Hardware to Solve Sparse Systems of Linear Equations over GF(2)

Willi Geiselmann and Rainer Steinwandt

IAKS, Arbeitsgruppe Systemsicherheit, Prof. Dr. Th. Beth,
Fakultät für Informatik, Universität Karlsruhe, Am Fasanengarten 5,
76 131 Karlsruhe, Germany

Abstract. Bernstein [1] and Lenstra et al. [5] have proposed specialized hardware devices for speeding up the linear algebra step of the number field sieve. A key issue in the design of these devices is the question whether the required hardware fits onto a single wafer when dealing with cryprographically relevant parameters.

We describe a modification of these devices which distributes the technologically challenging single wafer design onto separate parts (chips) where the inter-chip wiring is comparatively simple. A preliminary analysis of a 'distributed variant of the proposal in [5]' suggests that the linear algebra step for 1024-bit numbers could be doable on a 23×23-network with special purpose processors in less than 19 hours at a clocking rate of 200 MHz, where each processor has about the size of a Pentium Northwood. Allowing for a 16×16 mesh of processing units with 36 mm $\times$ 36 mm, the linear algebra step might take less than 3 hours.

Keywords: Factorization, number field sieve, linear algebra, RSA

1 Introduction

Nowadays, the most common algorithm for factoring large integers is the so-called number field sieve (NFS). The NFS involves two computationally particularly expensive steps — the relation collection step and the task of solving a large sparse system of linear equations over GF(2) resp. of finding a linear dependence among binary vectors. In this contribution we deal only with the latter step. Based on the block Wiedemann algorithm [4,7], Bernstein [1] and Lenstra et al. [5] recently proposed specialized hardware devices for speeding up this part of the NFS.

In the present form, a major problem of these proposals is the size of the circuits and thereby the question of scalability: for larger parameter values, the proposed circuits do not fit onto a single wafer of diameter 300 mm any more, and high-speed communication between wafers is quite difficult to realize. But having in mind imperfections in actual manufacturing processes, already a single wafer design as proposed in [5] is rather non-trivial to realize. For circumventing this problem, in this paper we propose a technique for distributing the algorithms in [1,5] in such a way onto several wafers, that—at least for the case of 1024-bit numbers—both the performance of the algorithms does not decrease and the inter-wafer communication can be kept rather simple. It is appropriate to mention here that the idea of distributing the linear algebra step onto several 'smaller computers' is not new; e. g., in [2] ideas for implementing the linear algebra step 'in parallel on a network of relatively small machines' are described.

In Section 2 we shortly recall the essential hardware requirements of the two specialized architectures due to Bernstein and Lenstra et al., and thereafter we describe a method for overcoming the hardware limits of these approaches to a certain extent. To get a better idea of the possible use of our approach, Sections 3.2 and 3.3 analyze a 'multi-wafer' variant of the proposals in [1,5] for 512-bit and 1024-bit numbers in more detail. It turns out that even for 1024-bit numbers the linear algebra step seems to be doable within a few hours by means of a distributed hardware that can be manufactured with currently available technology.

2 Two Architectures for the Linear Algebra Step

Within the relation collection step of the NFS a (w. l. o. g. square) sparse matrix $A \in \mathrm{GF}(2)^{m\times m}$ is constructed. For 1024-bit numbers, the estimations in [5, Section 5.1] suggest values of $m \approx 4\cdot 10^7$ or $m \approx 10^{10}$, where on average a column contains about 100 non-zero entries. For representing the matrix A throughout the computations, only the coordinates of its non-zero entries are stored.

To find the linear dependency among the columns of A needed in the NFS, the proposals in [1,5] make use of the block Wiedemann algorithm. Basically, this algorithm reduces the problem of finding a linear dependency among the columns of A to the problem of computing efficiently (long) sequences of the form

$$A \cdot v, A^2 \cdot v, \ldots, A^k \cdot v$$

where v is a—not necessarily sparse—binary vector $v \in \mathrm{GF}(2)^m$. A typical value is $k \approx 2m/K$ with a *blocking factor* $K = 1$ or $K \geq 32$ (for a blocking factor $K > 1$ several different values of the vector v are handled simultaneously).

Accordingly, for reducing the cost of the matrix step in the NFS, the devices proposed by Bernstein and Lenstra et al. aim at reducing the time required for computing such iterated (left-)multiplications with A. While the construction in [1] uses a parallel *sorting* algorithm for this purpose, the proposal in [5] relies on the use of a parallel *routing* algorithm. In the next two sections we shortly recall the respective hardware requirements of these devices; for an explanation of the algorithmic details we refer to the original papers.

2.1 Bernstein's Device for the Matrix Step

Concerning hardware requirements, the essential algorithmic tool in the proposal of [1] is Schimmler's sorting algorithm [1,6]: assume we are given a mesh of

$M \times M$ processing units $(Q_{i,j})_{1 \leq i,j \leq M}$ where $M := 2^n$ and each processing unit $Q_{i,j}$ stores an integer value $q_{i,j}$. Then Schimmler's sorting algorithm allows for sorting these M^2 numbers in $8M - 8$ 'steps' according to any of the following orders on the indices (i, j) of the processing units $Q_{i,j}$:

left-to-right: $(1,1) \leq (1,2) \leq \ldots \leq (1,M) \leq (2,1) \leq \ldots \leq (M,M)$
right-to-left: $(1,M) \leq (1,M-1) \leq \ldots \leq (1,1) \leq (2,M) \leq \ldots \leq (M,1)$
snakelike: $(1,1) \leq (1,2) \leq \ldots \leq (1,M) \leq (2,M) \leq (2,M-1) \leq \ldots \leq (M,1)$

An 'elementary step' of the algorithm looks as follows: analogously as in the odd-even transposition sorting, in a single step each processing unit $Q_{i,j}$ communicates with exactly one of its horizontal or vertical neighbours. So let $\hat{Q}$, $\tilde{Q}$ be two communicating processing units, and denote by $\hat{q}$, $\tilde{q}$ the integers stored in $\hat{Q}$, $\tilde{Q}$, respectively. At the end of one 'elementary step' one of the two processing units, say $\hat{Q}$, must hold the value $\min(\hat{q}, \tilde{q})$ while the other one has to store $\max(\hat{q}, \tilde{q})$. For achieving this one can proceed as follows:

1. $\hat{Q}$ sends $\hat{q}$ to $\tilde{Q}$, and $\tilde{Q}$ sends $\tilde{q}$ to $\hat{Q}$. E. g., if the stored integers represent natural numbers $< 2^{26}$, this operation can be completed in one clock cycle via a unidirectional 26-bit bus in each direction.
2. Both $\hat{Q}$ and $\tilde{Q}$ compute the boolean value *exchange* $:= (\tilde{q} < \hat{q})$. E. g., if $\hat{q}$ and $\tilde{q}$ are 26-bit numbers, this comparison can be done in one clock cycle.
3. If *exchange* evaluates to true, then $\hat{Q}$ stores $\tilde{q}$ and deletes $\hat{q}$. Analogously, $\tilde{Q}$ keeps $\hat{q}$ and deletes $\tilde{q}$, in this case. If *exchange* evaluates to false, then both $\hat{Q}$ and $\tilde{Q}$ keep their old values and delete the values received in the first step. Again, for 26-bit integers this operation does not require more than one clock cycle. In fact it is feasible to integrate this step into the previous one without requiring an additional clock cycle.

In summary, when dealing with natural numbers $< 2^{26}$, Schimmler's sorting algorithm enables us to sort M^2 numbers in less than $8M$ steps where each step takes 2 clock cycles. Assuming that each column of A contains d non-zero entries, one matrix-vector multiplication requires $m \cdot d$ processing units to store the matrix A and m processing units to store the entries of the vector v. Using Bernstein's approach, a matrix-vector multiplication can thus be realized on a mesh of size $M \times M$, provided that $M^2 \geq d \cdot m + m$. For one multiplication three sorting steps with $\approx 8 \cdot M$ exchange operations, requiring 2 clock cycles each, are necessary. Consequently, one matrix-vector multiplication can be performed in approximately $3 \cdot 2 \cdot 8 \cdot M = 48 \cdot M$ clock cycles.

For the factorization of 1024-bit numbers (using the 'small matrix' with $m \approx 4 \cdot 10^7$), in [5] the average number of transistors per processing unit is estimated to be around 2000. Assuming that a standard 0.13 μm manufacturing process is used, with [5, Table 2] we thus conclude, that one processing unit requires an area of $\approx$4760 μm^2 resp. a square of about 0.07×0.07 mm^2. Analogously, assuming a processing unit for the case of 512-bit numbers to require 1800 transistors, we obtain an estimated area of $\approx$4280 μm^2 per processing unit in this case. Here, the estimation for the number of transistors is based on a matrix of size

$6.7 \cdot 10^6 \times 6.7 \cdot 10^6$ where each column contains 63 non-zero entries (cf. [3]); for this matrix size 23 bits are sufficient to represent a column or row index.

2.2 Lenstra et al.'s Device for the Matrix Step

Similarly as Bernstein's proposal, the architecture put forward by Lenstra et al. is based on a mesh of simple processing units. But as opposed to [1], the mesh is used for *routing* rather than *sorting*. Concerning the factorization of 1024-bit numbers, Lenstra et al. discuss two possible matrix sizes ($m \approx 4 \cdot 10^7$ resp. $m \approx 10^{10}$). For describing the device that is to fit onto a single wafer of diameter 300 mm, the 'small matrix' is used, and we restrict our discussion to this case.

Depending on the precise choice of parameters, the mesh [5] uses one or two types of processing units. For the single wafer device just mentioned, only one type is used, and each node is a so-called *target node*. Basically, this means that each node stores all row coordinates of $\rho \geq 1$ non-zero entries of A as well as ρ entries of v. After having performed a complete matrix-vector multiplication $A \cdot v$, the entries storing v are replaced by the entries of the vector $A \cdot v$. Denoting again by d the number of non-zero entries per column, the non-zero entries of A can be distributed onto m/ρ processors where each processor has sufficient DRAM for storing $\rho \cdot d$ matrix entries.

The main tool utilized for the actual computation of a matrix-vector multiplication is so-called *clockwise transposition routing* which relies on the iterated application of (parallelly executed) exchange operations. Routing a single value takes about $2 \cdot \sqrt{m/\rho}$ clock cycles, and the processing of the individual matrix entries and matrix columns can overlap. As a worst-case bound we can assume a complete matrix-vector multiplication to require no more than $\rho \cdot d \cdot 2 \cdot \sqrt{m/\rho} = 2 \cdot d \cdot \sqrt{m \cdot \rho}$ clock cycles.

So far, our discussion ignored the blocking factor K: as pointed out in [5], for a given blocking factor K, Wiedemann's algorithm requires the computation of K multiplication chains

$$A \cdot v_i, A^2 \cdot v_i, \ldots, A^k \cdot v_i$$

with different vectors v_i ($1 \leq i \leq K$). Using a slightly more complicated hardware (see [5] for details), these K chains can be computed in parallel with the same routing circuit. Basically, for blocking factor K each processing unit needs $2 \cdot \rho \cdot K$ bit of memory to store the vectors v_i. In particular, the value of K is relevant when estimating the space requirement for the target units; here we assume $K = 208$ (the value chosen in [5] for the single wafer device for 1024-bit numbers).

With $m \approx 4 \cdot 10^7$ the average number of transistors necessary for one target unit—excluding DRAM—can be estimated to be around $2040 + 60 \cdot K$ (cf. Section 3.2). The area needed for a single DRAM bit is ≈ 0.7 μm^2 resp. 0.2 μm^2 with a specialized DRAM process. Thus, for the 'small matrix case' of 1024-bit numbers, the DRAM of a complete target cell occupies about 88700 μm^2 resp. 25300 μm^2 with a specialized DRAM process. The space requirement for the

$2040 + 60 \cdot 208$ transistors computes to 34600 μm^2 and 40700 μm^2, respectively. In total, for one target cell 123300 μm^2 with a standard process resp. 66000 μm^2 with a specialized DRAM process are needed in the 1024-bit case. When dealing with 512-bit numbers—and a matrix of size $6.7 \cdot 10^6 \times 6.7 \cdot 10^6$ with 63 non-zero entries per column—the DRAM per target cell occupies only 54800 μm^2 resp. 15700 μm^2 with a specialized DRAM process. Adding the space for $1920+60\cdot 208$ transistors, we obtain a total space requirement of 89100 μm^2 resp. 56100 μm^2 per target cell.

2.3 Estimated Mesh Size and Performance for 512 Bit and 1024 Bit

With a standard 0.13 μm process, there fit $\approx 2.915 \cdot 10^{10}$ transistors on a single wafer of diameter 300 mm (cf. [5]). Using the above figures, a straightforward computation now yields the following estimation for the wafer area and time needed by Bernstein's approach, when dealing with $\log_2(n)$-bit numbers:

$\log_2(n)$	m	d	# proc	M	area in wafers	clock cycles	LA
512	$6.7 \cdot 10^6$	63	$4.3 \cdot 10^8$	2^{15}	74 (26.6)	$1.6 \cdot 10^6$	18 h
1024	$4 \cdot 10^7$	100	$4.04 \cdot 10^9$	2^{16}	295 (277.2)	$3.1 \cdot 10^6$	207 h

The area in brackets indicates the area required to store the matrix and the vector; the difference to the real area required comes from choosing M as a power of 2 with $M^2 \geq m \cdot (d + 1)$. The last column (labeled with LA) is the estimated total time of the linear algebra step; more precisely, the value given is the time for performing $3 \cdot m$ matrix-vector multiplications (see [5]) at a clocking rate of 500 MHz.

A 512-bit device with these parameters has a size of 2.14 m×2.14 m; for the 1024-bit case we obtain a (wafer) area of 4.5 m×4.5 m. Thus, realizing such a device seems quite hypothetical.

For the device described by Lenstra et al. [5] we assume $\rho = 42$, i. e., each target unit takes care of 42 matrix columns (cf. [5, Table 3]). For 512-bit numbers, then 402^2 target units including DRAM fit on an area of 95 mm×95 mm; and for 1024-bit numbers 1026^2 target units including DRAM fit on the area of a single wafer. This results in the following estimated space and time requirements for performing the linear algebra step:

$\log_2(n)$	m	d	# targets	M	area (wafers)	clk cycles	LA
512	$6.7 \cdot 10^6$	63	$1.6 \cdot 10^5$	400	0.13	$2.1 \cdot 10^6$	17 min
1024	$4 \cdot 10^7$	100	$9.5 \cdot 10^5$	1024	1	$8.2 \cdot 10^6$	6.5 h

Here the estimatated total time of the linear algebra step is the time for performing $3 \cdot m/K = 3 \cdot m/208$ matrix-vector multiplications at a clocking rate of 200 MHz (cf. [5]).

3 Distributing the Computation

In several of the above mentioned sizes and in several other parameter choices described in [5], the specialized hardware for the linear algebra step does no longer fit on a single wafer. But due to the practical limitations of manufacturing processes, already realizing the single wafer devices is quite challenging. In this section we want to discuss an approach for circumventing the problem of handling sophisticated, highly parallel I/O hardware for fast inter-wafer communication; at least for 1024-bit numbers this approach seems to improve the situation significantly.

3.1 Using Block Matrix Multiplication

For our discussion we adopt the assumption from [5] that the non-zero entries in the matrix $A \in \mathrm{GF}(2)^{m\times m}$ are uniformly distributed. It should be emphasized, that the original matrix A cannot be expected to have such a uniform distribution, and here we do not discuss the problem of how a preprocessing for achieving this could look like. The 'rectangular matrix blocks' we will use should allow for some leeway here, and subsequently we make the assumption that a suitable preprocessing has been done already, e. g., by having applied suitable row and column permutations to the original A.

We start by splitting the matrix A into $s \cdot s$ submatrices $A_{i,j}, 1 \leq i, j \leq s$ of approximately the same size of $m/s \times m/s$. It is not mandatory that all $A_{i,j}$ are square matrices, but we insist that for fixed $i_0 \in \{1, \dots, s\}$ all matrices $A_{i_0,j}$ $(1 \leq j \leq s)$ have the same number of rows:

$$A = \left(\begin{array}{cccc} A_{1,1} & |\, A_{1,2}| & \dots & |A_{1,s} \\ \hline A_{2,1} & |A_{2,2}\,| & \dots| & A_{2,s} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline A_{s,1} & |\, A_{s,2}| & \dots & |A_{s,s} \end{array}\right)$$

The size of the hardware devices in [1,5] depends on the number of non-zero entries in the processed matrix, and the aim of the separation just mentioned is to split the matrix into s^2 submatrices with approximately the same number of non-zero elements. After splitting the vector v into appropriately sized parts

$$v = \begin{pmatrix} v_{1,1} \\ \vdots \\ v_{1,s} \end{pmatrix} = \dots = \begin{pmatrix} v_{s,1} \\ \vdots \\ v_{s,s} \end{pmatrix}$$

(where the number of rows of $v_{i,j}$ is equal to the number of columns of $A_{i,j}$), the multiplication $A \cdot v$ can be realized as

$$A \cdot v = \begin{pmatrix} \sum_{j=1}^{s} A_{1,j} \cdot v_{1,j} \\ \vdots \\ \sum_{j=1}^{s} A_{s,j} \cdot v_{s,j} \end{pmatrix}.$$

This can be performed with s^2 multiplication circuits (preloaded with the matrices $A_{i,j}$) in the following way:

1. Load $v_{i,j}$ into the circuit corresponding to $A_{i,j}$ through a pipeline of length s and a bus of width b (if $A_{i,j}$ is not a square matrix, we can think of the missing column/row entries as being 0).
2. Perform the s^2 matrix-vector multiplications $A_{i,j} \cdot v_{i,j}$ in parallel.
3. Output the resulting vectors $A_{i,j} \cdot v_{i,j}$ through a bus of width b.
4. Perform the summations $w_i := \sum_{j=1}^{s} A_{i,j} \cdot v_{i,j}$ (for $i = 1, \ldots, s$) with s XOR-pipelines of length s. Each of these s^2 XOR-circuits is adjacent to one multiplication circuit and adds the output of this circuit to the output of the previous stage of the pipeline. Each of these XOR-circuits has two inputs and one output of width b and works during the output of the multiplication circuits in a pipeline architecture.
   These XOR-circuits are extremely simple, but have to be built up out of several chips due to the width of the bus. A different approach is to include these XORs into the adjacent multiplication circuit; then the saved hardware has to be paid for by a doubling of the I/O time. This part of the hardware should not cause a major problem and is neglected here.
5. Analogously as the vector v before, now the vector $w := (w_1, \ldots, w_s)$ is split into s^2 parts $w_{i,j}$. These $w_{i,j}$ are now ready to be loaded into the s^2 multiplication circuits to perform the multiplication $A \cdot w$ if required.
   At this stage we can also easily perform a vector-vector multiplication of the form $u \cdot Av = u \cdot w$ as needed in the block Wiedemann algorithm. The vectors u are usually chosen to be of very low Hamming weight, and we thus ignore the (marginal) computational effort of these multiplications.
   The loading of $A \cdot v$ is performed through a pipeline structure, similar to the XOR-pipeline for the outputs. If the XOR-circuits are extended with an additional register and a multiplexer (to switch between the 'horizontal' and 'vertical' bus), the same chips can be used for both pipelines.

3.2 Performance of the Distributed Device

Let us now look at the space and time requirements of the distributed architecture just described; for sake of simplicity, we assume all submatrices $A_{i,j}$ to be $(m/s) \times (m/s)$ square matrices. We also consider the choice of a blocking factor $K \geq 1$; for $K > 1$ several vectors are handled in parallel, and of course we have to take into account the additional bandwidth required here.

Step 1. Loading the K vectors $v_{i,j}$ into the multiplication circuits requires approximately $4 \cdot m \cdot K/(s \cdot b) + 4 \cdot s$ clock cycles, where the time for loading one bit is estimated to be 4 clock cycles, and $4 \cdot s$ clock cycles are needed to empty the pipeline. All but the last b input bits can be distributed to the processing units (or target cells) while the following input bits arrive. The extra time required to distribute the last b bits is neglected here.

Step 2. Each of the multiplication circuits has to perform a multiplication of a matrix with about $m \cdot d/s^2$ entries with a binary vector of size approximately m/s. With Bernstein's approach, this can be done in $48 \cdot M$ clock cycles on an $M \times M$ mesh where $M \geq \sqrt{m \cdot d/s^2 + m/s}$ is a power of 2.

With the design from [5], this matrix-vector multiplication requires M^2 target cells for ρ columns each, where $M^2 \geq m/(s \cdot \rho)$. At this, each target cell is equipped with DRAM for $\rho \cdot d/s$ matrix entries, and we can estimate the number of clock cycles required for the matrix-vector multiplication to be no larger than $2 \cdot \rho \cdot d \cdot M/s$.

Step 3. Transfering the vector w_i to the output buffer requires one sorting step in Bernstein's architecture ($\approx 2 \cdot 8 \cdot M$ clock cycles). In the architecture of Lenstra et al. the computational effort of this step is negligible (due to the known addresses of the bits of w_i, and thus the possibility to use a pipeline procedure during the output); we estimate the output to require $\approx 4 \cdot m \cdot K/(s \cdot b)$ clock cycles.

Step 4. The summation of the subvectors is performed with a pipeline structure while the outputs of the multiplication units arrive. Additional $\approx 4 \cdot s$ clock cycles are needed to empty the pipeline.

Summarizing our discussion, we have the following characterizing figures:

– With Bernstein's architecture M^2 processing units for each of the s^2 parts are required, where M is a power of 2 and $M^2 \geq m \cdot d/s^2 + m/s$. Taking into account the registers ($2 \cdot 8 \cdot \lceil \log_2(m/s) \rceil$ transistors), multiplexers ($3 \cdot 4 \cdot \lceil \log_2(m/s) \rceil$ tansistors), a subtraction unit ($5 \cdot 8 \cdot \lceil \log_2(m/s) \rceil$), and control logic (300 transistors) needed, we estimate that one processing unit consists of $\approx 68 \cdot \lceil \log_2(m/s) \rceil + 300$ transistors.[1]

  The number of clock cycles for a complete matrix-vector multiplication is approximately $8 \cdot \lceil m/(s \cdot b) \rceil + 48 \cdot M + 8 \cdot s$.

– With the architecture of Lenstra et al. $M^2 \geq m/(s \cdot \rho)$ processing units resp. target cells are used on each of the s^2 parts, if one target cell takes care of ρ matrix columns. Taking into account the required DRAM for representing the non-zero matrix entries ($\rho \cdot d \cdot \lceil \log_2(m/s) \rceil / s$ bit), the DRAM bits for storing the K processed vectors ($2 \cdot \rho \cdot K$ bit) along with an access logic ($40 \cdot K$ transistors), a register for a received 'package' from the mesh ($8 \cdot (\lceil \log_2(m/s) \rceil + K)$ bit), three multiplexers ($3 \cdot 4 \cdot (\lceil \log_2(m/s) \rceil + K)$ transistors), a subtraction unit ($(8 \cdot 5 \cdot \lceil \log_2(m/s) \rceil)/2$ transistors), and additional logic (1000 transistors) we estimate one processing unit to require $\approx 40 \cdot \lceil \log_2(m/s) \rceil + 60 \cdot K + 1000$ transistors and $\rho \cdot (d \cdot \lceil \log_2(m/s) \rceil / s + 2 \cdot K)$ bit of DRAM. The number of clock cycles for a complete matrix-vector multiplication is $\approx 8 \cdot \lceil m \cdot K/(s \cdot b) \rceil + 2 \cdot \rho \cdot d \cdot M/s + 8 \cdot s$.

In the next section we examine in more detail the performance of this distributed approach when dealing with 512-bit and 1024-bit numbers. For doing so, we consider various choices of the bus width b, the blocking factor K, the number of columns ρ handled per target cell, and the 'degree of parallelism' s.

[1] Note that for storing a row or column index of a submatrix $A_{i,j} \in \mathrm{GF}(2)^{m/s \times m/s}$ only $\lceil \log_2(m/s) \rceil$ bits are needed.

3.3 Application to 512-Bit and 1024-Bit Numbers

Having in mind a practical manufacturing process, it is desirable that the individual parts of the distributed circuit are significantly smaller than (the inner square of) a complete 300 mm wafer. For the distributed circuit derived from the architecture in [5], we choose the size of the individual parts to be comparable to the size of an 'ordinary' Pentium Northwood processor. To avoid problems with the available number of pins connected to the bus b, we use somewhat conservative estimations for the bus width.

For Bernstein's circuit such small processing units are not really sensible, and we choose the individual parts to be larger. For these larger parts (or in other words chips), it is sensible to allow for a larger bus width b, as more pins can be located on the chip here.

The bus width b also limits the possible choices of the blocking factor K: before performing the (next) matrix-vector multiplication by means of a (routing or sorting) mesh, we have to load the respective parts of the vector to be processed next into the processing units via the bus. However, with a simple trick we can gain some parallelism 'almost for free': assume that each part of the distributed device—in other words each chip—handles K vectors in parallel (for Bernstein's approach we have $K = 1$). Then while these K vectors are processed, we can load another K-tuple of vectors into a separate buffer on that chip. So once the result of the previous multiplication is output, we can immediately load the new vectors into the mesh. If the I/O time is about the same as the computation time, then by interleaving the processing of two 'tuples of vectors' in this way, we can in the ideal case almost halve the time needed for loading vectors onto and from the chips (of course, the cells to store these additional vectors require additional place on the chip, which has to be taken into account then).

Table 1 and 2 show the performance of the distributed device for various parameter choices; at this, a potential optimization by 'interleaving tuples of vectors' is not taken into account. As in Section 2.3, for estimating the total time of the linear algebra step, the number of multiplications is assumed to be $3 \cdot m$ in the design derived from Bernstein's proposal, and $3 \cdot m/K$ in the design derived from the proposal of Lenstra et al.

Table 1. Time for the LA step with a 'distributed Bernstein design' at a clocking rate of 500 MHz.

$\log_2(n)$	b	#proc/chip	s^2	chip size	LA time
512	(single unit)	23768^2	1	2.14 m × 2.14 m	17.6 h
512	2048	2048^2	11^2	144 mm × 144 mm	1.1 h
512	1024	1024^2	24^2	72 mm × 72 mm	0.6 h
512	1024	512^2	55^2	36 mm × 36 mm	0.3 h
1024	(single unit)	65536^2	1	4.5 m × 4.5 m	210 h
1024	2048	2048^2	37^2	144 mm × 144 mm	6.8 h
1024	1024	1024^2	84^2	72 mm × 72 mm	3.4 h

Table 2. Time for the LA step with a 'distributed Lenstra et al. design' at a clocking rate of 200 MHz.

$\log_2(n)$	b	K	ρ	#proc/chip	s^2	chip size	LA time
512	128	65	116	76^2	10^2	11.4 mm × 11.4 mm	73 min
512	128	53	51	91^2	16^2	11.4 mm × 11.4 mm	45 min
512	512	63	29	152^2	10^2	20 mm × 20 mm	19 min
512	1280	49	8	290^2	10^2	34 mm × 34 mm	8 min
512	1024	40	6	265^2	16^2	29 mm × 29 mm	6 min
1024	(single unit)	208	42	975^2	1	265 mm × 265 mm	6.1 h
1024	(single unit)	42	216	430^2	1	162 mm × 162 mm	94.7 h
1024	128	30	1086	48^2	16^2	11.4 mm × 11.4 mm	29.7 h
1024	128	70	669	51^2	23^2	11.4 mm × 11.4 mm	18.8 h
1024	512	100	250	100^2	16^2	20 mm × 20 mm	7.0 h
1024	1024	160	278	120^2	10^2	30 mm × 30 mm	5.9 h
1024	1280	135	66	195^2	16^2	36 mm × 36 mm	2.8 h

For Bernstein's approach we recognize that the distributed variant looks much more practical than the original design. Also it is worth noting, that the communication cost—i. e., the time for loading vectors onto/from the chips—is less than 5% of the overall computation time, and thus is not really relevant. For the design of Lenstra et al. the situation is quite complementary: more than 90% of the time is spent for the I/O operations. However, the obtained circuitry is much smaller and thus more realistic than the sorting based approach. In particular, with a mesh of $23^2 = 529$ Pentium Northwood sized processing units, the linear algebra step for a 1024-bit number should be doable in less than 19 hours. Note here that the overall wafer area of this distributed device is the same as for the original single wafer design of Lenstra et al. Concerning speed the distribution has to be paid for with a slow-down of more than a factor 3. However, manufacturing the small processing units is significantly simpler. Further on, already with slightly larger processing units—which allow for a broader bus—, the overall computation time can be reduced to less than 3 hours. As most of the time is spent for the I/O, the bus width should be chosen as large as possible; in our estimations we tried to be conservative here.

4 Conclusion

The above discussion suggests that for 1024-bit numbers, a 'distributed variant of the design of Lenstra et al.' could be realizable by means of current technology. Besides circumventing a technologically challenging wafer-sized circuit, also a speed-up seems to be possible, if one allows for processing units of up to, say, 36 mm × 36 mm. But already with a mesh of 23^2 processing units, where each processing unit has approximately the size of a Pentium Northwood, the linear algebra step for 1024-bit numbers seems to be doable in less than a day.

Acknowledgement. We thank Eran Tromer for valuable discussions and remarks.

References

1. Daniel J. Bernstein. Circuits for Integer Factorization: a Proposal. At the time of writing available electronically at `http://cr.yp.to/papers/nfscircuit.pdf`, 2001.
2. Richard P. Brent. Recent Progress and Prospects for Integer Factorisation Algorithms. In Ding-Zhu Du, Peter Eades, Vladimir Estivill-Castro, Xuemin Lin, and Arun Sharma, editors, *Computing and Combinatorics; 6th Annual International Conference, COCOON 2000*, volume 1858 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.
3. Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter Lioen, Peter L. Montgomery, Brian Murphy, Herman te Riele, Karen Aardal, Jeff Gilchrist, Gérard Guillerm, Paul Leyland, Joël Marchand, François Morain, Alec Muffet, Chris Putnam, Craig Putnam, and Paul Zimmermann. Factorization of a 512-bit RSA Modulus. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.
4. Don Coppersmith. Solving Homogeneous Linear Equations over GF(2) via Block Wiedemann Algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.
5. Arjen K. Lenstra, Adi Shamir, Jim Tomlinson, and Eran Tromer. Analysis of Bernstein's Factorization Circuit. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2002.
6. Manfred Schimmler. Fast sorting on the instruction systolic array. Technical Report 8709, Christian Albrecht Universität Kiel, Germany, 1987.
7. Douglas H. Wiedemann. Solving Sparse Linear Equations Over Finite Fields. *IEEE Transactions on Information Theory*, 32(1):54–62, 1986.