W&L, whereby Paul teaches at St. Olaf in the fall and at W&L in the winter has continued for the past five years and has resulted in another nine or so collaborative mathematical publications. Most recently we have been investigating applications of real analysis to possible answers to questions dealing with algorithms for representing functions graphically on computers, questions raised by numerical analysts. Hopefully, this arrangement between the two schools will continue for several more years, even as Mike marks his final year of full-time teaching at W&L and moves toward phased retirement.

In summary, it's been a fun ride for 33 years with their accompanying 38 joint publications, and we're not done. Our hair is a little thinner and a lot grayer, we have few non-arthritic joints between us, and we are more likely to celebrate a new theorem with a story about one of our grandchildren than with a cheap cigar, but we're still at it. There seem to be plenty of epsilons and deltas that we haven't used yet. So, if you are on the Colonnade next winter term on a Tuesday or Thursday afternoon, stop by Robinson 34, but bring your own coffee and sense of humor.

Optimized Parallel Implementation of the Quadratic Sieve Factorization Algorithm

Andrew G. West, Washington and Lee University '07

The problem of breaking a composite number into its unique prime factors is one that has puzzled mathematicians for over two millenia. Indeed, it was around 300 B.C. when Euclid laid the framework for what would later be proven and formalized as the Fundamental Theorem of Arithmetic (FTOA). The FTOA states that every natural number greater than 1 has a unique representation as a product of primes [5]. The purpose of prime factorization is to determine this representation.

To produce prime decompositions, mathematics has advanced far beyond simplisitic and familiar methods such as trial division. The method discussed herein, the Quadratic Sieve (QS), is a combination of number theory, linear algebra, and brute processing power. Invented by Carl Pomerance in 1981, it remains the second fastest known general-purpose factorization algorithm as of this writing [4]. Due to structural similarities with faster methods, QS still resides in the realm of academic significance.

To produce factorizations of a given N, QS relies heavily on congruences of squares. Fermat demonstrated one could factor N by finding a, b such that $N = a^2 - b^2 = (a + b)(a - b)$. Such a and b are difficult to find. However, if one can find a and b satisfying $a^2 = b^2 \bmod N$, there is a good probability that gcd $(a \pm b, N)$ are the factors of N [5]. Because the congruence operator is less constrictive than the equality operator, there is some probability the greatest common denominator function will produce trivial results (1 or N as the result of relative-primality). As a result, modern algorithms, such as Dixon's method and QS, focus on finding multiple a, b to diminish the probability of failure.

Dixon's factorization algorithm is the predecessor to QS and with slight modification is a good place to begin discussion of QS itself. First, all primes less than a given prime bound, B, are stored in a data structure known as the prime base $\{PB\}$. Then, $f(x) = (x + \sqrt{N})^2 - N$ are generated on the interval $\sqrt{N} \leq x < \sqrt{2N}$ where $x \in N$. One looks for $f(x)$ which factor completely over the primes in $\{PB\}$ or mathematically, are $\{PB\}$-smooth. These are called relations. For each relation, a modulo 2 exponent vector is built of its factorization over the prime base [2]. For example if $\{PB\} = \{2, 3, 5, 7\}$ and $f(x) = 370 = 2^1 + 3^0 + 5^2 + 7^3$, then the mod 2 exponent vector is [1 0 0 1].

Once "sufficient" relations are found, these vectors are placed as rows into a matrix, the identity augmented, and matrix (e.g. Gaussian) elimination performed. Linear dependencies between the rows should produce several zero vectors (which correspond to perfect squares) and the adjacent (former) identity tells what combination of relations produced the vector. For each such vector, the products of all $f(x)$ and x in the combination are computed and stored as a and b, respectively. These a and b fulfill the previously mentioned congruence of squares and just as before gcd $(a \pm b, N)$ has decent probability of being non-trivial factors of N.

QS improves upon this thinking. Whereas Dixon uses brute force to try to factor each $f(x)$ over the prime-base, QS uses the more elegant "sieving." Under QS, a prime can only be added to {PB} if it has a quadratic residue modulo N. A number P is said to be a quadratic residue modulo N with solve value x if there exists a x such that $N \equiv x^2 \bmod P$. The solve values tells an individual which f(x) a given prime will be a factor [2]. For example, if the residue solve value of prime 7 is 3, it is known that 7 is a factor of $f(x)$ when $x = 3, (3 + 7), (3 + 2(7)), (3 + 3(7)) \ldots$ Arranging many sequential $f(x)$ into a block and using this logic for the length of

the block over the entirety of {PB} provides significant time savings compared to Dixon's method.

The focus of the current work (as advised by Rance Necaise, Department of Computer Science) is to build an optimized parallel implementation of QS. While literature addresses the topic, rarely does it focus on implementation-level detail aimed at an undergraduate audience nor are there claims made to optimization of the code within. While well-established optimizations exist, it is our goal to combine them with more subtle coding strategies and support these implementation decisions via mathematical reasoning and runtime statistics.

Code was written in C/C++ using LAM/MPI for parallel communications. Because no primitive datatype is capable of storing and manipulating variables on the order of N, the GNU Multiple Precision Arithmetic library (GMP) is used. All testing and timings are performed on The Inferno, Washington and Lee University's 48-node Beowulf cluster.

While optimization is not quantifiable, our executable has undergone many refinements and it can been demonstrated it is signficantly faster than existing (non-optimized) implementations of the same variety [1]. Our largest factorization completed to date is a 75 decimal digit (249 bit) semiprime, the product of two 38 digit primes. This decomposition took ≈ 26 hours on 48 nodes.

Most surprising is the amount of estimation necessary for an efficient implementation. For large N, sometimes millions of sequential $f(x)$ must be examined to find a single relation. Through logarithmic estimation strategies, function generation can be amortized into near constant time and sieving divisions can be reduced to subtractions. These strategies indicate "probable relations" which are then verified using more rigorous techniques.

Also of note are the expotential time and space growth rates of the algorithm. While the factorization of T_{75} took 26 hours, T_{65}, a semiprime just ten digits smaller can be completed in about 10 minutes. Similarly, memory requirements are large. A single exponent vector usually contains many thousand elements and ≈ $|\{PB\}|$ such vectors are needed to exit the sieving stage. In addition, there are allocations necessary for {PB} and associated residue solve values.

One should note our particular implementation concerns itself only with the more basic (single polynomial) QS. An altered version known as the Multiple Polynomial Quadratic Sieve (MPQS) periodically changes the function $f(x)$ and with this resets x to the lower bound. Smaller x generate smaller $f(x)$ which in turn are more likely to be $\{PB\}$-smooth and a relation. Algorithm author Carl Pomerance notes, "QS with multiple polynomials runs about 17 times as fast as the basic QS method" [2]. However, the improvement introduces a host of independent variables which obfuscate runtimes. For this reason, multiple polynomials are not examined in our analysis.

QS (and its variants) remain the fastest known method for factorizations of up to 100 digits. Efficient factorizations beyond this point require the General Number Field Sieve (GNFS), currently the fastest known algorithm. GNFS is structurally similar to QS but includes higher order polynomials and algebraic rings. Even using supercomputers and across the largest parallel environments, the largest general-purpose factorization ever is of RSA-200, a 200 decimal digit semiprime that took 18 months to factor on an 80 node cluster [3].

Even in the advanced state of the art, there exists no algorithm with polynomial runtime complexity. This fact forms the basis of several modern cryptography systems, most notably the RSA public-key algorithm (which uses 512 bit keys). If large N could be factored quickly the foundations of modern network security would be destroyed. A polynomial time method does exist for quantum computers, Shor's algorithm, but can not be implemented because no such machine exists.

REFERENCES

1. Olof Åsbrink and Joel Brynielsson. Factoring large integers using parallel quadratic sieve. Technical report, Royal Institute of Technology, Sweden, April 2000.
2. Richard Crandall and Carl Pomerance. *Prime Numbers: A Computational Perspective*, chapter 6, Subexpotential factoring algorithms, pages 225–238. Springer-Verlag, New York, 2001.
3. RSA Laboratories. RSA-200 is Factored! online: http://www.rsa.com/rsalabs/node.asp?id=2879, May 2005.
4. Carl Pomerance. The quadratic sieve factoring algorithm. In N. Cot T. Beth and I. Ingemarsson, editors, *Advances in Cryptology, Proceedings of EUROCRYPT*, Paris, 1984. EUROCRYPT, Springer-Verlag.
5. Eric W. Weisstein. MathWorld - a Wolfram web resource. online: http://www.mathworld.com.