MIMD-Factorisation on Hypercubes

F. Damm[1], F.-P. Heider[2], G. Wambach[1]

[1] University of Cologne, Institute of Computer Science, D-50969 Cologne, Germany
fdamm@informatik.uni-koeln.de, gw@informatik.uni-koeln.de
[2] University of Cologne, Institute of Mathematics, D-50923 Cologne, Germany
heider@itsec.bonn.cap-debis.de

Abstract. This paper describes the development and implementation of the MPQS factoring algorithm using multiple hypercubes customised to a MIMD parallel computer. The computationally most expensive steps ran on a Parsytec machine consisting of 1024 Inmos T805 microprocessors. General 100 decimal digit numbers can be factored in 1–2 days.

1 Introduction

The installation of a 1024 processor MIMD machine at our university in the spring of 1993, and a course on the parallelisation of number theoretical algorithms given by the second author incidentally in the winter term 1992/93 initiated the idea to estimate the theoretical and practical effort to start a significant factorisation experiment on a dedicated machine from scratch. The goal was to factor 100 digit numbers within 1–2 days of CPU time.

Of course, a parallel version of the MPQS algorithm, based on ideas of Dixon, Pomerance and Montgomery and described in [8, 11], seemed to be the right point to start. However, a straightforward approach with processors running independently as in [3] is impossible because of memory constraints. The analysis of the technical capabilities of the Parsytec machine shows there should not be too much communication to guarantee high performance computing.

From the many variations of the MPQS algorithm discussed in the literature we felt that Peralta's MPQS on a hypercube [7] which was still not implemented could serve as an adequate basis. A careful study showed that this idea did not fully exploit the internal structure but that considerable improvement was possible and useful on our machine. It seemed advantageous not to stay with a single hypercube but to work on several ones in parallel. Theoretically we expected a loss in useful relations originating in Peralta's method, due to the effect that a-values from the polynomials used are divided by several primes in the factorbase. To compensate for this effect we devised a subtle procedure similiar to the one used in [5] to combine partially factored numbers from the sieving stage.

The next major task was to find a parallel implementation of the algorithm which makes optimal use of the machine at hand. Our first method separates the traverse of the hypercubes from the sieving process. The MIMD property enables us to run the hypercube part on a small number of processors while the sieving part is performed at the majority of processors. Since the distribution of both parts is mainly determined by the number of processors at hand and the underlying hardware topology is neglected, we developed a more flexible parallelisation. Here the processors are grouped into rings along which the data is distributed.

The first factorisation falling within the initially intended domain was completed on December 24th after 41 hours total runnning time. As a contribution to the Cunningham list we give the formerly unknown 41 and 56 digit prime factors of the remaining 97 digit cofactor ($C97$) of $12^{327}+1$. Shortly afterwards, a 101 digit cofactor of $5^{273}-1$ ($C101$) was factored into 36 and 66 digit primes.

The complete development, programming from scratch and debugging of the software took about 6 man-months. Most of the programs are written in C, only some modules of the multiprecision arithmetic are written in T805-Assembler.

2 MPQS on a Hypercube

We assume familiarity with the 'Multiple Polynomial Quadratic Sieve' (MPQS) algorithm [8, 11] and will sketch only the improved hypercube variation used by us.

Let N be the composite integer to be factored. After choosing a factorbase $\mathcal{F}$ of R primes $p_i, 1 \leq i \leq R$, and a sieve length M, a lot of quadratic polynomials $Q_{ab}(X) = a^2X^2 + 2bX + c$ with $b^2 - N = a^2c$ are generated. It follows that $Q_{ab}(X) \equiv (a^2X + b)^2a^{-2} \bmod N$. The requirement $|Q_{ab}(-M)| \approx |Q_{ab}(0)| \approx |Q_{ab}(M)|$ leads to the condition $a^2 \approx \sqrt{2N}/M$. For every such polynomial the roots modulo $p_i, 1 \leq i \leq R$, must be computed, the interval $[-M, M[\cap\mathbb{Z}$ is sieved, and the candidates are collected.

Now for every prime p in the factorbase let t_p be a square root of $N \bmod p$: $t_p^2 \equiv N \bmod p$. If p does not divide a, then

$$Q_{ab}(x) \equiv 0 \bmod p \Leftrightarrow x \equiv (-b \pm t_p)a^{-2} \bmod p.$$

The t_p's are independent of the Q_{ab} and will be computed only once. But $a^{-2} \bmod p$ and $b \bmod p$ for every a and for every $p \in \mathcal{F}$ have to be computed.

In [11] the a's are (pseudo-)primes not divisible by any $p \in \mathcal{F}$. Montgomery (quoted in [9]) and Peralta [7] independently observed that if $a = \pi_1 \cdot \ldots \cdot \pi_l$ is the product of l primes π_i (such that N is a quadratic residue modulo π_i), there are 2^l different values $b \bmod a^2$ with $b^2 \equiv N \bmod a^2$. Since $Q_{ab}(x) = Q_{a(-b)}(-x)$, we get 2^{l-1} different polynomials with each a.

Given α_j, β_j with

$$\alpha_j^2 \equiv N \bmod \pi_j^2$$

and

$$\beta_j \equiv \begin{cases} 1 \bmod \pi_j^2 \\ 0 \bmod \pi_i^2 & , i \neq j \end{cases}$$

every b can be written uniquely as $b = \sum_{j=1}^{l} \delta_j \alpha_j \beta_j \bmod a^2$ where $\delta_j \in \{+1, -1\}$. We fix $\gamma_j = \pm\alpha_j\beta_j \bmod a^2$ such that γ_j is less than $\frac{a^2}{2}$.

R. Peralta further noticed that the solution set of $b^2 \equiv N \bmod a^2$ is structured like an l-dimensional hypercube $C_l = \{-1,+1\}^l$, vertices corresponding to solutions b. Two vertices are adjacent if the corresponding solutions b, b' differ at exactly one sign δ_j. He suggested to follow a certain hamiltonian cycle of $C_{l-1} = (C_{l-2} \times \{-1\}) \cup (C_{l-2} \times \{+1\})$ resulting in a sequence $k_i, 1 \leq i \leq 2^{(}l-1)-1$, with $k_i = j$ if step i changes coordinate j, such that the tour

$$b_{i+1} = b_i + 2\mu_i \gamma_{k_i} \bmod a^2$$

with $\mu_i = +1$ or -1 depending on whether step i changes coordinate k_i from $-$ to $+$ or from $+$ to $-$ visits all useful vertices of the hypercube C_l. Omitting the reduction of $b \bmod a^2$, still $|b| < l \cdot \frac{a^2}{2}$ holds which is sufficient for the estimate of $Q_{ab}(x)$ over $[-M, M[$.

Whereas Peralta devised an algorithm requiring three additions and one multiplication modulo every prime p in the factorbase and a table of $6lR$ integers to step from b_i to b_{i+1}, we use precomputed tables of $2\gamma_j a^{-2} \bmod p$ for $1 \leq j \leq l-1$ and every p in the factorbase to obtain from the modular roots $x_i = (-ba^{-2} \pm t_p a^{-2}) \bmod p$ of $Q_{ab_i}(X)$ the roots

$$x_{i+1} = (x_i - 2\mu_i \gamma_{k_i} a^{-2}) \bmod p$$

of $Q_{ab_{i+1}}(X)$ in only two additions mod p. The cost for doing this results in the additional space consumption of $(l-1)R$ integers. As a second improvement we sieve many hypercubes at a time which allows us to choose hypercubes of smaller dimensions.

Nearly factored candidates out of the sieve stage are relations of the form

$$q_1 q_2 \prod_{i=0}^{R} p_i^{c_i} \equiv z^2 \bmod N \tag{1}$$

with $q_i = 1$ or $q_i > p_R$ prime. This is called a full relation if $q_1 = q_2 = 1$, a partial relation if exactly one of the q_i's is one, and a partial partial relation otherwise. Exploiting the idea of using partial partial relations [5] more than compensates for the negative effect that a-values from the polynomials used are divided by several primes in the factorbase.

Among the techniques used to speed up MPQS this is the most efficient one. The factor graph $G = (V, E)$ is built from relations of the form (1). The first node in V is identified with 1. All the primes q_1 and q_2 appearing in (1) make up the rest of the nodes. An edge $e = (v, w)$ in the graph corresponds to a relation (1) with $v = q_1$ and $w = q_2$.

Two algorithms investigate and exploit the graph. The first one cuts off leaves and isolated nodes repeatedly and the second one is the breadth-first search algorithm. By cutting off leaves and isolated nodes we obtain structural information about the factor graph and reduce the memory demands when processing the cycles. By breadth-first search we look for cycles in G, whenever one is found, it is stored, the last edge traversed is deleted from G, and the search is continued. This gives a basis of the cycle space of G.

Each of the cycles found produces a full relation. Let us e.g. use a cycle of length 4 passing node 1, that is a situation like

$$1 \cdot r \cdot \prod_{i=0}^{R} p_i{}^{c_{1,i}} \equiv z_1^2 \bmod N \ , \ r \cdot s \cdot \prod_{i=0}^{R} p_i{}^{c_{2,i}} \equiv z_2^2 \bmod N,$$
$$s \cdot t \cdot \prod_{i=0}^{R} p_i{}^{c_{3,i}} \equiv z_3^2 \bmod N \ , \ 1 \cdot t \cdot \prod_{i=0}^{R} p_i{}^{c_{4,i}} \equiv z_4^2 \bmod N$$

Thus

$$\prod_{i=0}^{R} p_i{}^{c_{1,i}+c_{2,i}+c_{3,i}+c_{4,i}} \equiv \left(\frac{z_1 z_2 z_3 z_4}{rst}\right)^2 \bmod N$$

In this setting, pairs of partial relations are cycles of length 2.

In case a multiplier m is used with MPQS there are two possibilities. Either it is included in the factorbase or the graph is extended by a node for m. In the second possibility, at the place of $q_1 \cdot q_2$ in (1) $m \cdot q_1 \cdot q_2$ can appear. Using the corresponding edges, cycles must be treated differently depending on the parity of the number of such edges.

In the meantime we have learned from two other implementations of the hypercube MPQS algorithm. R. Alford and C. Pomerance [1] are using polynomials $Q_{ab}(X) = aX^2 - 2bX + c$ such that $aQ_{ab}(X) \equiv (aX - b)^2 \bmod N$. Taking a instead of a^2 makes the hypercube MPQS applicable for smaller numbers, too. On the other hand, the use of $aQ_{ab}(X)$ instead of $Q_{ab}(X)$ increases the number of ones in the matrix used in the final step of the MPQS algorithm. Moreover, special care has to be taken for avoiding redundant relations which we don't have considered yet.

Using only one precomputed table of $2a^{-2} \bmod p$ for all $p \in \mathcal{F}$, one still has to compute $\gamma_j \bmod p$, one multiplication and two additions for all $p \in \mathcal{F}$ when changing polynomials.

3 Machines Used

The Parsytec GCel installed at our university's "Zentrum für Paralleles Rechnen" consists of 1024 Inmos T805 transputers. These are clocked at 30 MHz. Every processor has 4 MByte RAM, 350 KB of which are occupied by the operating system, and 4 KB Cache, 3KB of which are occupied, too. Every 16 processors build a cluster in form of a 4x4-grid. The physical network topology is a two-dimensional grid, virtual topologies can be programmed in software. The machine is designed for up to 16.384 processors. The communication bandwidth is at most 1,1 MB/sec (no intermediate links) and 0,6 MB/sec (intermediate links), respectively. The outside gate is a Sun workstation.

To enable a comparison with other machines, one has to consider mainly the sieving capabilities of the processor. Therefore we used 'nsieve[3] 1.2', which rated one T805 with 1.8 nsieve-MIPS independently of the array size. (A notional 1.5 MIPS Sun 3/50 is rated 2.1–3.5 nsieve-MIPS.)

[3] available from `ftp.nosc.mil` in the directory pub/aburto

Concerning hardware and operating system (an unix-derivation called parix), the machine works reliably, while the support for software development could be improved upon. (We e.g. missed a tool logging processor activity for analysis after execution.) Most of the code was produced and tested on standard workstations running under the unix operating system.

The software building the graph and exploiting the cycles runs at a standalone risc workstation with 128MB memory and 1GB disk storage capacity. Approaching 100 digits, the needs for memory and disk capacity become substantial. However, it was still not yet necessary to use the possible improvements we could imagine of until now.

4 The Real Parallelisation

The parallel MPQS-implementations described in [3] and [6] were not applicable on our machine. Following Silverman's approach who implémented MPQS in a cluster of independent workstations, every processor would have to work on its own hypercube. But the need to keep $(l+3)R$ integers per hypercube does not leave enough memory for the sieve array with growing R, e.g. with $R = 80.000, l = 10$ this sums up to 4,16 MB. Additionally, every message sent spawns threads on the way taken by itself to the destination processor. The implementation of Lenstra described in [6] was done on a 'Single Instruction Multiple Data' parallel computer. We were glad not to face the difficulties resulting from a single instruction machine. Moreover, a forced synchronisation of all processors in our machine did not seem reasonable to us.

We sketch two parallel approaches particularly suited for MIMD parallel computers. The first one has been used for the results mentioned below, the second one will be the method of choice for even larger numbers. Both methods use a dedicated process (the 'root') whose only tasks consist in the collection of candidates and the input/output-operations. Imagine having more processors than hypercubes to do, it is obvious that we want to enable many processors to work on one hypercube.

In the first implementation the root process runs on a dedicated node. We have two additional types of processors which we will call 'masters' and 'slaves'. Each master creates its own set of hypercubes. After initializing its first hypercube, the first master travels along the hamiltonian cycle described above. At every vertex it computes the new set of modular roots of Q_{ab}. These $2R$ integers and the coefficients of the polynomial are sent to a consecutive set of slaves which will sieve with the same polynomial. After making busy all slaves, the first master leaves its hypercube and initializes the second one. In the meantime the slaves that have finished are at the disposal of the second master, and so on. Each slave sieves its part of the sieve array with the received roots. Any candidates found after the sieving process will be stored in a local buffer. When the buffer overflows its content is sent to the root (see Fig. 1).

Critical parameters here are the number m of masters and the number t of slaves in a consecutive set. These values obviously depend on the number to be

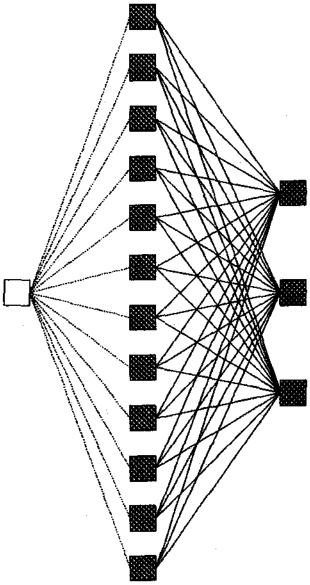

Fig. 1. one root processor, 12 slave processors, three master processors

factored which determines R and M and on the number of $1+m+s$ processors the program will run on. Suppose $T_{cube}, T_{change}, T_{comm}, T_{sieve}$ are the times needed by one processor for the initialisation of one hypercube, for the move along one edge of the hypercube, for one communication and for the sieving, respectively. To avoid any idle times, m and t are then chosen accordingly to:

$$\frac{s}{t}\left(T_{change} + T_{comm}\right) + T_{cube} \leq m \cdot \left(T_{sieve} + T_{comm}\right)$$

This method of parallelisation seems applicable to client-server structured networks, too, because servers typically have enough main memory for the hypercube traversal data.

The demand for easier scalability leads to the second parallelisation idea. Here the root process is running on the front-end computer. All processors of the parallel machine are grouped into rings of r processors each. Every ring works on its own hypercubes. The factorbase is split into r parts of size R/r. Moving from one vertex to another, each processor in a ring first computes its part of the modular roots of the new polynomial Q_{ab}. After $2(r-1)$ communications with its two neighbours involving $2R/r$ integers every processor knows the modular roots of Q_{ab} for the whole factorbase. The sieve array is split into r parts, too. Every processor sieves its part using blocks of predefined length. Any candidates found after the sieving process will be stored in a local buffer as in the first approach (see Fig. 2).

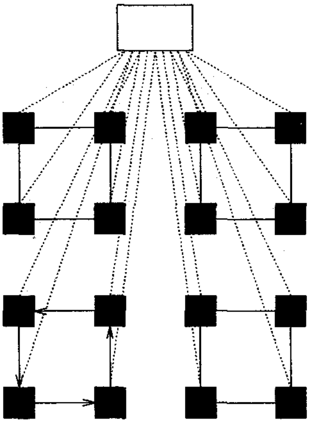

Fig. 2. 16 processors grouped into rings of four

While the amount of data sent among the processors is roughly the same, communicating processors are not as far apart as in the first approach. More precisely, it proved essential to have communication only between physically neighboured processors.

A full description of our implementation concerning the communication aspects will be given in [12].

5 Results and Conclusions

During development of our implementation several numbers in the range from 39 to 91 digits were factorised. However, the factors had been known beforehand and we therefore do not go into detail here.

The first new factorisation is that of the remaining 97 digit cofactor of $12^{327}+1$, namely

20 20744 77159 52927 76009 48240 57414 58126 58099 33659
53308 85724 55927 69199 90416 18031 26379 12970 62041 23709

from the Cunningham list [2]. We found the factors

3 18472 22390 43433 91950 61801 13623 14267 55054 00394 77811 18967

and

6 34512 09239 59276 02546 14461 12941 14196 14827

of 56 respectively 41 digits.

The factorbase contained 40.000 primes, the sieve $9 \cdot 2^{20}$ elements. From 11.400 hypercubes with $l = 7$ we totally used roughly 650.000 polynomials. From 'partial partial relations' we split about 1.100.000 numbers into 'large primes' (less than 2^{32}) that had not been completely factored over the factorbase. Altogether, about 46.000 useful relations were found. Of course, little more than 40.000 would have done.

The overall running time was dominated by the sieving stage taking 38,5 hours at 1024 processors.[4] The next step, the factorisation of large primes for 'partial partial relations', adds approximately 2,5 hours at 1024 processors.[5] The linear algebra could completely be done on a single workstation.

On a second view, the size of the factorbase was rather small (the largest prime in the factorbase was 1.014.649), and the sieve length was too big. These values were chosen mainly because the number of cycles grows nonlinearly with the number of relations produced (see Fig. 3). Hence, a big increase in the number of partial and partial partial relations seems more appropriate than a small gain in the number of full relations and pairs of partial relations originating from a larger factorbase and a smaller sievelength.

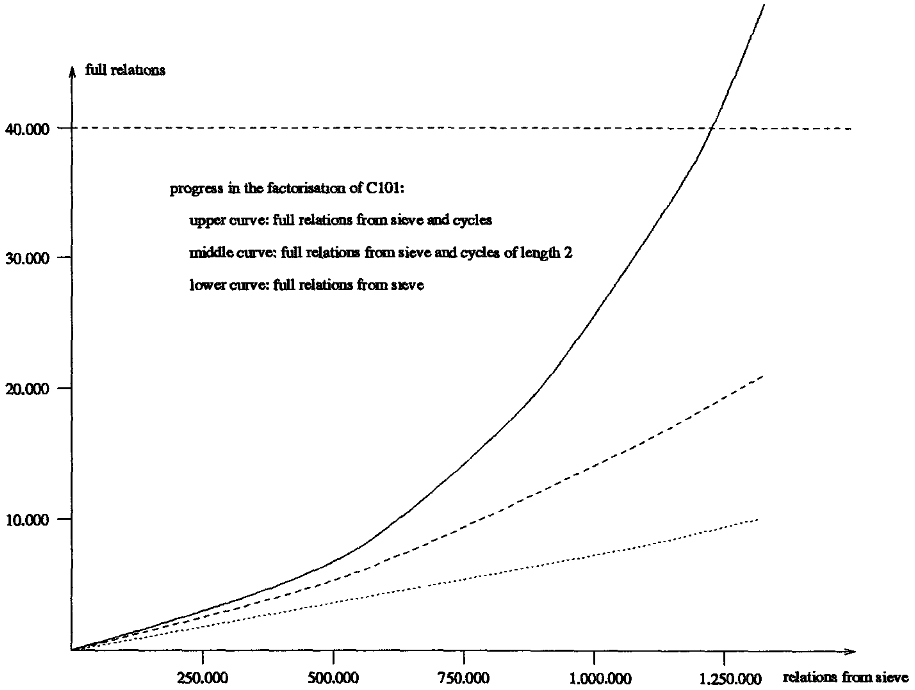

Fig. 3. Growth of the number of cycles in the factorisation of $C101$

[4] Actually, it was performed at 256 processors. In this case, scaling up is linear.

[5] The actually used factors were found at our workstation cluster. The software for the parallel machine then still was under development. Scaling up is very close to linear in this case too.

The graph of $C101$ was investigated in detail. Since we effectively did not use a large prime bound, it was built from 1.328.204 partial and partial relations (9.838 full relations from the sieve) and contained 1.470.729 nodes, 1.275.180 edges and 235.170 connected components. About 431.000 edges built a large star around node 1, while the rest of about 832.000 edges was scattered around. 39.621 cycles were in the component containing node 1. Most of the components only had 1 edge. The cycle lengths were 11.155 × 2 edges, 9.868 × 3, 7.554 × 4, 5.109 × 5, 2.946 × 6, 1.638 × 7, 738 × 8, 343 × 9, 270 cycles containing from 10 to 20 edges.

The main finding is that all the cycles lie in one connected component of the graph. The algorithm does not find a basis of cycles whose lengths are shortest, but this does not seem necessary because the cycles are rather short. Repeated deletion of leaves in the graph takes moderate computing time (25 cutoff steps in about 15 minutes at an ordinary workstation) and produces a much smaller subgraph which is equivalent to the original one when searching and exploiting cycles. We intend to further investigate into the structure of the factor graph [4].

Considering the use of a multiplier, we did not experience the savings in computing time reported in the literature, because we fixed the size of the factorbase in advance.

In [7], Peralta estimated to gain a factor of 25 in the initialisation time for the polynomials. Our version reached a significantly better factor (depending on the choice of l), which we attribute to the faster traverse of the hypercubes. Even the overhead for the initialization of the hypercubes does not compensate our improvement. [9] estimates the time to initialise the polynomials to take about 20–30 % of the overall MPQS running time (working around 100 digits). Using our techniques, this initialisation time becomes very small compared to the total MPQS running time (less than 3% for a typical 100 digit number), which justifies our parallelisation effort. Also, taking full advantage of the hypercube variation would have been impossible if the processors would be stand-alone. Because of these results, we feel encouraged to further deploy the theoretical insights we collected and use and improve on the implementation described here.

6 Acknowledgement

We are very grateful to R. Schrader for generous support. We owe further thanks to M. Behland for his help with the assembler programming, and to the "Zentrum für Paralleles Rechnen" for offering computing time.

References

1. W. R. Alford, C. Pomerance, "Implementing the self initializing qadratic sieve on a distributed network", *Preprint* November 1993.
2. J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr., *Factorizations of* $b^n \pm 1$ *for* b *= 2, 3, 5, 6, 7, 10, 12, up to High Powers.* American Mathematical Society, Providence, Rhode Island, 1983.

3. T. S. Caron, R. D. Silverman, "Parallel Implementation of the Quadratic Sieve", *Journal of Supercomputing*, 1 (1988), pp. 273-290.
4. F. Damm, "Cycle Structures in the Factor Graph of a Composite Number", *in preparation.*
5. A. K. Lenstra, M. S. Manasse, "Factoring with two large primes" (Extended Abstract), *Advances in Cryptology, Eurocrypt '90*, Lecture Notes in Computer Science 473 (1991), pp.72-82.
6. A. K. Lenstra, "Massively Parallel Computing and Factoring", *Proceedings Latin '92*, Lecture Notes in Computer Science 583 (1992), pp.344 - 355.
7. R. Peralta, "A quadratic sieve on the n-dimensional cube", *Advances in Cryptology, Crypto '92*, Lecture Notes in Computer Science 740 (1993), pp.324-332.
8. C. Pomerance, "The Quadratic Sieve Factoring Algorithm", *Advances in Cryptology, Eurocrypt '84*, Lecture Notes in Computer Science 209 (1985), pp.169-182.
9. C. Pomerance, J. W. Smith, R. Tuler, "A pipeline architecture for factoring large integers with the quadratic sieve algorithm", *SIAM Journal of Computation*, Vol.17, No.2, pp.387-403, Apr. 1988.
10. A. Schmidt, G. Wambach, "Parameter Optimisation for Some Variations of the MPQS Algorithm", *in preparation.*
11. R. D. Silverman, "The Multiple Polynomial Quadratic Sieve", *Mathematics of Computation*, Vol.48, No.177, pp.329-339, Jan. 1987.
12. G. Wambach, "A Comparison of Two Parallelisations of the MPQS Algorithm on the Parsytec GCel", *in preparation.*