Parallel Implementation of Elliptic Curve Method for Integer Factorization Using Message-Passing Interface (MPI)

E. Wolski[1], Joel G. S. Filho[2], M. A. R. Dantas[3]
[1] Electrical Engineering Department, University of Brasilia
Brasilia 70919-970, Brazil
{wolski@unb.br}
[2] Electrical Engineering Department, University of Brasilia
Brasilia 70919-970, Brazil
{joelgf@ene.unb.br}
[3] Computer Science Department, University of Brasilia
Brasilia 70919-970, Brazil
{mario@cic.unb.br}

Abstract—

One of the most prominent systems for securing electronic information, known as RSA (Rivest-Shamir-Adleman) [RIV78], relies upon the fact that it is computationally difficult to factor a *large* integer into its component prime integers. If an efficient algorithm is developed that can factor any arbitrarily large integer in a *reasonable* amount of time, the security value of the RSA system would be nullified.

In this paper we present the Elliptic Curve Method for integer factorization and results of its parallel implementation using Message-Passing Interface (MPI).

Keywords— **Integer factorization, RSA, Elliptic Curve Method, Parallel Processing, Cluster Computing.**

I. Introduction

The main methods developed until today for integer factorization are the Elliptic Curve Method (ECM), the Multiple Polynomial Quadratic Sieve (MPQS) and the Number Field Sieve (NFS).

ECM is a factoring method, due to H. W. Lenstra [LEN87], whose expected run time is $O(\exp(c(\ln p \ln \ln p)^{1/2})(\ln n)^2)$, where $c \approx 2$ is a constant, n is the number to be factored and p is a nontrivial factor of n. This is a sub-exponential time, mainly dependent of the size of the factor p.

MPQS [POM85] and NFS [LEN93, LEN90] are methods that use the approach of factor base and their performances depend mainly on the size of n, the number to be factored. MPQS has expected run time $O(\exp(c(\ln n \ln \ln n)^{1/2}))$, where $c \approx 1$ is constant. NFS has expected time $O(\exp(c(\ln n)^{1/3}(\ln \ln n)^{2/3}))$, where c is a constant depending on details of the algorithm and on the form of n (an admissible value is $c = (64/9)^{1/3}$). This is asymptotically considerably better than the other two.

In practice, all the three methods are important. Given a large integer, with no information about the sizes of their nontrivial factors, we could, typically, try ECM until the cofactor (i.e. the quotient after successive divisions by known prime factors) of the original number be sufficiently small. Then we could try MPQS or NFS (in the case that cofactor is greater than about 110 digits), if the cofactor doesn't be itself a prime number.

In this paper we are interested, particularly, on the results of implementation of ECM using Message-Passing Interface (MPI) on a cluster of workstations. Some results of synchronous and asynchronous communication implementation are also presented.

The paper is organized as follows. In section 2 the parallel cluster environment considered in the article is presented. In section 3 we explain the ECM. Our experimental results are presented in section 4. Finally, in section 5, we present our conclusions.

II. Parallel Cluster Environment

Workstations clustered together either physically or virtually represent interesting hardware platforms to meet the growing computational demand of many organizations. In addition, as presented in [DAN01], tacking some issues of this configuration (e.g. load balancing) the environment can be considered to some complex applications such as the factorization problem.

The configuration considered in the article is represented by eight loosely-coupled IBM personal computers (without monitor, keyboard or mouse) interconnect by a 100 Mbps switch. The switch has an equivalent function of the interconnection network found on parallel machine. Only one machine has all peripheral items (monitor, keyboard and mouse) because it works as a monitor of the cluster (i.e. a machine from where we install all software packages and some management actions are taken). Table I shows more characteristics of cluster environment.

TABLE I

CLUSTER ENVIRONMENT

Processor Architecture	Intel Pentium II
Clock (MHz)	350 MHz
Cache	512 Kb
Memory	32 Mb
Switch throughput	10-100 Mbps
Operating System	Linux kernel 2.2.16

The cluster was designed to execute only local parallel applications and it was not connected to main campus network. All experiments were executed in a quite workload environment (i.e. the cluster was dedicated to our experiments without any other user or application).

Parallel software environments are designed to enhance the execution of concurrent tasks, achieving reasonable parallel speedup. The parallel programming environment used in this work is based on MPI standard [MPI94, PAR94]. The particular implementation used is *mpich* 1.2.0, from the Argonne National Laboratories.

Multiple-precision arithmetic is necessary because the number n which we want to factor may be much larger than can be represented in a single computer word (otherwise the problem is trivial). As a multiple-precision arithmetic package, it was used the Miracl (Multiprecision Integer and Rational Arithmetic C/c++ Library), version 4.4 [MIR01].

III. Elliptic Curve Method

In 1985 H. W. Lenstra proposed the ECM, one of the three main methods in use today, together with MPQS and NFS. It possesses a number of properties which make it useful even if it is only used in conjunction with other algorithms.

An elliptic curve over a field K of characteristic not 2 is the set of solutions $(x, y) \in K \times K$ to a cubic equation

$$y^2 = x^3 + ax^2 + bx + c, \qquad (1)$$

together with a special point (conceptually (∞,∞)) called the point at infinity. There is one restriction to the coefficients in equation 1: the discriminant of the cubic polynomial must be nonzero, i.e.,

$$-4a^3c + a^2b^2 + 18abc - 4b^3 - 27c^2 \neq 0$$

The points on an elliptic curve form an abelian group $E(K)$, when the group operations are suitably defined. The negation of the point at infinity is itself; the negation of any other point $P = (x_1, y_1)$ is defined to be $-P = (x_1, -y_1)$. For addition, suppose that P and Q are two points on the elliptic curve. If either the P or Q is the point at infinity,

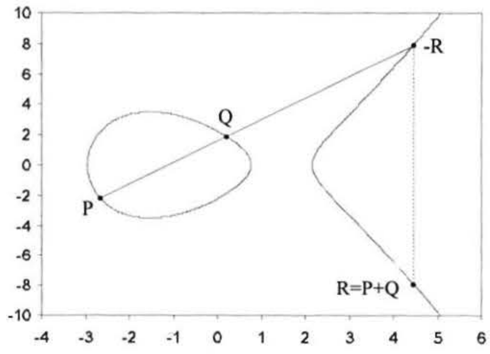

Fig. 1. Point addition at curve $y^2 = x^3 - 7x + 5$

then define $P + Q$ to be the other point. Otherwise suppose $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. If $x_1 \neq x_2$, then define $P + Q = R$, where $-R$ is the point where the straight line through P and Q re-intersects (see figure 1). If instead $P = Q$, the duplication of the point P is the point where the straight line tangent to the curve in P re-intersects. A calculation gives

$$R = (x_3, y_3)$$

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_3 - x_1) + y_1$$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \text{ if } P \neq Q$$

$$\lambda = \frac{3x_1^2 + a}{2y_1} \text{ if } P = Q.$$

All group operations are defined in terms of ordinary addition, subtraction, multiplication, division, and comparison (no square roots), and are meaningful over arbitrary fields where $2 \neq 0$. In particular, they are meaningful if $K = GF(p)$, where $GF(p)$ is the finite field of order the odd prime p. The resulting elliptic curve group $E(GF(p))$ is finite; Hasse [SIL86] showed that its order is $p + 1 - \tau$, where $|\tau| \leq 2\sqrt{p}$. By changing the constants in equation 1, we get another curve, whose order is usually different.

If n is composite, say $n = pq$ where p and q are distinct odd primes, then the ring $\mathbb{Z}/n\mathbb{Z}$ is not a field, but we can use equation 1 to define a curve and attempt to use the algebraic rules to do group operations modulo n . We will fail (i.e., be unable to execute the algebraic operations) only if we attempt to divide by a nonzero, non-invertible number modulo n. Such a denominator (called a zero divisor) will be divisible by p or q but not both, and will give us a factor of n.

A. ECM Examples

1. In 1995, it was completed the the factorization of the 309-decimal digit (1025-bit) Fermat number $F_{10} = 2^{2^{10}} + 1$. In fact

$$F_{10} = 45592577 \cdot 6487031809 \cdot$$

$$4659775785220018543264560743076778192897 \cdot p_{252}$$

where $46597\ldots92897$ is a 40-digit prime and $p_{252} = 13043\ldots24577$ is a 252-digit prime. The computation, which is described in [BRE99],took about 240 Mips-years. A Mips-year is the amount of computation that can be performed in one year by a single DEC/VAX 11/780.
2. The largest factor known to have been found by ECM is the 54-digit factor

$$484061254276878368125726870$$

$$789180231995964870094916937$$

of $(6^{43} - 1)^{42} + 1$, found by Nik Lygeros and Michel Mizony with Paul Zimmermann's GMP-ECM program [ZIM99] in December 1999 (for more details, see [BRE00]).

B. The Second Phase

Lenstra elliptic curve algorithm can be speeded up by the addition of a second phase. The idea of the second phase is to find a factor in the case that the first phase terminates with a group element $P \neq (\infty, \infty)$, such that $|\langle P\rangle|$ is reasonably small (say $O(m^2)$). ($\langle P\rangle$ is a cyclic group generated by P.) There are several possible implementations of the second phase. One of the simplest uses a pseudorandom walk in $\langle P\rangle$. There is a good chance that two points in the random walk will coincide after $O(|\langle P\rangle|)$ steps, and when this occurs a nontrivial factor of n can usually be found. Details of this and other implementations of the second phase may be found in [BRE86, BRE99, DIX93].

C. Parallel Implementation of ECM

Using ECM, each new trial (i.e, each new curve) is independent. So long as the expected number of trials is much larger than the number P of processors available, linear speedup is possible by performing P trials in parallel. In fact, if T_1 is the expected run time on one processor, then the expected run time on a MIMD parallel machine with P processors [BRE99] is

$$T_P = \frac{T_1}{P} + O(T_1^{1/2+\epsilon}).$$

IV. Experimental Results

The experimental work consisted of implementing the parallel version of the ECM using *mpich*, based on sequential version available in Miracl. The programming language used was C.

The implementation was based on a SPMD (Simple Program Multiple Data) approach and on the following procedures:

1. We took the sequential implementation of the Lentra algorithm available in the Miracl package.
2. The partitioning phase was naturally defined (i.e., for each new curve we have a partition).
3. The communication phase was defined as follows: the master process performs an initial broadcast in order to transmit to all processes the value and size of the number n to be factored. All remaining communication is made through send/receive primitives, existing two versions to be compared: synchronous and asynchronous mode. We use control messages to tell a process that it will receive the parameters of a new trial, or to tell it to finalize the program. Other control messages inform the master process that a slave found a factor or that this slave wants new parameters to perform another trial. The data messages are used to tell a process the parameters of a new curve, in master-slave direction, or to receive the found factor, as this is the case, in slave-master direction.
4. The agglomeration and mapping phases tend to become a single phase in SPMD implementation (as the mapping becomes implicit to agglomeration) [FOS95]. The single master and $P-1$ slaves approach was convenient for the processor amount used and the number size to be factored, as we will see in the graphics. Greater number sizes to be factored and processors may require other forms of agglomerations and mapping, like binary trees.

The data used in the tests was composed by the integers that was factored, in all cases generated as the product of two primes. The size of the integers factored and their factors are variable and they are presented in each test. The results consider:

- the average time of the values obtained during several executions with same source of data.
- the average time of the values obtained during several executions with different source of data, when some randomness is required.
- dedication of the cluster to the experiments without any other user or application (quite workload).
- time precision in seconds, since the total times are greater enough to demmand a finer precision.

The performed tests looked for results that:

1. check the run time dependency of ECM related to the nontrivial factor size.

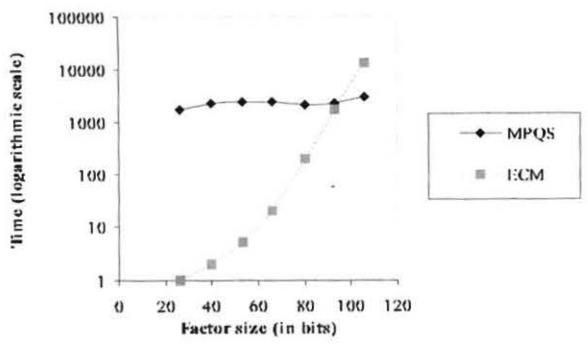

Fig. 2. Graphic of obtained run times in the asynchronous parallel implementation with fixed n

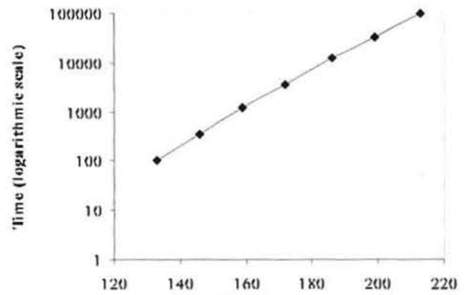

Fig. 3. Graphic of obtained run times in the asynchronous parallel implementation for several values of n

2. plot the increasing curve of run times, as the size of the problem increase.
3. check the parallel implementation of ECM considering the speedup, efficiency and scalability metrics.
4. compare run times between synchronous and asynchronous versions of ECM implementation.

In the first test we confirm that the expected run time of ECM is dependent on the factor size.

Using the asynchronous parallel version and executing on the 8 processors, we obtain the graphic of the ECM curve, showed in figure 2, where n is a 64-digit number (≈ 212 bits) and p has variable sizes $(26, 40, 53, 66, 80, 93, 106$ bits$)$.

We can observe the increasing run time of ECM related to the factor size. As MPQS (see figure 2) and NFS has an approximately constant run time behavior in function of the factor size, a nice strategy to apply in a conjugate use of methods to factor random large integers would be run first ECM until a limit of curves, or until we find small prime factors, and then run MPQS or NFS to factor the cofactor.

In the second test, we present the sub-exponential behavior of run times of ECM as the size of n increases. All such used

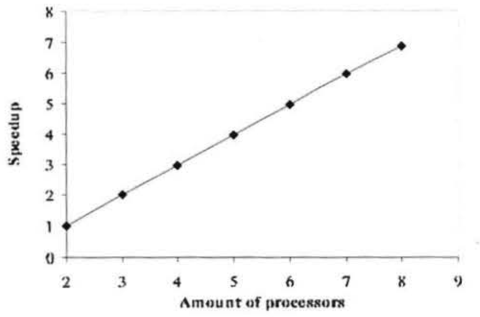

Fig. 4. Graphic of the speedup obtained

n have half size factors, which is the worst case for ECM. But this fact doesn't change the curve tendency. We can observe in the graphic of figure 3 that the increased times fit to a straight line (considering the logarithmic scale). It is risky to make an extrapolation of these results, since the size of factored numbers and the amount of processors are limited.

The third test checks the speedup obtained in the execution of processes in the cluster. We took 64-digit integers (212 bits) and factored them using several amount of processors. The results are presented in the graphic of the figure 4.

We define the speedup as a performance metric which states the ratio between the average sequential code time execution and the average required time to execute the same problem in a specific parallel architecture with P processors. Indeed, the *speedup* of a parallel algorithm is given by $S = \frac{T_S}{T_P}$.

Considering that we have few processors and that we adopted a strategy based on *divide-and-conquer* parallelism, we didn't include the master process in the computation of speedup, since this process doesn't perform a "trial". Proceeding accordingly, we have speedup 1 with two processors at the origin. If we work with a greater amount of processors, this detail would be less important.

ECM presents speedup very close to the linear one, for the amount of processors used, as we could expect from the theoretical considerations.

Let W be the problem size, i.e. the time complexity T_S of the sequential problem. This means that, if we have an increase of the problem size, we always have a proportional increase of computation. Furthermore, we can consider the interprocess communication (IPC) as the biggest overhead actor, here denoted by T_0. The overhead is a function of W and P (the amount of processors) and we can define it as:

$$T_0(W, P) = PT_P - W \tag{2}$$

From experimental results, we can obtain for equation 2 a very small overhead value. Therefore, the efficiency E is,

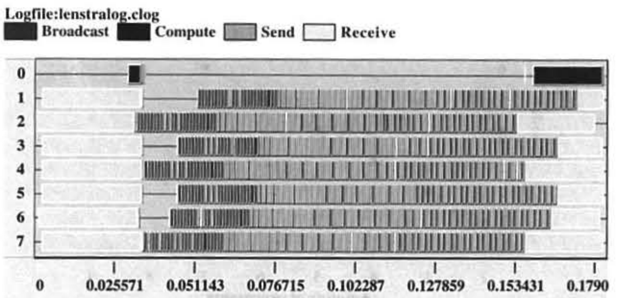

Fig. 5. Graphic of ECM execution on a 212-bit integer with a 40-bit factor

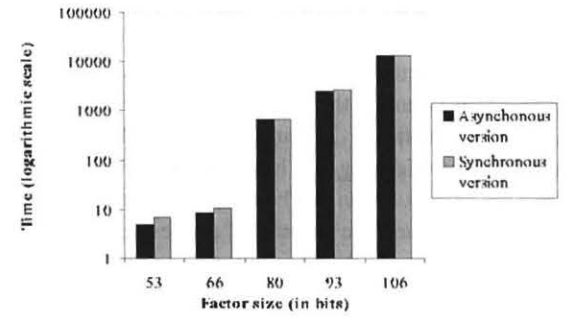

Fig. 6. Histogram of times obtained in synchronous and asynchronous implementation of ECM

basically, maximal. The figure 5 shows, graphically, that the overhead occurs at the beginning and at the end of processing, being the intermediate time used completely for the factoring computation, i.e. the communication/computation ratio is small. The graphic of this figure was generated from a *mpich* application called jumpshot. It shows, at y-axis, several events occuring in processors 0 to 7. The events are listed at the legend. At x-axis the graphic presents the time scale of events occurrences. The small amount of processors in the cluster, however, doesn't allow to measure, experimentally, the ratio between the size problem W and the number of processors P, to which we have the efficiency maintained.

As a final test we present the performance difference between the synchronous and asynchronous parallel implementations of ECM. The histogram of the figure 6 shows this difference, although small, always pointing to a better performance of the asynchronous version. This occurs since, in asynchronous version, the master process may not wait until the next slave process sends a message to it. Instead, the master checks whether there is a message from this process. If not, it follows probing the next process. Later, it probes that process again, and receives the message, as it is the case. Indeed, the whole execution is less dependent of slow processes.

It is important to note that our cluster is composed by homogeneous workstations. The showed difference tends to increase in favor of the asynchronous version using heterogeneous stations in the cluster, as the processing time difference among the workstations is higher than in the synchronous case.

V. Conclusions

Among the several known factorization methods, Number Field Sieve is the algorithm wich presents the better asymptotically execution time. But Elliptic Curve Method and Multiple Polynomial Quadratic Sieve still play important roles in this problem. Particularly, ECM can be used as a first approach if we do not know the factors sizes. Lenstra and Verheul [LEN99] do a prospection of the required computational power to factor several integers. A short extract from the data presented by them is showed in the table II.

TABLE II
REQUIRED COMPUTATIONAL POWER TO FACTORING INTEGERS USING NFS

number of bits of n	Required Mips-years
768	4×10^8
1024	2×10^{10}
1280	5×10^{11}
1536	9×10^{12}
2048	1×10^{15}

Still from this paper of Lenstra and Verheul [LEN99], RSA of 512 and 768 bits key sizes are nomore secure. Keys of 1024 bits will be secure until 2002 and keys of 1280 bits will be secure until 2008.

It is impossible to predict the next advance in integer factorization, but we could estimate the cryptanalytic progress based on the evolution in the last 20 years. The number of expected months, in average, such that cryptanalytic developments affecting classic asymmetric systems (RSA and discrete logarithm) become two-fold more effective is 18, i.e. we must wait 18 months, from now on, for an attack to the same classic asymmetric system does cost half of today computational effort. Therefore, the algorithmic development has been, historically, comparable to the hardware development (Moore's Law).

The question of adopting a greater or smaller RSA-modulus depends on the sensibility of the data records to be protected. For many records the security requirements are not serious, in that loss of secrecy in 10 months or 10 days is acceptable. However, for some records, even 10-year protection is not sufficient. In these cases it might be prudent to use 5,000-bit moduli or greater.

The multiple-precision library plays an important role in cryptographic applications. One must have special attention to use good implementation of basic operations over rings and fields.

The ECM parallel implementation is very efficient, making the linear -peedup a possible deal. It could be used as a parameter to test a distributed environment ("vanille" application).

It is important to the programmer to know the characteristics of the algorithm to implement and the parallel environment in order to explore the better implementation form.

It is worth to mention that a very active research area is the parallel programming based in patterns (also known as template-based programming). The aim is to offer to the programmer an abstract environment that allows him/her, simply based on high level patterns of parallel programming, to build parallel programs. As an example see POOMA (Parallel Object-Oriented Methods and Applications), developed by American Energy Department's Advanced Computing Laboratory [POO01].

MPI offers a good base for parallel programming development, using different environments (from network of workstations to massive parallel processors machines), allowing for the parallel computation a great advance in the next few years.

ACKNOWLEDGEMENTS

This paper used the laboratory of Computer Science Department of University of Brasilia. We gratefully acknowledge their contribution.

REFERENCES

[LEN93] LENSTRA, A. K. and LENSTRA, H. W. Jr. (editors). The Development of the Number Field Sieve. **Lecture Notes on Mathematics 1554**, Springer-Verlag, Berlin, 1993.

[FOS95] FOSTER, I. Designing and Building Parallel Programs. **Addison-Wesley Longman, Inc.**, Sidney, Australia, 1995.

[SIL86] SILVERMAN, J. H. The Arithmetic of Elliptic Curves. **volume 106 of Graduate Texts in Mathematics**, page 131, Springer-Verlag, New York, 1986.

[RIV78] RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, v.21, p.120-126, 1978.

[LEN87] LENSTRA, H. W. Jr. Factoring integers with elliptic curves. **Annals of Mathematics**, v.126, p.649-673, 1987.

[POM85] POMERANCE, C. The quadratic sieve factoring algorithm. **Advances in Cryptology, Proc. Eurocrypt '84**, v.LNCS 209, p.169-182, 1985.

[LEN90] LENSTRA, A. K.; LENSTRA, H. W. Jr.; MANASSE, M. S.; POLLARD, J. M. The number field sieve. **Proc 22nd Annual ACM Conference on Theory of Computing**, Baltimore, Maryland, p.564-572, 1990.

[MPI94] MPI-Forum. MPI: A Message-Passing Interface Standard. **International Journal of Supercomputer Application**, v.8, n.3-4, 1994.

[PAR94] PARALLEL COMPUTING, Special issue: Message passing interface. **Parallel Computing**, v.20, n.4, 1994.

[BRE99] BRENT, R. P. Factorization of the tenth Fermat number. **Math. Comp.**, v.68, p.429-451, 1999.

[BRE86] BRENT. R. P. Some integer factorisation algorithms using elliptic curves. **Australian Computer Science Communications**, v.8, p.149-163, 1986.

[DIX93] DIXON, B.; LENSTRA, A. K. Massively parallel elliptic curve factoring. **Proc. Eurocrypt'92**, v.LNCS 658, p.183-193, 1993.

[BRE99] BRENT, R. P. Some parallel algorithms for integer factorization. **European Conference on Parallel Processing**, p.1-22, 1999.

[LEN99] LENSTRA, A. K.; VERHEUL, E. R. Selecting Cryptographic Key Sizes. **http://www.cryptosavvy.com/**, 1999.

[ZIM99] ZIMMERMAN, P. The ECMNET Project. **http://www.loria.fr/~zimmerma/records/ecmnet.html**, 1999.

[BRE00] BRENT, R. P. Large factors found by ECM. **Oxford University Computing Laboratory - ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Richard.Brent/cha** 2000.

[DAN01] DANTAS, M. A. R.; LOPES, F. M. Improving load balancing in a parallel cluster environment using mobile agents. **Accepted paper HPCN2001 Europe, Amsterdam**, 2001.

[MIR01] SHAMUS SOFTWARE. Miracl. **http://indigo.ie/~mscott/**, 2001.

[POO01] ADVANCED COMPUTING LABORATORY. POOMA. **http://www.acl.lanl.gov/pooma**, 2001.