 as in Pass 2 or 4096 as in Pass 4.

2. The optimum size of the factor base will really depend on hardware constraints rather than theoretical considerations. Even though the choices for m_i given would permit a factor base size of 4688 if 4096 primes were used in the last pass, a total factor base size of 4096 is really preferable on the DAP. This is because a bit matrix of 4096×4096 will conveniently fit in the memory of the DAP and the elimination can be carried out in core without any disk swapping. Parkinson and Wunderlich [5] have shown how this can be done without the need for allocating additional memory for a history matrix. If one really has an unbounded number of processors, then one could compute the optimum number of processors and hence the number of primes in the factor base by minimizing the value of $\mathrm{NQ} = F/r$ given in (5). One then has to take into consideration the time taken to perform the elimination step and this makes the result hardware dependent. A preliminary computation shows the optimum number of processors to be around 50,000 and there is little hope that a computer will be built soon which has sufficient memory per processor to implement this kind of program.

3. For a 60-digit factorization, there seems to be enough memory in the DAP located at Queen Mary College, London, to implement the entire four pass algorithm without ever resorting to any external store. Pass 1 would be repeated until 4096 Q's are generated which survive Test 1. Then one iteration of Pass 2 would park a small number of Q which survived two tests. This would continue until 4096 of these Q's are parked, which would trigger one execution of Pass 3. Pass 3 is repeated until 4096 Q's are generated having survived all three tests, and then the final factorization identifies the small fraction of Q's which factor completely over the factor base or factor with a large prime. Only in the latter case will external store be needed to store the large prime factorizations. If the large prime variation is not used, the entire algorithm could be processed in one contiguous run.

4. The computation of the (Q, A) pairs is basically a serial operation although methods are known for generating the values of Q in parallel. Since the host computer for the DAP is an ICL 2980, and the DAP is always called from an executing 2980 program as a subroutine, the (Q, A) pairs could be generated in a 2980 program and stored in memory which is shared between the DAP and the 2980. The parallel factoring program in the DAP could be called as a subroutine from the 2980 whenever a new buffer of (Q, A) pairs have been generated. The 2980 operates in a time sharing mode whereas only one DAP program is executed at a time. The program would blend well with the usual mix of user programs in the system and it would not be necessary to dedicate the entire system to such a program.

5. There seems to be less advantage to using the Stage 1/Stage 2 variation when the number of primes is small as in Pass 1 than when the number of primes is large. Therefore, it may be sensible to use EAS without the two stage variation in Pass 1. Then we would merely put 4096 values of Q in the memory, one in each processor,

and proceed to divide all the Q's with one p at a time until all p's are divided out of all Q's. After doing this for 16 p's, we test $Q < B$, and proceed with Pass 2 using the two stage variation. We lose the factor of 2 because we cannot use the fast simple division method, but we may gain that back in a more efficient implementation of EAS. Only experience will tell.

Conclusion. At the time of this writing, there are some factorization programs being tested which achieve the performance suggested in this article. The most recent was done at Sandia Laboratories by J. A. Davis and D. B. Holdridge, where the quadratic sieve algorithm was programmed on a CRAY I. Davis and Holdridge have recently announced a successful factorization of a 63-digit number in a total of 5.4 hours of machine time. The author is designing an implementation of CFRAC on the MPP parallel processor which has four times as many processors and runs about twice as fast as the DAP, and such a program could conceivably factor a 60-digit number in a running time measured in minutes rather than hours.

The author would like to thank Professor Dennis Parkinson of Queen Mary College, London, who contributed many important ideas for the DAP implementation and the referee for several helpful suggestions in the preparation of the paper.

Department of Mathematical Sciences
Northern Illinois University
DeKalb, Illinois 60115

1. J. L. GERVER, "Factoring large numbers with a quadratic sieve," *Math. Comp.*, v. 41, 1983, pp. 287–294.

2. D. E. KNUTH, *The Art of Computer Programming*, Vol. 2, *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Mass., 1981.

3. D. E. KNUTH & L. TRABB PARDO, "Analysis of a simple factorization algorithm," *Theoret. Comput. Sci.*, v. 3, 1976, pp. 321–348.

4. M. A. MORRISON & J. BRILLHART, "A method of factoring and the factorization of F_7," *Math. Comp.*, v. 29, 1975, pp. 183–205.

5. D. PARKINSON & M. C. WUNDERLICH, "A compact algorithm for Gaussian elimination over $GF(2)$ implemented on highly parallel computers," *Parallel Computing*, v. 1, 1984.

6. C. POMERANCE, "Analysis and comparison of some integer factoring algorithms," *Computational Methods in Number Theory*, MC tract 154 (H. W. Lenstra and R. Tijdeman, eds.), pp. 89–139.

7. C. POMERANCE & S. S. WAGSTAFF, JR., "Implementation of the continued fraction factoring algorithm," *Congr. Numer.*, v. 37, 1983, pp. 99–118.

8. M. C. WUNDERLICH, "DAP FORTRAN. A versatile language for parallel computing," *Congr. Numer.*, v. 38, 1983, pp. 261–270.

9. M. C. WUNDERLICH, "Factoring numbers on the massively parallel computer," *Advances in Cryptology* (David Chaum, ed.), 1984, pp. 87–102.