FACTORIZATION OF LARGE INTEGERS ON A MASSIVELY PARALLEL COMPUTER*

James A. Davis and Diane B. Holdridge
Sandia National Laboratories
Albuquerque, New Mexico USA

I. INTRODUCTION

Our interest in integer factorization at Sandia National Laboratories is motivated by cryptographic applications and in particular the security of the RSA encryption-decryption algorithm. We have implemented our version of the quadratic sieve procedure on the NCUBE computer with 1024 processors (nodes). The new code is significantly different in all important aspects from the program used to factor numbers of order 10^{70} on a single processor CRAY computer. Capabilities of parallel processing and limitation of small local memory necessitated this entirely new implementation. This effort involved several restarts as realizations of program structures that seemed appealing bogged down due to inter-processor communications. We are presently working with integers of magnitude about 10^{70} in tuning this code to the novel hardware.

II. NCUBE COMPUTER

The basic element of the NCUBE computer is a 32-bit VLSI processor of the super-minicomputer range (10^6 integer operations per second). These processors are interconnected in the configuration of an N-dimensional cube. That is, an NCUBE of order k has 2^k nodes, $k = 0, 1, 2 \dots$ and one of order $k + 1$ is formed by connecting two cubes of order k at corresponding nodes. There is no common memory shared among the processors; each has one-half megabyte of local memory. Each node operates on its own stored program and data. They achieve cooperation by passing messages to one another. A very slow host board controls input-output and subcube allocation.

*This work was performed at Sandia National Laboratories and supported by the U.S. Department of Energy under contract number DE-AC04-76DP00789.

Figure 1 NCUBES of Small Order

III. THE QUADRATIC SIEVE ALGORITHM

The quadratic sieve factorization procedure is one of several methods of decomposing positive integers based on the difference-of-squares identity. If N is a composite integer and I, J are integers such that $I^2 \equiv J^2 \pmod{N}$ with $I \not\equiv \pm J \pmod{N}$, then $\text{GCD}(I + J, N)$ is a non-trivial factor of N . The difference between the various algorithms is in the means by which the quadratic congruence is generated.

In the quadratic sieve, as originally proposed by Dr. Carl Pomerance of the University of Georgia and implemented at Sandia National Laboratories on a CRAY 1S computer, many relatively small quadratic residues for N are generated by the polynomial $X^2 - N$ (X near $\sqrt{N}$). One attempts to factor a sufficient number of these residues into a set of powers of "small" primes, B , called the prime base. Gaussian elimination is then employed to determine a binary dependency; that is, a set S of factored residues such that

$$\prod_{X_i \in S} X_i^2 = \prod_{P_j \in B} P_j^{2\alpha_j} \pmod{N}.$$

If $\prod_{X_i \in S} X_i \neq \pm \prod_{P_j \in B} P_j^{\alpha_j}$ we have factorization; otherwise another quadratic congruence is formed and we try again.

The procedure is called a sieve because of its similarity to the prime number sieve (sieve of Eratosthenes). That is, if a prime p divides $X_p^2 - N$ then it divides residues at the entire arithmetic progression of arguments $X_p + kp$, $k = 0, \pm 1, \pm 2 \dots$. So once a residue divisible by p is identified, the prime may be divided out of the functional values defined by arguments in this progression. This operation is done very efficiently, particularly on a vector computer such as the CRAY. When one is dealing with large integers the frequency with which the residues factor completely is small, so we merely identify these successes by operating on the residues with single-precision logarithms, rather than multiple-precision division. After the array of residues has been sieved with each member of the prime base, B , the remainders are compared with a threshold value which indicates factorization. When enough of the factorable residues are identified (approximately the number of distinct primes in the base, B) the sieving portion of the algorithm is terminated.

The sieving and searching described above constitutes the lion's share of the computation. After the set of residues that factor is identified, the actual functional values are calculated in multiple-precision and decomposed into the primes by division. The final step is to determine a binary dependency by Gaussian elimination.

IV. MODIFICATIONS TO BASIC ALGORITHM

As one increases the size of the integers to be factored the size of the prime base must grow in order to have significant probability of factoring residues. Thus a larger number of factored residues is needed; hence a larger interval must be sieved. The functional values of $X^2 - N$ increase almost linearly as the distance between X and $\sqrt{N}$, and as the magnitudes increase the frequency of factorization decreases. At Sandia we were able to factor integers of size about 10^{55} with the basic algorithm, but for larger numbers computing time was becoming intolerable.

We were able to modify the algorithm such that the size of the residues to be sieved was periodically reduced and hence our factorization success rate remained relatively constant. The means by which we obtained these sequences of smaller residues was by identifying large primes which divide a residue, then sieving on the subsequences guaranteed divisible by the primes. That is if $q \mid X^2 - N$, then $q \mid (X + kq)^2 - N$ for all integer k . If more than one factorization is obtained in the subsequence, the large prime can be eliminated and we have quadratic residues factored entirely into the prime base, B .

Independently, Peter Montgomery [M] suggested a somewhat different procedure by which polynomials may be selected such that they generate quadratic residues and the coefficients adjusted to minimize magnitudes. Also, with some modification, the sieving procedure still applies. Robert Silverman [S] has enjoyed great success using these polynomials with his parallel implementation of the quadratic sieve. Our latest code uses further variations of this idea.

Several other additions and modifications to the basic algorithm have enhanced its capability. The "large prime" variation locates prime divisors of residues beyond the prime base and uses these to generate completely factored residues. Also, one can use a multiplier with the number to be factored to enrich the prime base with small primes; hence making residue factorization more likely.

V. FALSE STARTS

Having no experience with parallel processing and because of limited local

memory, we were initially tempted to rely heavily on interprocessor communications and the use of different units to perform very different tasks. Each of these attempts bogged down because of overloading of the channels that enable the processors to talk to one another.

Because generation of polynomials, as suggested by Peter Montgomery, requires considerable multiple-precision arithmetic, we asked certain processors to generate these polynomials and initialization parameters, and to distribute this information to other nodes which could then do the sieving without multiple-precision. This idea seemed good in several respects. It frees up storage to be used for efficient sieving, and load balancing could be achieved by varying the number of nodes supplied by one polynomial generator. There is, of course, considerable information needed by a processor in order to begin the sieving, and apparently this was more than the lines could handle; communication time became prohibitive.

Another approach that was implemented was to apportion the prime base among a ring of processors, all sieving the same polynomial. Each processor in the ring would sieve with the set of primes it was given, then pass these to a neighbor. When each prime had visited each member of the ring, the sieving would be complete. After searching for and saving successful factorizations, a new polynomial would be started.

The above and other plans that would have used memory efficiently at the expense of increased interprocessor communication were programmed, but stymied by the traffic.

VI. CURRENT IMPLEMENTATION

We used quadratic polynomials of the form $A^2 X^2 + 2 B X + C$ to generate the residues to be factored. It must be the case that $B^2 - A^2 \cdot C = 0 \pmod{N}$; hence we take $B^2 - A^2 C = kN$, a small multiple of N (k is the multiplier used to enrich the set of small primes which divide residues). In order to minimize the amount of multiple-precision necessary, we choose our leading coefficient to be "small". We take A from a set of primes just larger than those in the base. This enables much of the computation of sieving parameters to be done in single precision. Montgomery and Silverman choose their coefficients much larger in order that roots of the quadratic are sufficiently close together that a sieving interval may contain both. Our

choice of much smaller coefficients forces the roots to be very far apart; hence we sieve over a pair of disjoint intervals each about a root of the polynomial. The magnitudes of the residues to be factored are not affected by this choice.

Figure 2. Sieving Intervals

As described earlier, communication overhead is an extreme problem with the NCUBE, hence for the major portion of the computation (sieving) we are asking each processor to do the same program with different parameter sets. It is efficient to sieve a long contiguous block in memory so the need to minimize stored program and data in each processor is pressing. In order to save memory for a large sieve array, we have eliminated as much multiprecision code as possible and actually recompute some values that could be stored. After the sieving is done with each prime power in the base, the array is searched for residues which are completely factored and those that factor except for one prime somewhat larger than those in the base (large prime variation). Identifiers of these residues are saved in order that the polynomials may be reconstructed and factored by division. In addition to the sieving operation, the above-mentioned search and multiple-precision division were identified as major consumers of computing time. A rewrite of the division package achieved a 13-fold speed-up. The sieve and search routine are particularly expensive for the NCUBE because it does not vectorize. When these were written in assembly code however, a great reduction in overhead was realized.

The final stages of the algorithm are the set-up and solution of the matrix used in the Gaussian elimination. Because of the very large matrix that must be processed, we must use memory more efficiently. Each processor is allocated identifiers for a certain set of the factored residues and a certain portion of the factor base. The functional values are calculated at each node and the available set of primes divided out. Results are then transferred to a neighboring node which operates on the residues with its assigned primes. When the residues have passed through all nodes, factorization is simultaneously completed. Each residue that completely factors forms a row, as does each large prime which repeats in another factorization. The abundance of large prime factorizations and hardware limitations on array size introduce complications into the matching algorithm. These we overcome by asymptotic estimation of the frequency of occurrence of large primes of various sizes and assigning a large prime to a given block according to its magnitude. Then, the matching algorithm needs only operate within a bin without crossing boundaries.

At this point, the matrix is ready for processing by Gaussian elimination. To deal with a large prime base, we must use available memory efficiently. Each node is assigned an equal number of rows corresponding to factored residues. Each bit of a row represents the parity of the

exponent to which the corresponding prime in B is raised in this factored residue. At this point we apply a Gaussian elimination procedure [PN] which is particularly memory efficient and suited to parallel processing. This yields the binary dependency which is then evaluated, and if non-trivial, we have factorization.

VII. Results

As stated earlier, we are still adapting this procedure to the hardware. As we attack larger integers, additional complications arise and changes are necessary. In terms of the numbers we have factored for comparison, we have been able to remain well below the computing times achieved on the CRAY. Below we list some of the integers we have factored with both of our codes for comparison. The figures in parentheses refer to the CRAY code, and designation refers to the Cunningham Tables [BLSTW].

Designation	C[2,193-]	C[5,79-]	V 471*	C[2,211-]
Magnitude	9.1×10^{50}	4.1×10^{54}	2.5×10^{56}	2.2×10^{59}
Primes in Base	999 (6514)	1278 (6800)	1366 (5000)	2036 (6671)
Number of Polynomials	6518 (188)	1648 (81)	899 (1000)	4042 (27)
Sieve Time In Hours	.043 (.425)	.044 (.66)	.047 (.22)	.193 (22.0)

FIGURE 3 Comparison of Factorization

*A Fibonacci number suggested by Peter Montgomery.

References

- [BLSTW] J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, S. S. Wagstaff, Jr., Factorization of $b^n \pm 1$ up to High Powers, American Math. Soc., 1983.
- [DH1] J. A. Davis, D. B. Holdridge, Factorization Using the Quadratic Sieve Algorithm, Sandia National Laboratories Report, SAND 83-1346, Dec., 1983
- [DH2] J. A. Davis, D. B. Holdridge, Most Wanted Factorizations Using the Quadratic Sieve, Sandia National Laboratories Report, SAND 84-1658, Aug., 1984
- [M] P. Montgomery, Personal Communications, 19 Feb. 1984
- [PW] D. Parkinson, M. C. Wunderlich, "A Memory Efficient Algorithm for Gaussian Elimination over GF(2) on Parallel Computers", Personal Communication, Feb., 1983.
- [S] R. D. Silverman, "The Multiple Polynomial Quadratic Sieve"; Math. Comp. V. 48 No. 177, Jan., 1987.