Parallelization of Pollard-rho factorization

Richard E. Crandall
Center for Advanced Computation, Reed College, Portland OR

Abstract. It is an established heuristic that if each of m machines carries out an independent (in a very specific statistical sense) "Pollard-rho"—sometimes called "Monte Carlo"—factorization sequence, then the expected time to discover a prime factor p of N is on the order of

$$\tau_N \ \frac{\sqrt{p}}{\sqrt{m}},$$

where τ_N is the time for a multiplication (mod N). This gain of $\sqrt{m}$ from such parallelization of m machines is of course disappointing. Herein we show that, by invoking appropriate fast algorithms to resolve certain correlation products, one may obtain (if the heuristics be valid) virtually ideal parallelism, in the sense that the time to discover p is now estimated as:

$$\tau_N \ \frac{\sqrt{p}\log^2 m}{m},$$

with somewhat lower complexities obtainable in certain cases via known modifications of the Pollard iteration itself. On this heuristic for parallelism, various factoring labors are assessed in regard to their practicality.

DRAFT 14 Jun 1999

1. Motivation for parallel Pollard-rho

The "Pollard-rho" factorization method is based on an idea of [Pollard 1975] to employ random, or "Monte Carlo" sequences to discover, by virtue of a deterministic periodicity inherent to certain such sequences, hidden factors of a given number N. The method has enjoyed a certain vogue, peaking in a sense with the impressive factorization of $F_8 = 2^{2^8} + 1$ about two decades ago [Brent and Pollard 1981]. In general, the Pollard-rho method or any of its modern variants and enhancements [Montgomery 1987][Brent 1980] discovers a hidden prime factor p of N in time

$$O(\tau_N \sqrt{p}), \tag{1.1}$$

where τ_N is the time for a ring multiplication in Z_N. We remind ourselves right off that this $p^{1/2}$ behavior is heuristic, yet seems to be quite the case in practice. All of these established enhancements alter only the implied big-O constant, so we can think of any Pollard-rho variant as requiring $O(p^{1/2})$ operations in some statistical sense, with worst-case factoring of composite N involving therefore $O(N^{1/4})$ operations.

Though the elliptic curve method (ECM) of H. Lenstra has become a dominant scheme for extracting relatively small factors of large numbers, there remains a genuine niche for Pollard-rho and its variants. The Pollard-rho amounts in a certain sense to a statistical sieve with which one can typically attain sieving bounds as high as 10^{12} or more on possible p, at least with reasonable probability that all relevant p have thus been checked. (One infers from the above heuristic big-O estimate that only a few million Pollard iterations are required to saturate with high chance such a sieve limit.) As pointed out by [Brent 1990], the use of m machines in an elementary attempt to parallelize the Pollard-rho, say by running each machine independently (meaning, with independent iteration polynomials and independent random starting values, as we later clarify) stopping the entire ensemble of machines when some machine finds a factor p, should result in expected discovery time

$$O(\tau_N \frac{\sqrt{p}}{\sqrt{m}}). \tag{1.2}$$

There is an easy heuristic argument that explains the disappointing acceleration factor of only $\sqrt{m}$, as we shall see.

Because it is a kind of simple sieve, the Pollard-rho method still has applications in computational number theory. For example, pre-factoring of residues from other problems, quick factorization of elliptic curve orders in elliptic-curve primality proving (ECPP), and so on. What is more, there are other domains in which a parellelized rho method might apply. For example, as known to Brent and Pollard, if the hidden prime factor of N is known to be of the form $p \equiv 1 (\bmod\ 2K)$ then the time to discover p is reduced further by about $(\log_2 K)/\sqrt{2K-1}$, the logarithm appearing due to extra operations in a modified, K-dependent rho sequence. So for such as Fermat numbers $F_n = 2^{2^n} + 1$, with $n > 1$, any prime factor of which being $\equiv 1 (\bmod\ 2^{n+2})$, the rho method enjoys a substantial boost; witness the success on F_8 as mentioned above, for which Brent and Pollard used $K = 512$. For example, the recent (April 1999) discovery by R. McIntosh and C. Tardif, namely the factor

$$81274690703860512587777 \quad | \ F_{18}$$

though found via a particular ECM variant [Crandall and Pomerance 1999][Brent et. al. 1999], could in principle have been found on the same type of machinery, as we argue later, via the new parallel realization of the rho method.

Recently, [van Oorschot and Wiener 1999] reported a clever way of applying a rho method to the discrete logarithm (DL) problem (mod p) in parallel fashion. In their use of "distinguished points" the utilization of m machines is ideal, in that the time to solve DL is reduced by a factor of m, so that a time bound of the type (1.2) is obtained, but with m in the denominator instead of $\sqrt{m}$. However, as pointed out by [Wiener 1999], there is so far no known extension of their parallel-DL scheme to factorization *per se*.

The purpose of the present paper is to show how Pollard-rho factoring can, with m machines operating in parallel fashion, and under heuristic assumptions about the statistics of the relevant iterations, indeed be achieved in time

$$O(\tau_N \frac{\sqrt{p}\log^2 m}{m}), \tag{1.3}$$

using a method which appears on the face of it to be unrelated to the DL parallelism of van Oorschot and Wiener.

2. Statistical and algebraic pictures

The original—and in many ways still the most elegant—manifestation of the idea of Pollard is, for given N to be factored, to start with random x_0, a and iterate a quadratic map:

$$x_{n+1} := (x_n^2 + a) \bmod N, \tag{2.1}$$

where here and elsewhere the final operation on the right is reduction to least nonnegative residue (mod N). Normally one uses an a parameter $\neq 0, -2$, to ensure a suitable pseudorandom sequence $\{x_n\}$. If p be a hidden prime factor of N, an occurrence of $x_i = x_j \pmod p$ for $i \neq j$ is sought, in the sense that p (sometimes along with yet other prime factors of N) might be the result of an operation $\gcd(x_i - x_j, N)$. Thus the Pollard sequence (2.1) is likely to succeed if we can detect such an i, j-collision via the gcd. The celebrated Floyd expedient is to observe that the fact of a collision also means there must be an instance of $x_{2k} = x_k$, some k; indeed the least such successful k does not exceed the first i for which there is a collision of x_i and $x_{j<i}$. This means one need not check pairs $x_i - x_j$ directly, testing instead various $\gcd(x_{2i} - x_i, N)$. Now one sees how very simple the rho method is; one only need initialize random $x = w$, and iterate:

$$x := (x^2 + a) \bmod N, \tag{2.2}$$

$$w := (w^2 + a) \bmod N,$$

$$w := (w^2 + a) \bmod N,$$

where the w iteration happens intentionally twice, so that w values amount to the x_{2i}. Thus the memory usage is about as low as can be for any factorization method. One need not even perform a gcd operation on every iteration. Instead, simply accumulating the product of many differences $(w - x)$, seldom taking one gcd of the accumulation with N, is the common practice. Various authors [Brent 1980][Montgomery 1987] have reduced the necessary arithmetic, for example by noting other ways to detect collision and stating simplifications of the iteration algebra, but it remains that the (heuristic) expected time behaves as $\sqrt{p}$, in the manner of relation (1.1).

There are two interesting ways to look at the underlying mechanism of the Pollard-rho method. One we shall call the "statistical picture" and the other the "algebraic picture." In the statistical picture we think of the sequence $\{x_i : i = 1, ..., n\}$ as somehow random modulo a hidden factor p of N. How far should this sequence be taken before it replicates a previous value? Let us estimate crudely the probability that a random sequence traversing a set of P elements has its first collision on the n-th iterate. Starting with random x_0, the probability that x_1 misses x_0 is $(P-1)/P$, the probability x_2 collides with neither x_0 nor x_1 is $(P-2)/P$ and so on, so that the first collision happens for x_n with probability:

$$(1 - \frac{1}{P})(1 - \frac{2}{P})...(1 - \frac{n-1}{P})\frac{n}{P}$$

which we further approximate as a particular, Poisson density function

$$f(n) = \frac{n}{P} e^{-n^2/(2P)},$$

whose integral over $n \in (0, \infty)$ is in fact unity. Crude as this line of argument may be, it explains well the basic statistical measures encountered in actual Pollard-rho applications. For example, we find the expected number of iterates required for a collision to be:

$$< n > \sim \int_0^\infty n f(n) dn = \sqrt{\frac{\pi P}{2}},$$

Now as to the comparison of the effective set size P with the prime p, we have, at least roughly speaking, $P \sim p$. There is a delicate issue here—a paradox, if you will—that runs as follows. Because every element in our set (save perhaps for x_0) is a number of the form (quadratic residue + a) modulo p, it would appear that $P = p/2$ is a good choice for the effective cardinality of the set. But as explained by [Brent and Pollard 1981], this notion is faulty, in that the empirical expectation seems to be $< n > \ \sqrt{\pi p/2}$ for the collision index (i of the first $x_i = x_{j<i}$). To paraphrase [Pollard 1999], the iterative map behaves as if it acts on a set of the full p elements. More precise heuristics are also presented in [Cohen 1993], who gives for example the expected length of the actual *period* of the $\{x_i\}$ sequence to be $< n > /2 \sim \sqrt{\pi p/8}$; yet in that work the paradox is not addressed directly. Note that the period is not the same as the expected collision time; for one thing, the collision happens after an initial "pretail" and then a full period—hence the origins of the algorithm label "rho" whose very symbol shape reminds one of a typical collision scenario. Regardless of which analysis is used and what is the resolution of the aforementioned paradox, the established Pollard-rho heuristic is that the expectation of necessary iterations to discover p is: $< n > \sim c\sqrt{p}$ for constant c.

Now to the algebraic picture, which has been analyzed before in such treatments as [Riesel 1994]. Observe the chain of factors that accrues from *every* index k of the x_{2k} iterates:

$$\begin{aligned} x_{2k} - x_k &= x_{2k-1}^2 + a - x_{k-1}^2 - a = x_{2k-1}^2 - x_{k-1}^2 \\ &= (x_{2k-1} + x_{k-1})(x_{2k-1} - x_{k-1}) \\ &= (x_{2k-1} + x_{k-1})(x_{2k-2} + x_{k-2})(x_{2k-3} + x_{k-3})... \end{aligned} \tag{2.3}$$

and we conclude that the difference $x_{2k} - x_k$ is possessed of about k algebraic factors. This means that if we run the index k from 1 through some n, we accumulate about $n^2/2$ algebraic factors. Though the rest of this line of argument is fraught with complications—such as log log factors in the theory of the number of prime factors of a number, not to mention the interference between algebraic factors—it is reasonable that $O(n^2)$ "random" factors have a fair chance of involving p, when n is $O(\sqrt{p})$. Though less precise even than the already heuristic statistical picture, this algebraic picture lends itself easily to considerations of parallelization.

The important special instance when one has foreknowledge of $p \equiv 1 (\mathrm{mod}\ 2K)$ has interesting interpretations in the statistical and algebraic pictures. Statistically, the iteration:

$$x := (x^{2K} + a) \bmod p$$

runs over a restricted set, since for the class of p in question the set of $2K$-th powers has cardinality $P = (p-1)/(2K)$. Thus an operation reduction by $1/\sqrt{K}$ might be expected to appear in the relevant expectation formulae. Note however, that this thinking leads to the aforementioned paradox in the case of $K = 1$, and we recall the discovery of [Brent and Pollard 1981] that the better reduction factor is $1/\sqrt{2K-1}$. On the other heuristic hand, that is in the algebraic picture, we have

$$x_{2i} - x_i = x_{2i-1}^{2K} - x_{i-1}^{2K}$$

and the right-hand side can be expected to have amplified probability of containing a factor p, because in the field $F_p[X, Y]$ the binomial $X^{2K} - Y^{2K}$ has factors $X - g^\gamma Y$, where g is a primitive $2K$-th root of unity. For either picture, the heuristic reduction in total time is (avoiding the more correct $2K-1$ factor for the moment) $O(\log_2 K/\sqrt{K})$, the logarithmic factor for the extra powering inherent to the $2K$-power iteration. Incidentally, even if one uses iteration $x := x^{2K} + a$ without any known caveat on the hidden p, it is an established heuristic [Montgomery 1987][Brent and Pollard 1981], and it can be inferred roughly from either of our heuristic pictures, that the reduction factor in expected number of ring operations is:

$$1/\sqrt{\gcd(p-1, 2K) - 1},$$

where here we have given what is believed to be the correct radicand as a function of K.

3. Parallelization

Let us use the statistical picture to infer the observation of [Brent 1990], that the use of m machines in parallel—if running independent (in both iteration polynomial and initial seeds) Pollard-rho processes—reduces the expected factoring time only by $\sqrt{m}$. Denote by $x_i^{(k)}$ the i-th iterate on machine k of m machines. For the moment, we assume that the k-th machine uses an iteration $x := x^2 + a_k$, with the a_k chosen independently (except $\neq 0, -2$ say); and also each machine is independently seeded with an $x_0^{(k)}$. Along the lines of the statistical picture of the previous section, the probability that none of the m machines has, upon its i-th iterate, a collision $x_i^{(k)} = x_j^{(k)}$ for any $j < i$ would be

$$(1 - \frac{i}{P})^m,$$

and we arrive at an approximate probability density

$$f(n) = \frac{2mn}{P} e^{-mn^2/(2P)} \tag{3.1}$$

for the first collision on some machine to happen at iterate n. This leads immediately to the expectation

$$< n > \sim \sqrt{\frac{\pi p}{2m}},$$

showing the disappointing reduction by only $1/\sqrt{m}$.

From the algebraic picture we can deduce the same heuristic for this weak parallelism. Since we have algebraic factors as in (2.3) for each of the machines, a total of $O(mn^2/2)$ factors are generated. Thus the heuristic is that $n \sim \sqrt{p/m}$ steps will be required to discover p.

This having been said about noncoupled machines, let us consider a more elaborate construct from the algebraic picture, and hope that stronger machine coupling will allow the construct to be efficiently evaluated. At this juncture it is important to note that we shall have to use, in the following parallel construction, the *same* a-parameter across machines. Later in Section 5 we bring up the difficult issue of whether said parameter can be thus frozen.

The new parallel model on which we shall base the new heuristic has, at its core, the correlation product

$$Q = \prod_{i=1}^{n} \prod_{k=0}^{m-1} \prod_{j=0}^{m-1} (x_{2i}^{(k)} - x_i^{(j)}). \tag{3.2}$$

Now assume that each machine has the *same* iteration parameter a, yet the initial seedings $\{x_0^{(k)} : k = 0...m-1\}$ are independently random. We should then have, in the manner of the formal expansion (2.3), $O(m^2n^2)$ algebraic factors comprising Q, and thus the n required to discover p should be $O(\frac{\sqrt{p}}{m})$. From the statistical picture, the same heuristic obtains, as the probability density for the first collision *amongst* machines is just (3.1) but with $m \to m^2$.

The reduction by $1/m$ of the expected n is fine and good, but what about the time to evaluate the elaborate product (3.2)? An answer is, the evaluation of product Q takes, in an appropriate asymptotic sense, just a little longer than the calculation of the usual, basic rho sequence through index $2n$. In particuler, let us say for convenience m divides n, whence machine μ of m machines can compute the partial correlation product

$$Q_\mu = \prod_{i=\mu n/m}^{(\mu+1)n/m-1} \prod_{k=0}^{m-1} \prod_{j=0}^{m-1} (x_{2i}^{(k)} - x_i^{(j)}) \tag{3.3}$$

in just n/m applications of a fast polynomial evaluation algorithm. In fact, machine μ can evaluate the polynomial

$$q_i(x) = \prod_{k=0}^{m-1} (x_{2i}^{(k)} - x)$$

at the points $x \in \{x_i^{(j)} : j = 0, ..., m-1\}$ in $O(m \log^2 m)$ ring operations [Montgomery 1992][Crandall and Pomerance 1999]. Thus the total number of ring operations, on the μ-th machine, is

$$O(\frac{n}{m} m \log^2 m) = O(n \log^2 m).$$

This basic idea, together with the observation that all other calculations, such as accumulation of the actual product in (3.3), are $O(n)$ ring operations, leads to the heuristic estimate (1.3) of total *time* to discover p, with the effective reduction factor being $O(\log^2 m/m)$.

Subject as usual to the validity of heurism, the advantages of using iteration $x := (x^{2K} + a) \bmod N$ when it be given that a prime divisor p of N is $1 (\bmod\ 2K)$, or at least has a fairly large $\gcd(p-1, 2K)$, likewise apply in this new parallel mode. That is to say, each machine would use the degree-$(2K)$ iteration, with a absolutely fixed across machines, but $x_0^{(k)}$ random on the k-th machine as before. With all of these considerations in place, the total time to discover a factor with our parallel algorithm should follow the heuristic estimate:

$$O\left(\tau_N \ \frac{\sqrt{p}}{m} \ \frac{1 + \log K + \log^2 m}{\sqrt{\gcd(p-1, 2K) - 1}}\right). \tag{3.4}$$

We are now positioned to summarize a parallelization algorithm, as follows:

Algorithm for parallelization of Pollard-rho

0) We are given N to be factored, and m machines with which to do this. We assume a number K which will determine the degree (as $2K$) of the Pollard iteration, knowing that a statistical gain from large $\gcd(p-1, 2K)$ is possible. For example, if it be known that a hidden odd prime factor p of N must have $p \equiv 1 (\bmod\ 2K')$, then K should be a multiple of K'.

1) Choose a trial n, being a multiple of m, noting that primes up to a bound $p \sim n^2 m^2 (\gcd(p-1, 2K) - 1)$ have a good chance of being discovered via this algorithm, and under the constraint that acyclic convolution of length-m sequences modulo N be possible on each machine.

2) For each machine k of m, establish an initial independent random seed $x_0^{(k)}$ and set $w_0^{(k)}$ equal to that same respective seed. Then choose a parameter a which is the *same* across all machines.

3) For $i \in [1, ..., n]$, iterate on each machine k of the m machines simultaneously:

$$x_i^{(k)} = ((x_{i-1}^{(k)})^{2K} + a) \bmod N,$$

$$w = ((w_{i-1}^{(k)})^{2K} + a) \bmod N,$$

$$w_i^{(k)} = (w^{2K} + a) \bmod N,$$

so that $w_i^{(k)} = x_{2i}^{(k)}$ always.

4) On each machine μ of the m machines, evaluate modulo N the product

$$Q_\mu = \prod_{i=\mu n/m}^{(\mu+1)n/m-1} \prod_{k=0}^{m-1} \prod_{j=0}^{m-1} (w_i^{(k)} - x_i^{(j)})$$

via n/m polynomial evaluations of degree m each, so that this step requires $O(n \log^2 m)$ ring operations on each machine.

5) On each machine, take the $\gcd(Q_\mu, N)$ and report any nontrivial factor arising from any machine.

6) On failure of step (5), go to (1) to increase n, or continue iterations in (3), or effect some other means of effectively modifying n and/or parameters such as a, K.

4. Open questions, Fermat numbers, quantum computation

First we remind ourselves that little if anything has been proven so far, and mention some open issues in regard to the kind of heuristics we have so brazenly invoked. The "final" heuristic relation (3.4) for the parallel scheme might be viewed with suspicion, along the following lines. As J. M. Pollard himself has pointed out to the present author, the assumption of fixed a-parameter leads to certain difficulties in the heuristic analysis. In the scenario of weak parallelism as enunciated in the very opening of Section 3 (i.e., independent seeds and independent a-parameters), different machines can be expected to have widely varying iterative cycles in force. But if a is fixed across machines, there is a very small expected number (actually $O(\log p)$) of cycles at work [Pollard 1999]. Pollard cites the small example $p = 257$, $a = -1$, for which there are only three periods 12, 7, 2 (and we note that the number of iterates to discovery is the least multiple of said period that is not less than the tail). For example, the initial value $x_0 = 0$ leads to the 2-cycle $\{0, -1\}$, while the initial value $x_0 = 6$ gives rise to the sequence

$$\{6, 35, 196, 122, 234, 14, 195, 245, 143, 145, 207, 186, 157, 233, 61, 122\}$$

whose period—after a tail of 3 iterates—is one of the possible ones cited, namely 12. The component counts for the possible cycles decompose respectively as $142 + 98 + 17 = 257$. For example, the probability that an initial seed x_0 will eventually exhibit the period 12 is $142/257$. Already, in this small example with three possible periods, it is not clear until one does some detailed combinatorics how the *expected* minimum period across m machines decays; although the expected period will of course be monotonic nondecreasing in m.

What happens in practice is that when p and a fixed a are chosen, with larger and larger machine counts m emulated (say m is much less than say $\sqrt{p}$ itself) the tendency for some machine to pick up one of the smaller possible periods naturally increases. However in such experiments with fixed a one does witness the aforementioned quantization of cycle periods. In order to get the theoretically correct expectation of minimum period for fixed a across machines therefore, one would have to know something about the relatively small set of $O(\log p)$ periods, and furthermore know how random initial seeds deposit themselves into the respective cycles. We repeat here the reason that it is not so hard to infer the $1/\sqrt{m}$ acceleration in the Brent scenario—in which each machine has a differant a_k–namely, widely varying periods will be available to different machines; in effect the space of possible cycle periods will be widely sampled. Instead of $O(\log p)$ possible cycles, there would be perhaps as many as $O(m \log p)$ cycles available on the parallel network.

Faced with such intricate statistical issues, one might be suspicious of the full parallel scheme. The present author carried out the following experiment on the Q product (3.2) for which a is fixed across machines. Choosing:

$p = 2^{31} - 1$,

$n = 250$ iterations,

$m = 250$ machines,

and one random a parameter for every full parallel run, the empirical probability that Q contains the factor p was a healthy 0.61. This was based on one hundred random choices of a and a complete evaluation of Q for each choice of a. Now for these choices we have $n^2m^2 \sim p$, so this brief experiment supports—at least in order-of-magnitude fashion—the parallel heuristic based on the Q product. So even in the face of the the complicated quantization of cycle periods when a is fixed across machines, this experiment is encouraging.

Next, regardless of the open issues we turn to speculations stemming from the heuristic (3.4). Let us attempt some estimates of actual run time for hypothetical factorizations. We shall use formula (3.4) with the big-O cavalierly stripped away, using $\log_2$ for any logarithms (actually such is a realistic substitution), with the effective coefficients of the $\log K$, $\log^2 m$ terms both unity (actually a reasonable guess in actual practice), and a time for one multiplication modulo $N = F_{18}$ say, to be 0.1 seconds (today this is a realistic estimate on such a large-integer operation, in fact multiply times an order of magnitude lower than this have been reported for certain machine architectures [Mayer 1999]). Then to discover the 23-digit McIntosh-Tardif factor mentioned in Section 1 we need an (operation $\rightarrow$ time) equivalent count of

$$\sqrt{p} \quad\rightarrow\quad 1000 \text{ years}$$

using basic, unadorned Pollard-rho. Using the fact that any factor of F_{18} is of the form $p \equiv 1 (\mathrm{mod}\ 2^{20})$ we set $2K = 2^{20}$ and note that the better iteration $x := x^{2K} + a$, for which the estimate of (operations $\rightarrow$ time) to discover the factor is:

$$\sqrt{p}\,\frac{\log_2 2K}{\sqrt{2K-1}} \quad\rightarrow\quad 17 \text{ years.}$$

On the other hand, we assess the impact of parallelism as follows. In this age of massive volunteer network computations, let us be generous and take $m = 10000$ machines. Now in the "weak" parallel mode, where there is essentially no intermachine communication, the (operation $\rightarrow$ time) estimate comes out as:

$$\sqrt{p}\,\frac{1}{\sqrt{m}}\frac{\log_2 2K}{\sqrt{2K-1}} \quad\rightarrow\quad 2 \text{ months.}$$

The new parallel scheme though has estimate:

$$\frac{\sqrt{p}}{m\sqrt{2K-1}}(\log_2 2K + \log_2^2 m) \quad\rightarrow\quad 4 \text{ days.}$$

As with this example, it is generally true that hundreds or thousands of machines would be required to bring the new parallel method beyond the efficiency of the weak one, due to the $\log^2 m$ factor. But let us at least establish that the machine parameters are not too absurd in such regions of advantage. From the algorithm of the previous section, we see that for n parallel iterates, per-machine, of Pollard-rho sequences the approximate relation

$$p \sim n^2m^2K$$

is expected. This means that each of the $m = 10000$ machines would only do about $n \sim 40000$ iterations, or on the order 10^6 multiplies modulo F_{18}. So the number of iterates in step (3) of the algorithm is entirely realistic, and along with the time for product evaluation in step (4), is on the order of days. It may be possible to bring overall times down yet further by allowing the parallel machines to cooperate even more in the evaluation of the products. (For example it is unknown to the present author whether a degree-m polynomial can be evaluated at m points via m coupled machines in $O(m^\epsilon)$ ring operations per machine, yet this may well be possible.)

As it stands the new algorithm requires extensive sharing of stored iterates. Let us see if such storage is even possible for our example of F_{18}. Each machine must be capable of fast polynomial evaluation for degree m, so via convolution techniques (such as Nussbaumer convolution with memory optimizations [Crandall 1995]) that would involve a small multiple of the memory required for 10^4 copies of F_{18} (a 32-kbyte entity). This comes out to about 700 megabytes per machine, which although stultifying is not out of the question even for personal computers of today. It is also the case that some kind of memory pool must exist for all the stored iterates from step (3) of the algorithm. The total memory required is about $2n$ copies of size-F_{18} residues on each machine μ, where we might assume that the sequence $\{x_i^{(\mu)}\}$ is actually stored. In our example that is about 7 gigabytes per machine, again unfortunate but not out of the question. An interesting line of future research is to determine whether alterations of the precise of order of doing things between algorithm steps (3) and (4) will save memory. For example, when machine $\mu = 0$ does its n/m polynomial evaluations each of degree m, at m points, perhaps that effort should be shared amongst all machines, and an accumulation (or gcd) operation performed, so that the indices $i \in [0, n/m - 1]$ once processed may theretofore be ignored. If such memory-reduction schema can be made to work, then the dominant memory is that of the polynomial evaluation itself, or as we have said, $O(m)$ copies of size-N residues, per machine.

Another interesting line of possible optimization is the following. Observe that the choice of K can be arbitrary, on the idea that a time reduction of about $1/(\gcd(p-1, 2K)-1)^{1/2}$ results. There is nothing preventing one from trying various K values at the start of the algorithm. It is not yet known whether using different K values *per machine* in this new parallel scenario affords any advantage. It would be interesting to analyze this question from both statistical and algebraic pictures. But since the McIntosh-Tardif factor is

$$81274690703860512587777 = 1 + 2^{23} \cdot 29 \cdot 293 \cdot 1259 \cdot 905678539,$$

any machine that happens to have an extra factor 2^3 or even $2^3 \cdot 29$ (beyond the known efficiency factor $2K' = 2^{20}$) contained in its $2K$ value will likely contribute somewhat more powerfully than otherwise. Indeed, the appearance of the $(\log K + \log^2 m)$ factor in the overall time estimate (3.4) shows that for large numbers m of machines, one will usually obtain a definite acceleration from a lucky K value, and in any case one may use various small K values with relative impunity.

Aside from the issue of whether it makes sense to "blindly" adopt various K accelerators, and whether to do this on a per-machine basis, there is the question of the Pollard iteration itself, and how its effective traversal set can be restricted. Though linear iterations $x := ax + b$ are known to be unsatisfactory (e.g. they might have periods of length $O(p)$), it would be good to know the effective periods for $x := a \bullet x + b$ where $\bullet$ is an elliptic multiplication on an elliptic curve, with x now being a point. Such an elliptic iteration was once suggested by V. Miller, in regard to pseudorandom number generation. In fact, [Miller 1999] points out that the period of such an elliptic generator can again be p (good for cryptography, bad for rho-factoring), depending on the group structure at hand; i.e. whether the elliptic group is cyclic. Thus the elliptic generator—with each step involving expensive elliptic arithmetic—may be of no use in rho-based factoring; yet the elliptic generator should still be studied completely in this regard. Then, too, it would be of interest to apply various of the ideas herein to the Pollard-$(p-1)$ method. For the moment, we note that [Montgomery and Silverman 1990][Montgomery 1992] have shown the importance of polynomial-evaluation acceleration to Pollard-$(p-1)$ and ECM, respectively. It should be remarked that the aforementioned McIntosh-Tardif factor, as well as other ECM-based discoveries such as the factor

$$3603109844542291969 = 1 + 2^{19} \cdot 3 \cdot 4363 \cdot 525050549$$

of F_{13}, could be found rather quickly with Pollard-$(p-1)$ with suitable second stage, perhaps in the "FFT" style of Montgomery and Silverman (to pick out the factor 525050549). Certainly the corresponding time estimates [Mayer 1999] show that such an effort would clearly be more lucrative than single-machine Pollard-rho. But this comparison is not at all so one-sided in a parallel-rho scenario for which some lucky machine is performing an iteration: $x := x^{2^{19} \cdot 3 \cdot 4363} + a$, say. For that machine should only require about $n \sim 10^5$ iterates all by itself, to discover the given 19-digit factor of F_{13}, and this would only take a matter of hours. Again this raises the open question of whether there be genuine gain in having machines carry out separate Pollard iterations. One may look longingly at the older results of [Brent 1985] on ECM-rho comparisons—which results imply a kind of crossover at about 11 digit factors—and wonder what higher crossover might accrue from both parallelism and use of hidden K values.

Now, if the present author may speculate in unrestrained fashion: whether or not further reduction in the complexity estimate (3.4) can be effected in future, there is the issue of quantum computation. It is now known that factorization via quantum Turing machine (QTM) can proceed, at least in principle, at unprecedented speed, as in the Shor factoring algorithm [Ekert and Jozsa 1996]. In such an approach, quantum cells using natural quantum interference would be used to perform gargantuan fast Fourier transforms (FFTs), and so look for a certain kind of periodicity of powers modulo an N to be factored. By the same token, we expect the new parallel scheme for Pollard-rho to have a reasonably direct analogue in the QTM world. After all, the notions of parallelism, inter-processor communication, and large, fast transforms are all present. In fact, transforms tend to figure into the best polynomial evaluation schemes. If there be a valid analogue of the basic Pollard-rho on QTMs, we can say now that there should also be a *parallel* analogue. If yet more speculation can be extended from this already speculative posture, let us be truly generous and think of $m = 10^{24}$ (one "mole") of quantum cells. Not to stretch the notion too far, but a nanotechnological society might one day view such a cell count as, well, commonplace. Then for this m,

$$\frac{\log_2^2 m}{m} \sim 10^{-20},$$

so that formula (3.4) allows 70-digit factors p of N to be obtained in about 10^{15} ring operations in Z_N—certainly a feasible operation count by comparison with many of the deeper computational successes of the present era of conventional TMs. Needless to say, 70-digit extractions are right about at the current state of the factoring art (via the number field sieve (NFS), say). Of course one could respond that, by the time a quantum-nano technology is upon us, other dominant factorization methods will have been correspondingly accelerated to unprecedented performance levels. But to this we can say, Pollard-rho is so very simple, it may be one of the first implementations in any really new technology. After all, it has already been, once for a time the best available method for certain classes of N. True, there is polynomial evaluation of the Q products indicated for our stated algorithm, but for all this author knows, a future machine would be better off doing brute-force $O(m^2)$ product evaluations, and *that* is a manifestly simple procedure. Because of allowed special K values, something like the elusive F_{14} or one of its relatives might finally be demolished in this way (it is an interesting exercise to estimate how many new Fermat factors would accrue at a 70-digit search limit). It would therefore seem that, even though more dominant factorization schemes have migrated to the forefront over the last two decades, the last chapter in the story of the Pollard-rho method is not yet written.

5. Acknowledgments

The author is indebted to M. Wiener for commentary and insights relevant to the theoretical issues herein, to R. Brent for remarks pertaining to the manuscript, and to C. Pomerance for his particular insight into Pollard-rho and exponential factoring algorithms in general. J. M. Pollard himself corrected several misconceptions on the present author's part, thereby rendering this a (relatively) more professional paper.

6. References

Brent R 1980, "An improved Monte Carlo facorization algorithm," *BIT*, 20, 176-184.

Brent R 1985, "Some integer factorization algorithms using elliptic curves," manuscript, ftp://pub/Documents/techpapers/Richard.Brent/rpb102.dvi.gz

Brent R 1990, "Parallel algorithms for integer facorization," in Loxton J H Ed., *Number Theory and Cryptography*, *London Math. Soc. Lect. Note Series*, Cambridge University Press, Ca.b., 154, 26-37

Brent R and Pollard J 1981, "Factorization of the eight Fermat number," *Math. Comp.*, 36, 627-630.

Brent R, Crandall R, Dilcher K, and van Halewyn C 1999, "New Factors of Fermat Numbers," *Math. Comp.*, to appear

Cohen H 1993, *A course in computational algebraic number theory*, Graduate Texts in Math. 138, Springer-Verlag, Berlin Heidelberg New York

Crandall R 1995, *Topics in Advanced Scientific Computation*, Springer-Verlag, New York.

Crandall R and Pomerance C 1999, "Prime numbers: a computational perspective," Springer-Verlag (mansuscript t.b.p.)

Ekert A and Jozsa R 1996, "Quantum computation and Shor's factoring algorithm," *Rev. Mod. Phys.* 68, 3, 733-753

Mayer E 1999, private communication

Miller V 1999, private communication

Montgomery P L 1987, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Math. Comp.*, 48, 177, 243-264

Montgomery P L 1992," An FFT Extension of the Elliptic Curve Method of Factorization," Ph. D. Dissertation, University of California, Los Angeles

Montgomery P and Silverman R 1990, Math. Comp., 54, 839-854.

Pollard J M 1975, "A Monte Carlo method for factorization," *BIT*, 15, 331-334.

Pollard J M 1999, private communication.

Riesel H 1994, *Prime Numbers and Computer Methods for Factorization*, Birkhauser, Boston.

van Oorschot P C and Wiener M J 1999, "Parallel Collision Search with Cryptanalytic Applications," *J. Cryptology*, 12, 1-28.

Wiener M 1999 1998, private communication