MPQS with Three Large Primes

Paul Leyland[1], Arjen Lenstra[2], Bruce Dodson[3],
Alec Muffett[4], and Sam Wagstaff[5]

[1] Microsoft Research Ltd, 7 JJ Thomson Avenue, Cambridge, CB3 0FB, UK,
pleyland@microsoft.com
[2] Citibank, 1 North Gate Road, Mendham, NJ 07945-3104, USA,
arjen.lenstra@citicorp.com
[3] Lehigh University, Bethlehem, PA 18015, USA,
bad0@Lehigh.edu
[4] Sun Microsystems Professional Services, Riverside Way, Watchmoor Park,
Camberley, GU15 3YL, UK, alec.muffett@uk.sun.com
[5] CERIAS and Department of Computer Sciences, Purdue University,
West Lafayette, IN 47907-1315, USA, ssw@cs.purdue.edu

Abstract. We report the factorization of a 135-digit integer by the three-large-prime variation of the multiple polynomial quadratic sieve, the largest factorization ever performed with MPQS. Previous workers [8][12] had suggested that using more than two large primes would be counterproductive, because of the greatly increased number of false reports from the sievers. We show this not to be the case: for our implementation and for integers of about 135 digits using three primes is almost twice as efficient as using only two. The greatest gain in efficiency comes from a sudden growth in the number of cycles arising from relations which contain three large primes. This effect, which more than compensates for the false reports, was not anticipated by the authors of [8][12]. We characterize the various types of cycles present, and give a semi-quantitative description of their rather mysterious behaviour.

1 Introduction

The use of large primes in the Multiple Polynomial Quadratic Sieve (MPQS) has been suggested many times. Earlier studies [12][15] suggested that using one large prime is always better than using none, and that the double large prime variation (often called PPMPQS) is more efficient than using fewer large primes when factoring integers with more than about 80 decimal digits. For N near 10^{100} these studies showed that the double-prime version of MPQS is 2 to 2.5 times faster than the single-prime variant. A spectacular example of a PPMPQS factorization was the record-breaking factorization of the 129-digit Scientific American RSA challenge RSA-129 in 1993-4, reported in [2].

In [8][12] it was suggested that using three large primes (as opposed to just two) would be counterproductive because of the very large number of false reports. In this paper we call the triple large prime version of the multiple polynomial quadratic sieve "TMPQS" for conciseness. Nevertheless, proposals have

been made that three or more primes may be effective, even though they are unlikely to lead to major performance increases. Experience with the NFS has shown that the use of more than two large primes when finding cycles amongst the relations found by the siever gives many more cycles than would be obtained with only two primes [9]. We expected that similar behaviour may also occur with TMPQS, and that it may compensate for the extra false reports. A few preliminary and small-scale computations by the first author suggested that the use of 3 large primes was certainly no worse than 2 large primes when N has about 100 to 105 digits and may be slightly better.

To put this hypothesis to the test, we factored the 135-digit cofactor of $2^{803} - 2^{402} + 1$, also known as 2,1606L.c135 in the well-known Cunningham newsletters. This particular number was chosen because it is: somewhat larger than the previous record-breaking MPQS factorization but close enough in size to RSA-129 that we may make reasonably accurate comparisons with the earlier computation; comparable with recent large factorizations using the General Number Field Sieve; not easily factored with the Special NFS. It should be noted, however, that the GNFS would have been much faster than the computation we decided to embark upon. We factored 2,1606L.c135 solely to satisfy our curiosity with respect to the relative speed of TMPQS and PPMPQS and to get more insight in the cycle behaviour. Setting the current record for a factorization with the Quadratic Sieve algorithm was entirely incidental...

In Section 2 we describe our implementation of TMPQS, concentrating almost exclusively on how it differs from previous implementations of MPQS. Section 3 provides a detailed description of the cycle behaviour we found as the computation progressed, and gives a semi-quantitative analysis of the results. In Section 4, we compare the performance of PPMPQS and TMPQS, taking care to minimize the effects of other differences between the two computations — to do this we used a machine which is now very nearly a museum piece.

2 The Multiple Polynomial Quadratic Sieve

The multiple polynomial quadratic sieve (MPQS) algorithm has been described many times in easily accessible literature. For this reason, we do not describe the complete algorithm in any detail, but rather concentrate on the differences between our implementation and its predecessors. The major difference is that we allowed for up to three large primes in a relation. As a consequence, we had to re-write the relations processing software, and to implement quite different cycle counting and cycle finding algorithms.

The particular implementation of the siever that the first four authors used was almost identical to the 'factoring by email' version described in [11], which should be consulted if greater detail is required. Even though other implementations of MPQS are readily available, some of them perhaps more efficient than ours, using this old implementation had an important benefit: we were able to compare our performance figures directly to those obtained in previous large-

scale factorizations, such as that of the 129-digit RSA challenge number [2] which was also based on the software from [11].

The fifth author replaced the reporting code in his implementation of a Self-Initializing Quadratic Sieve siever [4] with the one in the TMPQS siever used by the other four authors to make it generate relations, including relations with three large primes, compatible with the others. Because of the nature of the SIQS, the relations produced by this siever contained markedly more small primes, but the large-prime statistics within the relations having one, two and three large primes, and the relative proportions of the three types of relations, was closely similar.

The MPQS algorithm to factor an integer N can be conveniently split into five phases: selection of parameters; sieving; counting and finding cycles within the partial relations; linear algebra and combination of relations to find a factorization.

2.1 Parameter Selection

The parameter selection phase chooses a factor base of B_1 small primes which are quadratic residues modulo kN where k is a small integer multiplier chosen to maximize the number of very small primes in the factor base. In this stage, we also select the number n of large primes permissible in a relation ($n = 3$ in this work and $n = 1$ or $n = 2$ in earlier factorizations); the maximum value B_2 of an acceptable large prime; and the range over which each quadratic polynomial is sieved for smooth quadratic residues — i.e., values of the polynomial which factor entirely into primes in the factor base and at most n prime factors each bounded by B_2. Optimal selection of these parameters is still something of an art, not least because they depend somewhat on the hardware and software implementation, but a great deal of experience has been acquired over the past decade or so when factoring integers in the range $10^{50} < N < 10^{130}$. To factor 2,1606L.c135 we chose a factor base containing 550 000 primes (so the largest prime in the factor base was 17 157 953) and allowed up to three large primes not exceeding 2^{30}. The multiplier $k = 1$ was found to be optimal; the same value $k = 1$ was used in the factorization of RSA-129.

The size of the factor base was chosen in a rather ad hoc manner. Several different choices were made with values between 400 000 and 750 000 and sieving experiments made for each. In the experiments we tried a number of different values for the siever reporting threshold for each factor base size. We attempted to maximize the rate at which relations were found while simultaneously getting a reasonably high fraction of relations containing three large primes. The value eventually chosen may not be optimal, but it appears not to be too bad. It is, however, substantially smaller than would be chosen for a PPMPQS factorization of a 135-digit integer, which would be closer to 900 000. We can justify a smaller size to some extent by noting that the larger the factor base, the longer is the time spent sieving and trial dividing, but the smaller is the expected size of the unfactored remainder. Allowing an extra large prime in a relation compensates somewhat for the larger remainders which must be factored.

We experimented with several values for the sieving range. The siever used by most of us was tested with values lying between 17 million and 100 million, attempting to maximize the rate at which relations were produced. The rate was only slightly dependent on this value but was somewhat higher at the lower end of the range. The bulk of the computation used a value of 17 158 000. The SIQS siever employed by the fifth author used a sieving range of only 2 000 000 because the SIQS algorithm initializes quadratic polynomials so efficiently. For comparison, the factorization of RSA-129 used a factor base of 524 338 primes, two large primes each less than 2^{30} and a sieving range of 163 336 000 values per polynomial.

It is not a surprise that the best sieving range turned out to be much smaller than the one used for RSA129, despite the fact that the same sieving software was used with almost the same factor base size. Namely, it is well known that the ratio of the time spent on the polynomial selection (and root finding) and sieving should be about 1:3. Polynomials and their roots are found about equally fast for the two numbers, so about the same time should be spent on sieving. But for our current TMPQS factorization many more reports have to be processed per sieve range (i.e., per polynomial) due to the lower report bound. Furthermore, the average processing time per report is substantially larger than for PPMPQS, because substantially larger cofactors have to be tested for primality and factored (if not prime). Thus, to keep the ratio constant, the sieve length has to be shorter. It should be noted, however, that we did not set out to keep the ratio constant, but that we simply looked for the best sieve length — the above is just an explanation why the best sieve length was found to be much smaller than before. Note that a shorter sieve also leads to, on average, smaller polynomial values. All these effects have been analyzed in [1].

2.2 Sieving

Again, we refer to the extensive literature on the sieving phase of the MPQS. The only difference between the siever used by most of us and that used to factor RSA-129 is that we attempted to factor quadratic residues not exceeding 2^{90} after primes in the factor base had been divided out. We used a fast pseudoprimality test to reject candidates greater than B_2 that were not recognized as composites, we used ECM to factor those which were, and rejected those for which ECM failed or found a prime factor larger than B_2. Note that per report the pseudoprime test may have to be applied to two different numbers (one around 90, and one around 60 bits), and that ECM may have to be used to factor both those numbers (if composite). Thus, testing candidate quadratic residues is rather more expensive in TMPQS than PPMPQS. The output of the siever was a series of relations, consisting of the prime factorization of a square modulo N and the corresponding square root. As in [2], we denote a relation as a *ful*, *par* and *ppr* according to whether it contains zero, one or two large primes. In addition, we have *tpr* relations which contain three large primes.

As in [2], the sieving process was distributed over many client machines. Each client sieved a distinct range of polynomials and wrote its relations to a file. The

output files were transferred to a central machine in Cambridge by a variety of ad hoc methods, including electronic mail, FTP, shared file systems, and by physical transport of diskettes and laptop computers. The central machine performed sanity checks on newly arrived data, in particular ensuring that the purported prime factorization was indeed that of the square of the purported square root. Valid relations were then written to one of four files, according to whether they were *ful*, *par*, *ppr* or *tpr* relations. The relations were rewritten, if necessary, to ensure that any large primes appeared in numerical order within a relation. For convenience in the following stages, a *par* relation had two copies of **1** prepended to its large prime, and a *ppr* had a single **1** prepended.

Following this stage, the newly-collected relations were sorted and merged into four accumulation files, and duplicates discarded. (Although duplicates should not have appeared, as clients were supposedly sieving disjoint ranges, operator errors in starting clients and in gathering output led to a small number of duplicates in practice.)

We began sieving on 10^{th} January 2001 and finished 231 days later on 29^{th} August. By the time we finished sieving, we had accumulated a total of 13 441 627 relations, made up of 62 626 *ful*s, 790 129 *par*s, 4 080 732 *ppr*s and 8 508 140 *tpr*s. The contribution of each author is summarized in Table 1.

Table 1. Number and type of relations found by each contributor

Contributor	*ful*	*par*	*ppr*	*tpr*	Total	%
Dodson	3806	48603	255329	562312	870050	6.47
Lenstra	366	4621	24178	52189	81354	0.61
Leyland	21496	277912	1460826	2958308	4718542	35.10
Muffett	6783	87080	454569	898223	1446655	10.76
Wagstaff	30175	371913	1885830	4037108	6325026	47.06
Total	62626	790129	4080732	8508140	13441627	100.00

2.3 Counting and Finding Cycles

In principle, once enough relations have been output by the sieving phase, linear dependencies in the matrix of exponent vectors modulo 2 could be found by standard techniques of linear algebra, such as Gaussian elimination or block Lanczos. In practice, the matrix would be far too large to be easily dealt with by these methods and we use a preliminary step to reduce the size of the matrix without thereby making it too dense. It has been traditional in MPQS factorizations to use a process of cycle-finding and relation-combining to generate a set of relations in which all large primes are eliminated. An alternative "filtering" approach is often employed to solve the analogous problem in the Number

Field Sieve [5] (but see [3] for an application of filtering in NFS). We followed tradition in that we found cycles and combined relations containing at least one large prime, but the techniques used in [11] are not entirely applicable when more than two large primes are present.

During the sieving phase, it is unnecessary actually to generate the cycles present. However, it is necessary to know how many there are so that progress can be monitored and the sievers stopped once enough cycles have been produced. Accordingly, we wrote both a cycle-counting algorithm and a cycle-finding algorithm. If all that was required from the counting algorithm was a indication of when to stop sieving, we could have used the very simple approach of comparing the number of distinct primes in the relations to the number of relations. However, this tells us nothing about the details of the cycles' characteristics and so we used a more complicated algorithm.

A fundamental observation is that a relation containing a prime p can only form part of a linear dependency if p occurs at least twice in the set of relations. Removing singletons, i.e., those relations containing a p which occurs only once in the entire data set, is the first stage of the cycle finding algorithm. Note that removing a singleton *ppr* or *tpr* relation may generate further singletons from the other prime(s) present in the relation. We must, therefore, repeat this action iteratively until no more singletons are removed. The number of repeats needed gives an indication of the length of the chains present in the graph. More will be said on this in section 3. When no singletons remain, all primes occurring in the surviving relations must occur at least twice and all surviving relations must form part of at least one cycle.

The singleton removal, or "pruning" algorithm is straightforward. Each of the four files containing relations was read and the large primes (including the special value **1**) contained within the relations were inserted into a heap. The contents of the heap was then read in numerical order, and the values **1** and the primes which occurred at least twice (the "useful" primes) were stored in a table. The number of entries in this table was printed so that progress could be monitored. The relations file was then re-opened and those which contained only useful primes were written to a temporary file. On completion of this process all singletons, if any, had been removed. If no removals had taken place, the pruning procedure terminated, otherwise the temporary file became the input file for a succeeding pass of the same procedure.

The cycle-counting algorithm is a simple modification of that used in [11]. Exactly the same basic machinery is used, in that a graph is constructed in which nodes are labelled by useful primes, together with the special node labelled **1** for the *par* and *ppr* relations. An arc is created between nodes p and q if a relation contains the primes p and q. As each relation contains three "primes", p, q and r (which need not be distinct and up to two of which could be the special value **1**) it generates three arcs pq, qr and rp. Note that this is unlike the case described in [11], where a relation contained only two primes and generated a single arc. The eventual output of the algorithm is the number of primes (equal to the number of nodes in the graph) P, the number of disconnected components C,

the number of arcs A and the total number of relations R. In the PPMPQS case, the number of cycles amongst the relations is equal to the number of cycles in the primes graph and is given by $C + R - P$. In our case, however, the number of cycles within the relations (not the primes) was thought to be given by $C + A - P - 2R$.

The curious phrasing "was thought to be" in the previous sentence is a consequence of an unpleasant surprise we received during the writing of the present paper. The method was developed about six years ago and it seemed to work fine, both for simple test data we created and for a number of TMPQS factorizations of integers in the 100 to 105 digit range. Upon closer inspection, the formula used turns out to be incorrect when applied to a particular test case. We are still investigating how to fix this. All we can say at this point is that the formula used gave a close enough approximation to the actual number of cycles, and that all factorizations attempted while using it were successfully completed. Thus, though wrong, the formula cannot be too far from the truth. In particular, in every factorization we performed the number of cycles produced by the cycle-finding algorithm was exactly the same as reported by the cycle-counting algorithm.

Applying the above algorithms to a subset of the data and, especially, the data without the inclusion of *tpr* relations and without anything but the *par*s enabled us to separate out the effects of each type of relation and, in effect, to compare TMPQS simultaneously with the progress of the analogous one-prime and two-prime variations of MPQS.

We counted cycles approximately daily until the sum of the number of cycles and the number of *ful* relations exceeded the size of the factor base, 550 000, when the siever machines were stopped. At this point, we can guarantee that there will be linear dependencies in the matrix of exponent vectors. The final collection of relations produced 494 077 cycles, only 67 543 of which (14%) did not contain at least one *tpr*; there were 62 626 *ful* relations at this point. A detailed account of the behaviour of the growth in the number of cycles as relations were added to the collection is given below.

Once we have determined that there are sufficient cycles to complete the factorization, we must generate them and combine the corresponding relations to form new relations which contain no large primes.

Each relation is read in turn and two hash tables built. (Note, in distinction to earlier phases, we do *not* treat the value **1** as a prime for the purposes of this algorithm.) The first, called `rbp` for "relations-by-primes", contains one record per prime, each record keyed by a prime being a linked list of relations which contain that prime. The other, `pbr`, contains two records per relation that are keyed by the line number of the relation: a linked list of primes appearing in that relation, and a linked list called the "chain", initially empty, to hold relations which are in the process of being formed into a cycle. There is a case which needs to be handled specially: when a *ppr* or *tpr* relation has a large prime, p say, which occurs at least twice. A *ppr* of the form $(\mathbf{1}, p, p)$ is emitted as a cycle of length one and not processed further. Such relations are rare, but about one

in every few million is found to be of this form. A *tpr* which contains two copies of a prime p is entered into the tables as if it were a *par* containing only the third prime. In principle, the third prime could also be p, which would be entered into the hash tables, but no such *tpr*s were discovered in the present work.

Once all the relations have been read and the hash tables built, we begin growing chains of relations until a cycle is formed, which is then emitted as a line containing the line numbers of the relations concerned. Repeated linear sweeps through the `pbr` table are made: if the referenced relation r_0 is a *par*, i.e., its list of primes consists of a single element p, the list of relations containing p is retrieved from the `rbp` table. Each relation r_i in the list, other than r_0, is dealt with in turn. If the relation r_i is a *par*, the pair form a cycle; r_0, r_i and their respective chains (if non-empty) are emitted. Otherwise, the chain of r_0 and r_0 itself is appended to the chain of r_i, and the prime p is deleted from the prime-list in r_i. When all the list of relations containing p has been processed in this manner, the entry keyed by r_0 is deleted from `pbr` and the entry for r_0 keyed by p is deleted from `rbp`.

The procedure described in the preceding paragraph is then repeated until no further changes to the hash tables are made. Only cycles which contain at least one *par* are generated by this procedure[1] but, in practice, all cycles within the relations were found to be of this form and we did not need to implement further phases to process any relations remaining in the tables after these cycles had been emitted.

The final combination algorithm used was exactly that of [11] apart for the trivial modification to multiply up to three large primes per relation and will not be described here. We generated all 494 077 combined relations and selected the 487 424 least dense. Combining the relations in the cycles took nine hours on a 1.3GHz Athlon processor. The longest cycle used contained 215 relations; the longest present had well over a thousand. With the addition of the 62 626 *ful* relations a matrix with 550 000 rows and 550 050 columns was produced. The matrix required 616 megabytes to store in a reasonably compact but printable-ASCII format; the relations from which it was generated required a 1822 megabyte file and came very close to exceeding the maximum size on the file system in use.

With a total of 226 131 280 set bits (an average of 411 per row) this matrix is much denser than usual for an MPQS factorization. We could doubtless have reduced the density of the matrix by sieving further, but had no need to as the matrix was tractable with the resources available to us.

[1] Actually, not all cycles with this form are found by the procedure given. Consider the cycle formed by the set of relations $\{ (\mathbf{1}, \mathbf{1}, p), (\mathbf{1}, q, r), (p, q, r) \}$ where primes p, q and r appear nowhere else in the data. The algorithm as stated reduces the *tpr* to a *ppr*, puts the *par* on its chain and discards p and the *par* from the hash tables. Neither of the remaining relations is a *par* yet together they form a cycle. In practice, cases like this did not occur.

2.4 Linear Algebra

Finding dependencies in the relations matrix was done with the block Lanczos method of [13] and [14]. We performed this computation twice, once on a conventional uniprocessor machine and once on a cluster of workstations. The reason for duplicating this effort is two-fold. First, we were able to make an approximate comparison of the resources used by each implementation. Secondly, we were working to a tight deadline and so having two independent computations running at the same time gave us insurance if one machine were to crash. In both cases, the Lanczos algorithm used 128–bit vectors and took 4324 iterations to find 57 dependencies.

The uniprocessor machine was fitted with a 1.33GHz AMD Athlon processor and had 768 megabytes of memory. The software was compiled with the GCC compiler; RedHat Linux 7.0 was the operating system. The complete computation took 146446 seconds, or a little under 40.7 hours. The amount of active virtual memory reached a maximum of 537 megabytes, but fell to 480 megabytes for the main part of the computation. These figures are well below the size of real memory available and so paging was not a problem.

The parallel implementation ran on a cluster of sixteen machines, each of which contained two 300MHz Pentium-II processors and 384 megabytes of memory. (Note that the memory available on a single cluster node would not have been sufficient to hold the entire data set.) We used the Microsoft Visual C++ compiler, together with the MPIPro multi-processor harness communicating via 100Mbps ethernet; the nodes ran the Windows 2000 operating system. In our computation we used 12 nodes and only one processor per node as the remainder of the cluster was required by other workers. Detailed records of the resources used are available only for the master node, which is the node responsible for all the I/O required for the computation. It took 33808 seconds, or 9.4 hours, for its share of the computation and used 63 megabytes of active memory. The other nodes would have taken about the same cpu time, or very slightly less. One of the slave nodes was observed to be using 53 megabytes of memory. The 10 megabyte difference between these two figures is accounted for by the data structures needed by the master node to co-ordinate the computation as a whole. If we assume the eleven slave nodes each used 53 megabytes, the total memory usage came to 646 megabytes, substantially more than the 480 megabytes used by the uniprocessor.

If we assume that all the nodes took the same 9.4 hours of cpu time, the total computation comes to twelve times this figure, or 112.8 hours. Earlier experiments with heavily instrumented versions of the parallel code, admittedly working on much less dense matrixes, showed that this assumption is not too far from the truth. A very naive computation of the total number of cpu cycles used by each implementation yields 1.9×10^{14} for the uniprocessor, and 1.2×10^{14} for the cluster. Although it appears at first sight that the parallel implementation is *more* efficient, it must be stressed that this is a very over-simplified analysis. Firstly, the code was compiled with substantially different compilers and run under very different operating systems. Secondly, even when the compilers and

operating systems are identical, the run-time can be heavily dependent on memory bandwidth, cache efficiency, and other non-computational effects. In both cases, the linear algebra phase took less than 0.1% as much computation as did the sieving (Section 4 below).

Our discovery that the parallel version of the block Lanczos computation took *fewer* machine cycles than the serial version came as a surprise, albeit a pleasant one. Previous (and so far unpublished) runs on other larger but sparser matrixes have suggested that there is a speed-up from parallelization, in that the computation is completed in a shorter elapsed time, but that a greater number of machine instructions are executed overall. In any event, the parallel implementation has one advantage: a 32-bit machine rarely supports more than a gigabyte or two of physical memory, whereas our relatively small and old cluster contains a total of six gigabytes and much bigger clusters are economically feasible. For these reasons we caution against assuming that fairly small RSA public moduli are safe because "the matrix problem is too hard".

2.5 Combination of Relations and Production of the Factors

The combination of linearly-dependent relations produced by the linear algebra used exactly the same code as in [11]. Processing each dependency took 18 minutes on a 400MHz machine. The third dependency yielded the factors $p =$ 337 779 774 700 456 816 455 577 092 228 603 627 733 197 301 999 086 530 154 776 370 553 and $q =$ 346 129 173 115 857 975 809 709 331 088 291 920 685 569 205 287 238 835 924 196 565 083 957 of 2,1606L.c135 = 116 915 434 112 329 921 568 236 283 928 181 979 297 762 987 646 390 347 857 868 153 872 054 154 807 376 462 439 621 333 455 331 738 807 075 404 918 922 573 575 454 310 187 518 221. At 66 digits, p is the largest penultimate factor ever found by the MPQS algorithm.

3 Cycle Behaviour

3.1 Introduction

As the number of relations increases, the number of cycles also increases — at an ever increasing rate. This much has been known ever since the introduction of the quadratic sieve algorithm around twenty years ago. When only one large prime is allowed in relations, the rate of growth in number of cycles is very nearly proportional to the number of relations squared. An explanation of this behaviour in terms of the birthday paradox (slightly modified because the large primes do not appear with equal frequency) was given in [11]. In [2], where two large primes were used, it was reported that the quadratic behaviour broke down about half way through the computation, and that a better approximation for the later stages was that the number of cycles grew quartically with the number of relations. It was mentioned that a quartic function, although a good approximation in practice, could not be the entire truth as the residuals showed

distinctly non-random behaviour. This factorization used PPMPQS and a large fraction of the cycles arose from the presence of *ppr* relations.

The small-scale preliminary tests of the TMPQS algorithm mentioned earlier showed that deviations from quadratic growth were much more pronounced and, in particular, a simple power law never gave a good fit to the number of cycles produced during the entire course of the computation. A few test runs seemed to indicate that a sudden growth in the number of cycles appeared towards the end of the calculation, reminiscent of the behaviour reported in [9] where four large primes were used in a GNFS factorization.

Our results for the TMPQS factorization of 2,1606L.c135 are summarized in Fig. 1, where ln(number of cycles to date) is plotted against ln(number of relations to date) during the course of the computation. Points corresponding to

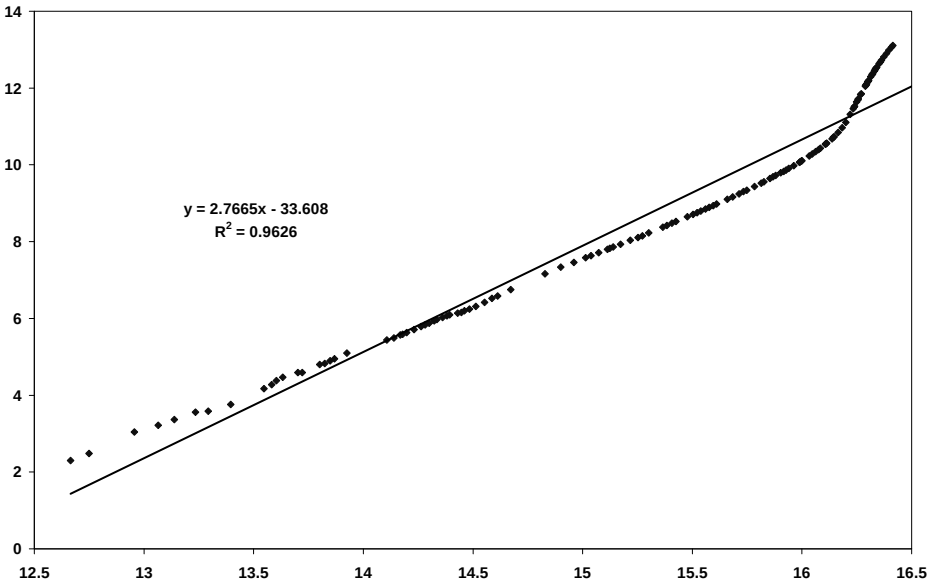

Fig. 1. Behaviour of ln(# cycles) with ln(# relations). The line is the least squares best fit to the data. Points corresponding to data with fewer than 10 cycles are omitted.

the period when fewer than ten cycles were available are omitted. The line is the least squares fit to the data: its slope of 2.7665 shows that the growth is faster than quadratic, but it is very clear that a power law is not a good approximation to the behaviour.

3.2 Cycle Types

The observation that cycle-growth behaviour depends on the type of relations appearing in cycles occurring was made in [2] but the authors (two of whom are authors of this paper) did not fully appreciate the consequences. As part of this investigation we characterized the cycles more fully and were able to get a better (though far from complete) understanding of the situation.

As already stated, if only *par* relations are used in cycles, the cycle-growth behaviour is well understood and shows very nearly quadratic growth. Our idea was to remove the cycles of this form (which necessarily consist of only two relations each) and to examine the behaviour of the remainder. Cycles consisting only of *pars* we termed S-cycles (S for single prime). Having made this distinction, there is an obvious characterization of the remaining cycles: those which do and those which do not contain at least one *tpr*. The former we called T-cycles ("triple prime") and the latter D-cycles ("double prime"). For convenience, we call the total number of cycles C and so, with the obvious notation, $C = S + D + T$.

It was thought that two other quantities might be useful in understanding the manner in which cycles were formed as relations were added. These are the number of useful primes, U, defined as the number of distinct primes appearing in those relations which form cycles, and the number of pruning passes, P, needed to remove singleton primes from the graph.

During the course of the computation we calculated C, S, D, T, P and U approximately daily. We were thus able to examine the variation of these quantities as a function of the total number of relations R in some detail. The behaviour of each of these quantities as a function of R is described in subsequent subsections. In each case where logarithms are taken, we restrict the data points to those with values ≥ 10 so as to minimize noise effects arising from insufficient data.

3.3 S-Cycles

In Fig. 2 we plot $\ln(S)$ against $\ln(R)$. The birthday-paradox analysis would suggest that the plot should be linear and with a slope very close to 2.0. This is, in fact, what we find. The least-squares fitted slope is 2.078 with a correlation coefficient squared of 0.9997. At the end of the factorization we had 25 603 S-cycles, or 5.18% of the total.

3.4 D-cycles

In [2] it was observed that $S + D$ grew more rapidly than a quadratic function of R, with some suggestion that a quartic might be a better approximation. We are not aware of any other analysis of this quantity. In Fig. 3 we show $\ln(D)$ plotted against $\ln(R)$ together with the least squares straight line fitted through the points. The large gap between $14.7 < \ln(R) < 15.0$ corresponds to a period when counts broken down into S, D, and T were not taken. The edges of the missing-data gap correspond to $D = 100$ and $D = 389$. Prior to this period, T

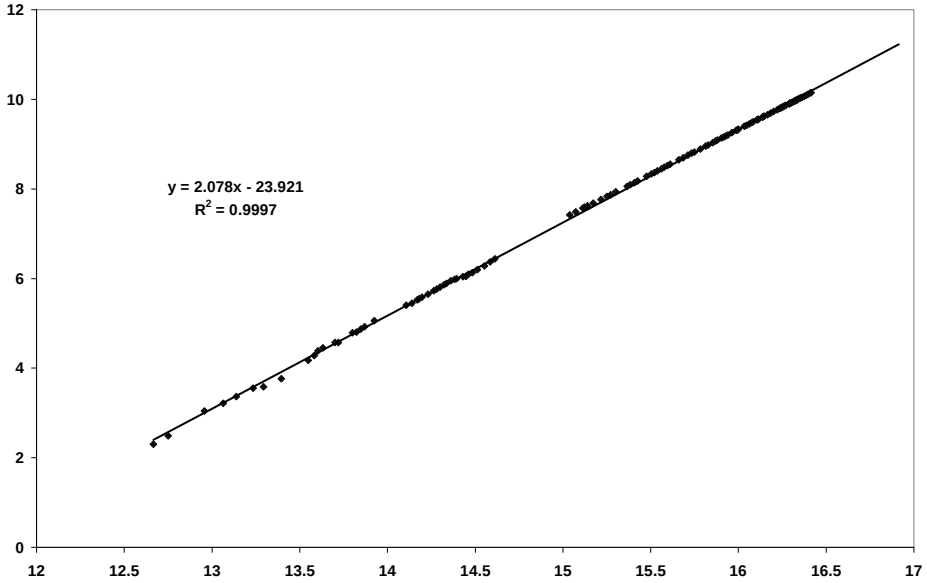

Fig. 2. Behaviour of ln(# S-cycles) with ln(# relations). The line is the least squares best fit to the data. Points corresponding to data with fewer than 10 cycles are omitted.

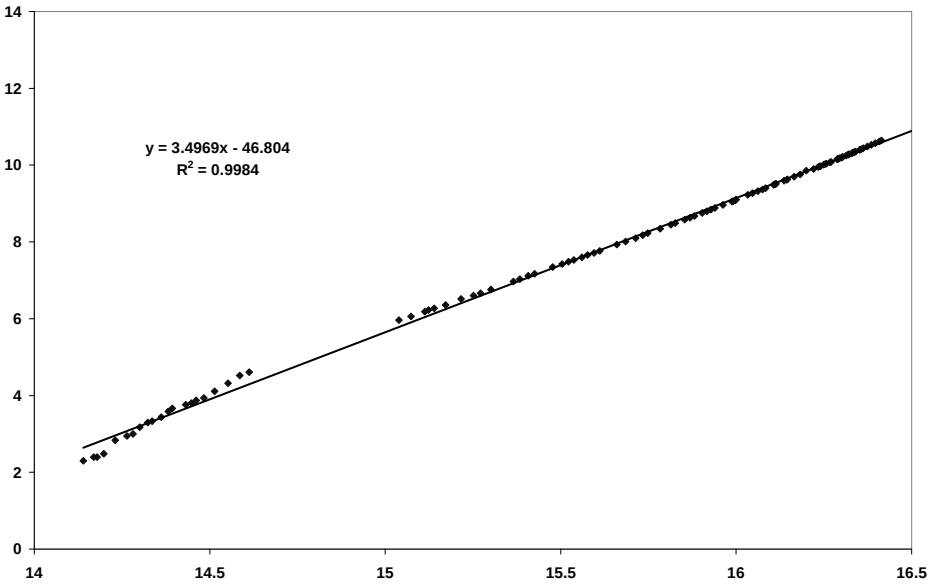

Fig. 3. Behaviour of ln(# D-cycles) with ln(# relations). The line is the least squares best fit to the data. Points corresponding to data with fewer than 10 cycles are omitted.

was zero or one (the first T-cycle was eagerly awaited!) and it was possible to calculate D from the values of C and U, which were recorded.

The points lie remarkably closely on a straight line which has slope 3.4969, though there is a hint that the relationship breaks down at very small values of D. It is tempting to suggest that the slope should be exactly $7/2$, but we have *no* theoretical justification for this claim.

By the end of the factorization we had 41 940 D-cycles, or 8.49% of the total.

3.5 T-cycles

When Figs. 2 and 3 are compared with Fig. 1, it is clear that any major non-linearity in the log-log plots must arise from the T-cycles. This is indeed what we find in Fig. 4. The fitted straight line, with a slope of 7.3207, is not a particularly good fit to the data and the curve between $16.1 < \ln(R) < 16.3$ shows pronounced curvature. This region corresponds approximately to $1 \times 10^6 < R < 1.2 \times 10^6$.

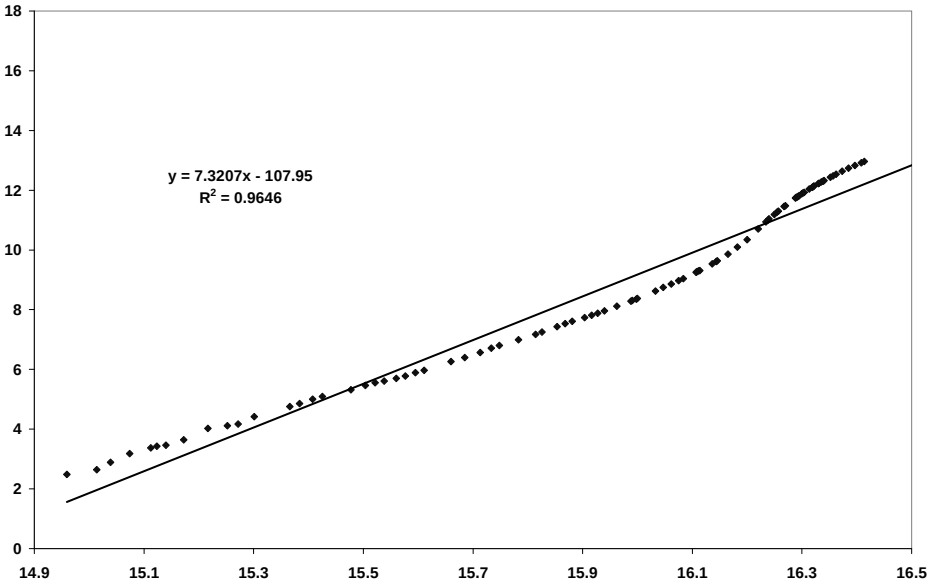

Fig. 4. Behaviour of ln(# T-cycles) with ln(# relations). The line is the least squares best fit to the data. Points corresponding to data with fewer than 10 cycles are omitted.

As can be seen from Fig. 4, the initial growth of T-cycles is quite smooth. The portion of the plot for $\ln(R) < 15.9$ (or $R < 810\,000$) lies closely on a straight line with slope 5.4785 corresponding to a power law with this exponent. Beyond this point, the plot shows an initially rising gradient, reaching a maximum of around

15, and then tailing off again to about 6 near $R = 16.35$. A somewhat similar effect was observed in an early multiple large prime GNFS factorization [9] where it was described as explosive growth.

In our computation, the T-cycle growth showed strongly superpolynomial growth for a short while, but the term "explosion" seems too strong for our experience and we prefer "phase transition" as being more appropriate. The two cases, GNFS in [9] and TMPQS in this paper, would be expected to show different cycle behaviour. In the former work, relations had four large primes — two algebraic and two rational. The large primes may only be combined with others of the same type, which limits the extent to which relations may be cross-linked. In TMPQS, although there are fewer primes per relation, any of the three primes appearing in a *tpr* relation may combine with any of the primes in other relations. Therefore, it is perhaps not too surprising that we see a phase transition towards the end of the sieving phase.

At the end of the factorization we had 426 534 T-cycles, which is 86.33% of the total number of cycles $C = 494\,077$.

3.6 Phase Transition in T-cycle Growth

We now justify our choice of the phrase "phase transition" to describe the superpolynomial growth in T cycles when relations are added to the graph beyond a certain point. Forgetting for a while the reason why we construct a graph of large-prime relations and seek cycles within it, we concentrate on a simple physical model for the repeated addition of relations to the graph. It was convenient when counting cycles to perform manipulations on the graph in which nodes represented large primes and three arcs represented a relation. A simple relationship holds between the number of objects in this graph, and the number of cycles among the relations themselves. In our simple model we picture relations as having some of the properties of atoms in chemistry. In particular, *par*s are represented as univalent atoms, *ppr*s as divalent and *tpr*s as trivalent atoms. The silicon - oxygen system shows some of the phenomena we are attempting to model, though care must be taken not to push the analogy too far.

At the beginning of the computation there are no relations, and so the model consists of a vacuum containing no atoms. As relations are added most will not share any primes with relations already present. In our model, these form a monatomic gas. A few will share a prime with another relation; the two atoms will form a chemical bond with one of their valencies. Initially at least, only pairs of *par* relations will have all three primes matched (a $(\mathbf{1}, \mathbf{1}, p)$ relation forming a cycle with another containing the same large prime p). This corresponds in our model to a diatomic molecule formed of two univalent atoms. Other matched relations (diatomic molecules) contain unmatched primes (unsatisfied valencies) and remain available for the addition of further relations (are very reactive in the presence of other atoms and molecules).

As relations are added to the collection, the number with matched primes increases. Equivalently, as atoms are added to the gas the density rises and more molecules are formed. Eventually, the density rises high enough that large

and complicated molecules are produced, some forming rings and chains. Each divalent atom added to a molecule is likely to bind only to one free valency, thereby growing a chain. Initially, at least, each trivalent atom will add an extra free valency to the growing molecule.

We now ask: what is the connection between the chemical model and the cycle-counting algorithm? The answer should be clear: each pass of the singleton removing algorithm removes those primes which occur only once at that point. In the chemical model, this corresponds to breaking the bonds to those atoms which contain a free valency. When singleton removal has run its course, only cycles remain; when all bonds to reactive sites have been broken, only stable molecules remain.

As the density continues to increase, not only will rings of atoms within a molecule build up, adjacent molecules will connect to each other via a di- or trivalent atom. In real chemical systems, such as the addition of silicon and oxygen atoms to a container, at first the atoms are largely isolated. Then stable O_2 and reactive SiO and Si_2 molecules are formed, and then stable SiO_2 molecules together with a whole raft of highly reactive silicon-oxygen molecules with free valencies. All these atoms and molecules are still in the gas phase. Eventually the density becomes so high that the molecules cross-link in profusion and the system as a whole becomes a liquid, glass or solid (depending on temperature and pressure) in equilibrium with a vapour. A phase transition has taken place and the properties of the condensed phase is very different from the properties of the earlier gaseous phase.[2] In the TMPQS case, the condensed phase appears not to be crystalline (we do not find one massive and almost fully interconnected component), but rather more akin to a glass. Many of the relations are connected together but in a number of components and with the presence of a large number of lengthy chains which terminate in a free valency and, in all, roughly three-quarters of the relations remain in the gas phase — to mix metaphors badly.

When phase changes occur in physical systems, it is usual for many physical properties to change dramatically over a small variation in a quantity such as temperature, density or magnetic field strength. Some properties, such as the density increase on condensation, or viscosity on polymerization, show themselves as near-step functions with an increased gradient at the phase transition in the phase diagram for the system. Such behaviour is similar to that seen in Fig. 4. Other quantities, such as the specific heat capacity, show relatively flat behaviour well away from the phase transition and rise to a sharp peak at the transition itself. It is normal for the heat capacity of the two phases to be different. To take a physical system almost at random, Doye, Sear and Frenkel [10] describe a phase transition in the molecular configuration of four moderately-sized polymers. Fig. 6a of their paper looks somewhat similar to our Fig. 4

[2] In a real physical system, molecular bonds are broken and reformed in different combinations and, all the time, the condensed phase is dynamically exchanging material with the vapour. Relations do not break and reform arcs to other relations. We did warn against pushing the model too far.

though, it must be admitted, the slopes are very different. When we realise that there is a systematic slope in the $\ln(T)$ versus $\ln(R)$ graph for small values of R and compensate for this feature by subtracting the least squares fitted line to this data ($\ln(T) = 5.4785\ln(R) - 79.469$) we produce Fig. 5. The resemblance

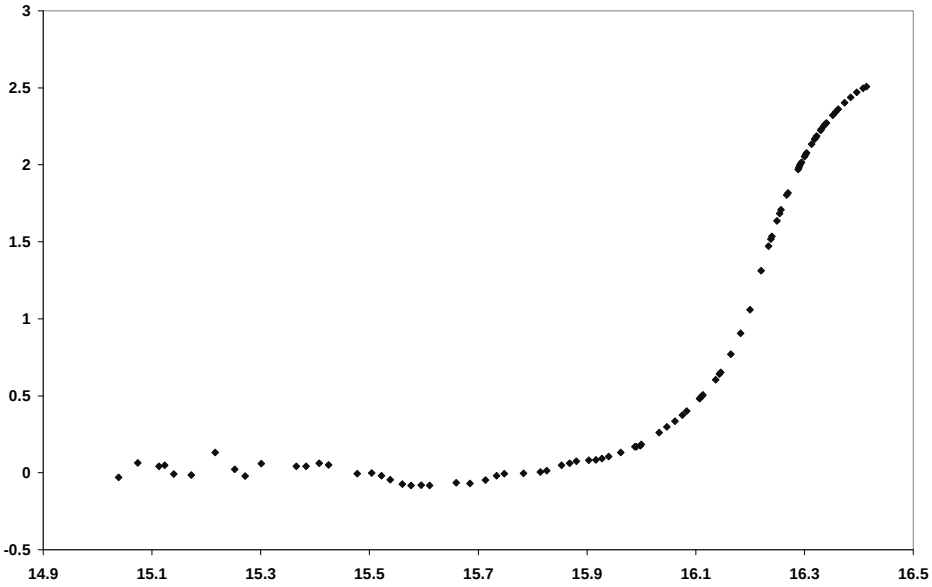

Fig. 5. $\ln(T) - 5.4785\ln(R) + 79.469$ plotted against $\ln(R)$

between this plot and Fig. 6a of [10] is remarkable. An even more startling similarity can be seen between Fig. 6b of [10] (the temperature - specific heat capacity plot) and our Fig. 6 which shows a plot of the number of pruning passes P against R. The two are almost identical in appearance! In our experiment, the "pruning capacity" rises slowly from about 10 to around 25 in the gaseous phase, grows through a peak of almost 140 at the phase change itself and settles down at about 40 in the condensed phase.

We conclude this section with a repeat of our warning: cycles among relations in a TMPQS factorization are *not* isomorphic to molecules in a chemical mixture, and the analogy should not be pushed too far. Nonetheless, the two systems show remarkably similar behaviour and it would appear that this may be a fruitful field for further study, not least because a similar phenomenon also seems to occur in the GNFS. If the phenomenon were better understood, we may have a hope of selecting sieving parameters which bring forward the onset of the phase transition without unduly slowing down the rate at which relations are produced.

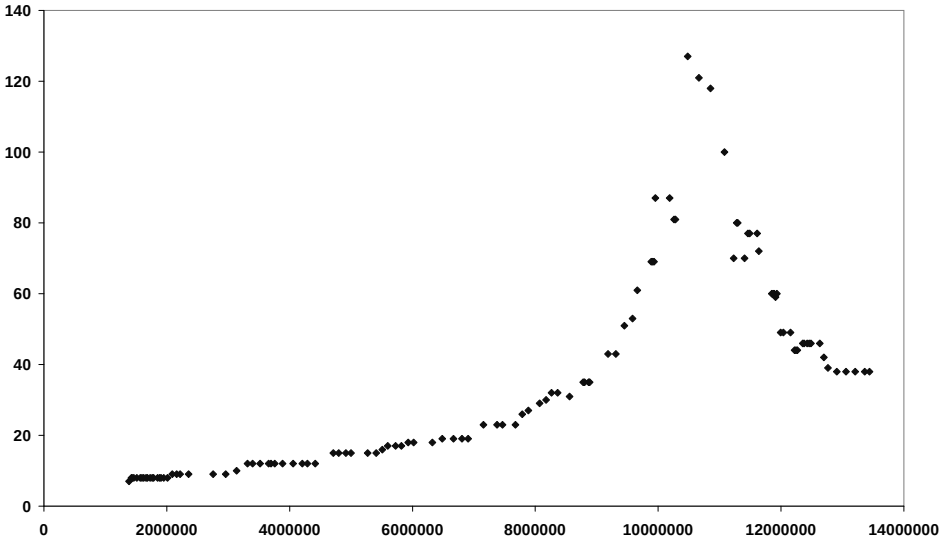

Fig. 6. Behaviour of (# pruning passes) with (# relations).

4 Performance Comparison of TMPQS and PPMPQS

As mentioned in the abstract, one of our objectives in performing the factorization of 2,1606L.c135 with TMPQS was to compare its performance with that of PPMPQS and to test the assertion made in [12] and [8] that TMPQS would be the slower algorithm. Because of the magnitude of the computations required, it is unreasonable to repeat the factorization of a 135-digit integer with PPMPQS purely to make the performance comparison. Luckily, we need not re-do the computation because the factorization of a number of quite similar magnitude (RSA-129) has been performed by PPMPQS and performance estimates published [2].

Estimating the amount of computation required to complete a calculation which has been spread over many machines of radically different architectures is fraught with difficulty and can, at best, give only an approximate answer. Nonetheless, it is useful to attempt the estimate, if only to give guidance to (or to daunt!) prospective factorers. In the earlier work [2], values ranging from 1700 MIPS-years to 6000 MIPS-years to factor a 129-digit integer were given, depending on machine architecture. An "average" value was quoted as 5000 MIPS-years, where a MIPS-year corresponds to 3.1×10^{13} machine instructions — so the complete computation took around 1.5×10^{17} instructions on an "average" machine. The first author took part in that earlier computation and still has some records from it. He also has one of the machines, a DECStation 5000/25, which contributed several months work to the sieving effort. Detailed records from that

machine have been lost, but an email sent in 1994 from the first author to the second contains:

```
A notional 3 GIPS 16k MasPar would have required 7000 mips years.
A notional 15 MIPS DEC 5000/25 would have required 5960 mips years.
A notional 25 MIPS Sparc-2 would have required 4200 mips years.
A notional 20 MIPS 486DX33 would have required 4200 mips years.
A notional 1.5 MIPS Sun 3/50 would have required 1500 mips years.
```

A figure of 5960 MIPS-years corresponds to 1.85×10^{17} instructions on this 15 MIPS machine.

The same machine, in a single uninterrupted run of TMPQS lasting 82.5 days, generated 4901 relations (24 *ful*s, 295 *par*s, 1608 *ppr*s and 2974 *tpr*s) in 7 117 213.74 cpu-seconds. The complete factorization of 2,1606L.c135 required a total of 13 441 627 relations. To generate this number of relations would have taken the notional 15 MIPS DECstation 5000/25 a total of 2.92×10^{17} machine instructions ($13.4 \times 10^6 \cdot 7.12 \times 10^6 \cdot 15 \times 10^6/4901$). Thus we conclude that this machine would have taken 1.6 times as much computation to factor the 135-digit integer 2,1606L.c135 by TMPQS as it would have done to factor the 129-digit integer RSA-129 by PPMPQS.

The asymptotic runtime of all variants of the quadratic sieve algorithm to factor an integer N is well known to be $O(\exp((1+o(1))\sqrt{\log N \log\log N}))$ [15]. Variations such as the number of large primes used, the value of the multiplier k, the machine architecture and the efficiency of the compiler affect only the $o(1)$ term. Differences in the last two of these quantities can be negated by using identical environments. By a happy coincidence, $k = 1$ was used in both factorizations, removing another source of variability. These constants permit credible estimation of the effect of the first factor. Even though the $o(1)$ term for a given implementation is neither zero nor constant, making the assumption that it is zero has been shown to be reasonable when the variation in size of N is quite small and when all that is required is an estimate of the relative amounts of work taken to factor two integers with the same implementation. In our case, the six digit difference between RSA-129 and 2,1606.c135 is sufficiently small to make this assumption.

Substituting the values of RSA-129 and 2,1606L.c135 into the runtime expression above suggests that for the same implementation running on the same hardware, the latter integer is 3.1 times more difficult to factor than the former. As the hardware is identical, and the software implementation as near to identical as possible, we can conclude that the difference between the two estimates above is due almost entirely to the effect of using three large primes for the latter factorization and only two for the former. Accordingly, we conclude that for integers in the neighbourhood of 10^{130} to 10^{135} TMPQS is approximately $3.1/1.6 = 1.9$ times as efficient as PPMPQS. Note that this factor of 1.9 increase in performance is a minimum value. We may have chosen the sieving parameters sub-optimally, especially the size of the factor base, and an improved understanding of the effects of these choices on the location and behaviour of the phase transition may well improve the situation further.

Unfortunately, although TMPQS appears to be an improvement on PPMPQS, it is still not competitive with GNFS when factoring integers of around 135 digits. Results for the GNFS factorizations of RSA-130 and RSA-140 have been published in [7] and in [6] respectively. On a scale where all costs are normalized to RSA-129 = 5000 MIPS-years we used 8000 MIPS-years to factor 2,1606L.c135 whereas RSA-130 required 750 MIPS-years and RSA-140 took 2000 MIPS-years.

5 Conclusions

We have completed the factorization of the largest integer ever split by the Quadratic Sieve algorithm. At 135 digits, our achievement breaks the previous record [2] by 6 digits. The penultimate prime factor, at 66 digits, is also the largest ever found by QS.

Our work has shown that TMPQS can be competitive with PPMPQS, despite the gloomy prognostications of [8] and [12]. Indeed, for our implementation and for integers with about 135 digits, the method has been measured to be almost twice as fast as the earlier algorithm. However, TMPQS is still slower than GNFS at factoring integers of this size. Performance figures reported in [6] and [7] indicate that GNFS is about six times faster.

The reason why TMPQS is more efficient than PPMPQS is because cycles amongst relations containing three large primes grow much faster than cycles restricted to containing two or fewer primes and, in particular, a sudden rapid rise in the number of cycles occurs when a critical number of relations have been found. We do not understand the complete details of what is taking place at this point, but we have given semi-quantitative descriptions of the phenomenon and have suggested an interpretation in terms of a phase transition, analogous to condensation or polymerization in a chemical system. Further work to gain a more thorough understanding of the cycle-growth behaviour would be valuable, not least because a similar phenomenon has been shown to occur in GNFS and we may be able to employ this understanding to improve the efficiency of that algorithm.

References

1. W.R. Alford, C. Pomerance, *Implementing the self-initializing quadratic sieve on a distributed network* in A.J. van der Poorten, I. Shparlinski, H.G. Zimmer, editors, *Number-theoretic and algebraic methods in computer science: NTAMCS '93*, World Scientific Publishing, (1995) 163–174.
2. D. Atkins, M. Graff, A.K. Lenstra, P.C. Leyland, *THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE*, Proceedings Asiacrypt'94, LNCS 917, Springer-Verlag (1995), 265-277.
3. J. Buchmann, J. Loho, J. Zayer, *Triple-large-prime variation*, manuscript, 1993.
4. B. Carrier, S.S. Wagstaff Jr., *Implementing the hypercube quadratic sieve with two large primes*, Technical Report 2001-45, Purdue University CERIAS (2001) URL `http://www.cerias.purdue.edu/papers/archive/2001-45.{ps,pdf}`.

5. S. Cavallar, *Strategies in filtering in the number field sieve*, ANTS-IV, LNCS 1838, Springer-Verlag (2000) 209–231.
6. S. Cavallar, B. Dodson, A.K. Lenstra, P.C. Leyland, W. Lioen, P.L. Montgomery, B. Murphy, H. te Riele, P. Zimmermann, *Factorization of RSA–140 using the number field sieve*, Proceedings Asiacrypt'99, LNCS 1716, Springer-Verlag (1999), 195–207.
7. J. Cowie, B. Dodson, R.-M. Elkenbracht-Huizing, A.K. Lenstra, P.L. Montgomery, J. Zayer, *A world wide number field sieve factoring record: on to 512 bits*, Proceedings Asiacrypt'96, LNCS 1163, Springer-Verlag (1996), 382–394.
8. R. Crandall, C. Pomerance, *Prime Numbers, a computational perspective*, Springer (2001) 237.
9. B. Dodson, A.K. Lenstra, *NFS with four large primes: an explosive experiment*, Proceedings Crypto'95, LNCS 963, Springer-Verlag (1995) 372–385.
10. J.P.K. Doye, R.P. Sear, D. Frenkel, *The effect of chain stiffness on the phase behaviour of isolated homopolymers*, J. Chemical Physics 108, (1998) 2134–2142.
11. A.K. Lenstra, M.S. Manasse, *Factoring by electronic mail*, Proceedings Eurocrypt'89, LNCS 434, Springer-Verlag (1990) 355–371.
12. A.K. Lenstra, M.S. Manasse, *Factoring with two large primes*, Math. Comp. 63 (1994) 785-798.
13. P.L. Montgomery, *A block Lanczos algorithm for finding dependencies over GF(2)*, Proceedings Eurocrypt'95, LNCS 921, Springer-Verlag (1995), 106–120.
14. P.L. Montgomery, *Distributed Linear Algebra*, 4th Workshop on Elliptic Curve Cryptography, Essen (2000)
URL `http://www.cacr.math.uwaterloo.ca/conferences/2000/ecc2000/montgomery.{ppt,ps}`.
15. C. Pomerance, *Analysis and comparison of some integer factoring algorithms* in H.W. Lenstra Jr, R. Tijdeman, editors, *Computational methods in number theory, Part I*, Math. Centre Tracts, Math. Centrum (1982) 89–139.