

# **A special purpose mesh architecture for sieving in the number field sieve**

Willi Geiselman

Rainer Steinwandt

Universität Karlsruhe, Germany



# Dedicated hardware for the NFS

## Linear algebra step:

Implement (iterated) matrix-vector multiplications on special device to speed up (block) Wiedemann algorithm.

## Sieving step:

Implement (line) sieving on specialized hardware to allow for quick identification of candidate  $(a,b)$ -pairs.

**Main focus:** Can 1024-bit numbers "in principle" be handled by means of "realistic" hardware?



# A design for the linear algebra part

## D. Bernstein 2001:

Reduction of matrix-vector multiplications to parallel sorting problems that can be solved on a mesh of **simple** processing units.

feasibility in hardware

**Problem:** Already for 512-bit numbers such a device spreads over several (standard 300 mm silicon) wafers and requires inter-wafer communication



Direct implementation of this idea not practical.



# A circuit design based on routing

**A. Lenstra, A. Shamir, J. Tomlinson, E. Tromer 2002:**  
Reduce matrix-vector multiplication to a parallel routing problem which can be implemented on a mesh of simple processing units.

**Technological idea:** compact storage of matrix using a DRAM process → avoid inter-wafer communication.

CHES 2003: can be split onto "small" chips

**Algorithmic idea:** very efficient routing through so-called **clockwise transposition routing**



# Clockwise transposition routing

## Hardware:

- mesh of simple processing units (compare-exchange)
  - simple control logic, only local communication
- ➔ well-suited for hardware implementation

## Performance:

- complete theoretical analysis still lacking
- experimentally extraordinary efficient

several variants are possible in the linear algebra and sieving step

**Used parameters are optimized "only" experimentally.**



# Designs for the sieving step (I)

## A. Shamir 1999 / A. Lenstra, A. Shamir 2000:

TWINKLE: - could be used to sieve for 768-bit numbers  
- technologically challenging opto-electronic hybrid design, expensive GaAs technology

## PKC 2003:

"Sorting mesh ": - based on parallel sorting network similar to Bernstein's proposal for the LA step  
- number of processors  $>$  factor base size  
- handling of small primes troublesome  
- not suitable for numbers  $>512$ -bit



# Designs for the sieving step (II)

**A. Shamir, E. Tromer 2003:**

"standard chip":  $(1.14\text{cm})^2$

- TWIRL: - storage of rational and algebraic factor base on separate (connected) wafers
- proposed chip sizes for 768-bit: up to  $(6.7\text{cm})^2$
  - irregular layout, but can in principle handle factor bases for 1024-bit numbers (with a wafer-sized chip for storing the algebraic factor base)

**Is it possible to use a "simpler", more regular layout?**



# Yet another sieving device

**Hardware:** "optimized" routing mesh

- torus instead of rectangle  
(implemented via interleaving of processors)
- use additional metal layers to "overlay"  
four smaller meshes

**Algorithm:** variant of clockwise transposition routing

- meshes are refilled during routing
- experimentally quite fast, theoretical analysis  
lacking

heuristics avoid  
congestion

"a loop once in a while"  
does not matter



# Basic organization of the sieving

1024-bit: to be explored

**Focus: 768-bit numbers (with estimates from TWIRL)**

- **Rational factor base:** primes  $< 10^8$
- **Algebraic factor base:** primes  $< 10^9$
- **Algebraic & rational side** are dealt with **simultaneously**
- Sieving uses **approximations**  $\lceil \log_{\sqrt{2}}(p) \rceil$
- "Interval-wise" line sieving:  **$2^{24}$  sieve locations per run;**  
**new data is to be loaded when changing the sieve line**
- **Report "big" (e.g.  $> 2^{22}$ ) factors** found during sieving



# Storing the factor base

Efficient DRAM representation of factor base (along with efficient, regular access) desirable to reduce chip size.



Represent tiny ( $2 \leq p < 2^{17}$ ), smallish ( $2^{17} \leq p \leq S (=2^{24})$ ), largish ( $S < p \leq B_r (=10^8)$ ), hugish ( $p > B_r$ ) primes differently

## "Tuning" the representation:

- take care of varying number of algebraic roots
- handle successive largish/huge primes in one processor  $\rightarrow$  storing (half of) the difference is sufficient



# Example: largish primes

$\Delta p/2$  ( $\neq 0$ , 7 bit)



rational root (27 bit)	$b_{27}$ (1 bit)
1 <sup>st</sup> algebraic root (27 bit)	$b_{27}$ (1 bit)

...

0...0 (7 bit)



$(p-1)/2$ (26 bit)	$l_p$ (2 bit)
rational root (27 bit)	$b_{27}$ (1 bit)
1 <sup>st</sup> algebraic root (27 bit)	$b_{27}$ (1 bit)

...

1...1 (7 bit)



(end of largish primes)



# Assigning sieve locations (I)

Each sieving location in an interval of length  $S(=2^{24})$  is assigned to a processor in a  $2^m \times 2^m (=256 \times 256)$ -mesh.



each processor in charge of  $2^8$  consecutive sieve positions  
 $\{s_0 + i \cdot 256, s_0 + i \cdot 256 + 1, \dots, s_0 + i \cdot 256 + 255\}$  ( $0 \leq i \leq 2^{16} - 1$ )

least significant 8 bit of 24-bit "remainder"  $0 \leq r < S$   
identify the sieve location within a processor.

**How to find (i.e., route to) the correct processing unit?**



# Assigning sieve locations (II)

$x$ -coordinate of target: odd-numbered bit positions

$y$ -coordinate of target: even-numbered bit positions

Kronecker/tensor product structure

00	02	08	10	32	34	40	42
01	03	09	11	33	35	41	43
04	06	12	14	36	38	44	46
05	07	13	15	37	39	45	47
16	18	24	26	48	50	56	58
17	19	25	27	49	51	57	59
20	22	28	30	52	54	60	62
21	23	29	31	53	55	61	63

node handling  
interval no. 58



# Which prime is stored where?

Idea: route "locally" to keep distances short



store factor base elements  $(p,r)$  "repeatedly"

$p \leq 2^8$ : each node stores  $(p, (s_0 + r + i \cdot 2^8) \bmod p)$

"number"  
of node

$2^8 < p \leq 2^{10}$ : store  $(p, (s_0 + r + j \cdot 2^{10}) \bmod p)$  once per  $2 \times 2$  square

$2^{10} < p \leq 2^{12}$ : taken into account once per  $4 \times 4$  square

⋮

$2^{18} < p \leq 2^{20}$ : taken into account once per  $64 \times 64$  square

$p > 2^{20}$ : stored only once

torus topology  
 $\Rightarrow$  no  $128 \times 128$   
submeshes



# Structure of the processors

Basically, each processor consists of three parts:

**Main part:**

- contains ( $\approx 1300$ ) factor base pairs
- updates  $r$ -values & if appropriate puts them (+info on  $p$ ) into an output buffer

**Memory part:**

- controls rational & algebraic  $\lceil \log_{\sqrt{2}}(p) \rceil$ -counters for each sieve location
- logs factors received on the mesh

**Mesh part:**

- controls mesh input/output



# Role of main part

Read  $(p, r)$ -pairs from memory & check "relevance"

- updates current  $\lceil \log_{\sqrt{2}}(p) \rceil$ -value

- "irrelevant"  $r$ : update  $r$  for the next interval:

$$r := r - (S \bmod p) [+p \text{ if negative}]$$

(simplified)

- "relevant"  $r$ : find target node & write  $(p, r)$  to output buffer

proceed analogously with  $r = r + p$

## What if the target node is the node itself?

→ write data to input buffer, as if received from the mesh



# Role of memory part

Obtains packets from mesh through input buffer

- updates corresponding rational/algebraic  $\lceil \log_{\sqrt{2}}(p) \rceil$ -counter
- stores "big" found prime factors in DRAM  
(shared memory with physical neighbour)
- if algebraic and rational smoothness thresholds are reached: respective factors are "collected and prepared for output"

During "cleaning" no new factors can be stored.



# Output of the result

After completion of a subinterval, "successful" nodes report their hits (including found factors on both sides) via the available (59-bit) bus

**When is a subinterval complete?**

→ use an (experimentally determined) bound for the number of clock cycles & enforce missing  $r$ -updates

Next interval in the **same line**: just **reset counters**

**Next sieving line**: new data has to be loaded ( $\approx 0.02\text{s}$ )



# Estimated performance (768-bit)

- estimated **size** (with  $0.13\mu\text{m}$ ) of  $256\times 256$ -mesh:  $(4.9\text{cm})^2$   
→ still much larger than standard chip, but smaller than proposed TWIRL chips (algebraic sieve:  $(6.7\text{cm})^2$ )
- clock **speed**: 500 MHz (1 GHz of TWIRL is not doable here)
- **exclusion of  $(a,b)$ -pairs** with  $2 \mid \text{gcd}(a, b)$  possible  
(TWIRL also excludes  $3 \mid \text{gcd}(a, b)$ ; this does not apply here)



**1 wafer needs  $\approx 600$  days – ca. 6.3 times more than TWIRL**



# Conclusion

At least for 768-bit numbers, a mesh-based design could offer an interesting alternative to TWIRL:

- smaller chip size applies also to 1024-bit case
- comparatively simple and regular layout

**But:** - slowdown by a factor of  $\approx 6.3$

- "realistic" chip size & run-time for 1024-bit possible?
- theoretical analysis of clockwise transposition routing (variant) still lacking

