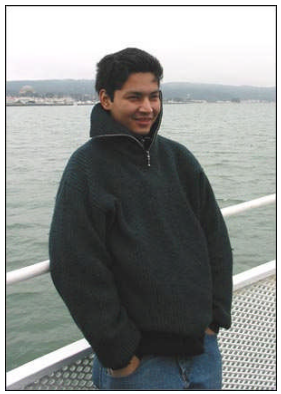
Reconfigurable Hardware Implementation of Mesh Routing in Number Field Sieve Factorization

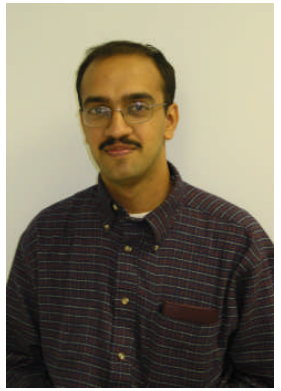


Sashisu Bajracharya,

Deapesh Misra,

Kris Gaj

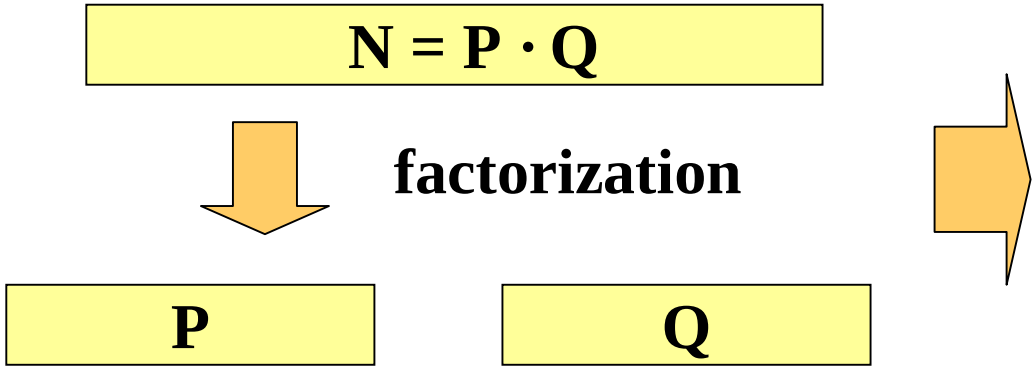
George Mason University



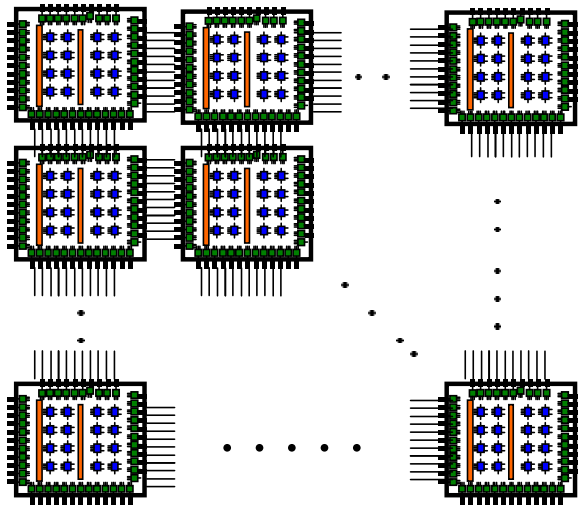
Tarek El-Ghazawi

The George Washington University

Objective & Outline



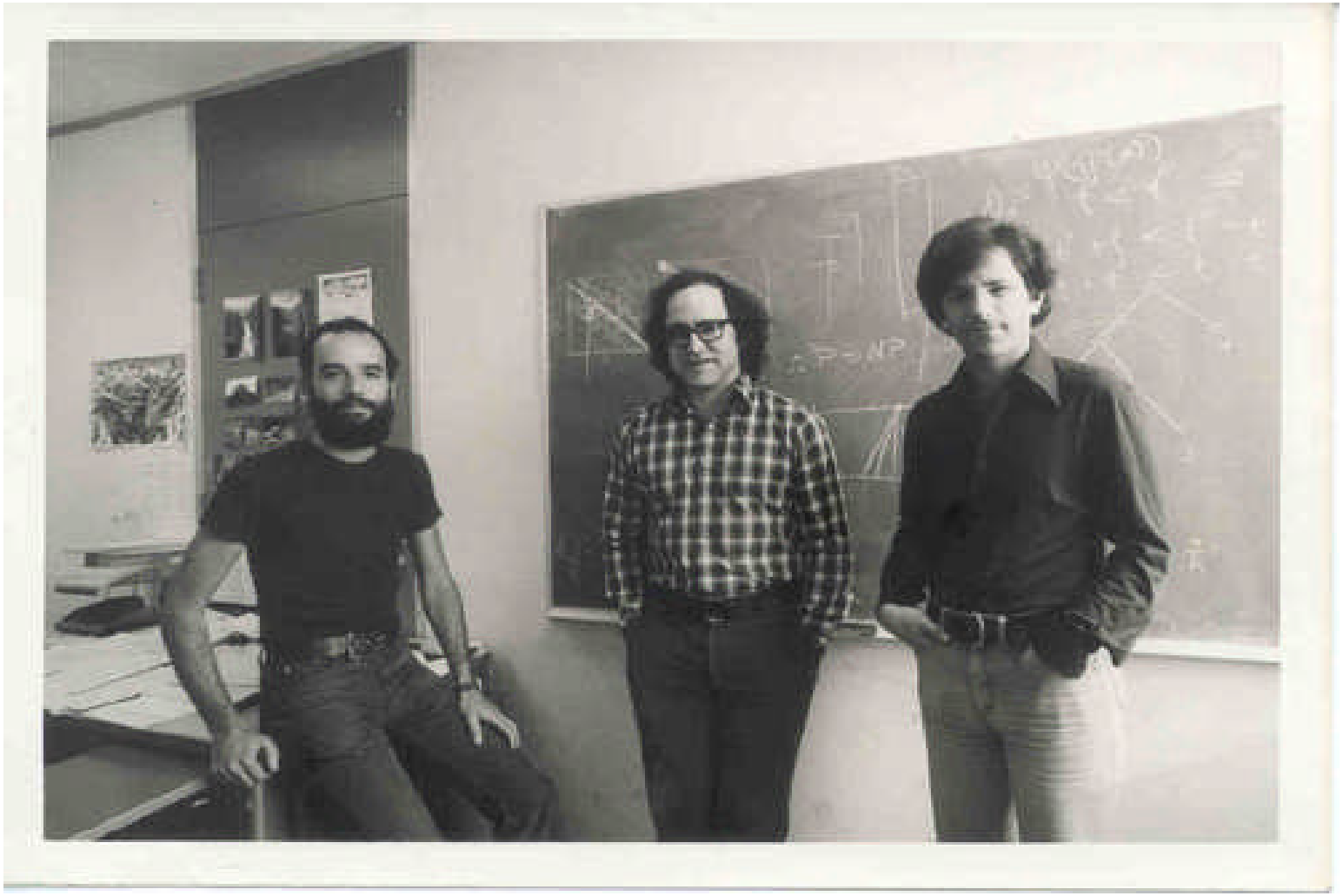
P, Q – large integers



FPGA Array

1. RSA and Factoring
2. Number Field Sieve (NFS) Factorization
3. Mesh Routing Architecture for the Matrix Step of NFS
4. Results and Analysis
5. Conclusions

RSA & Factoring



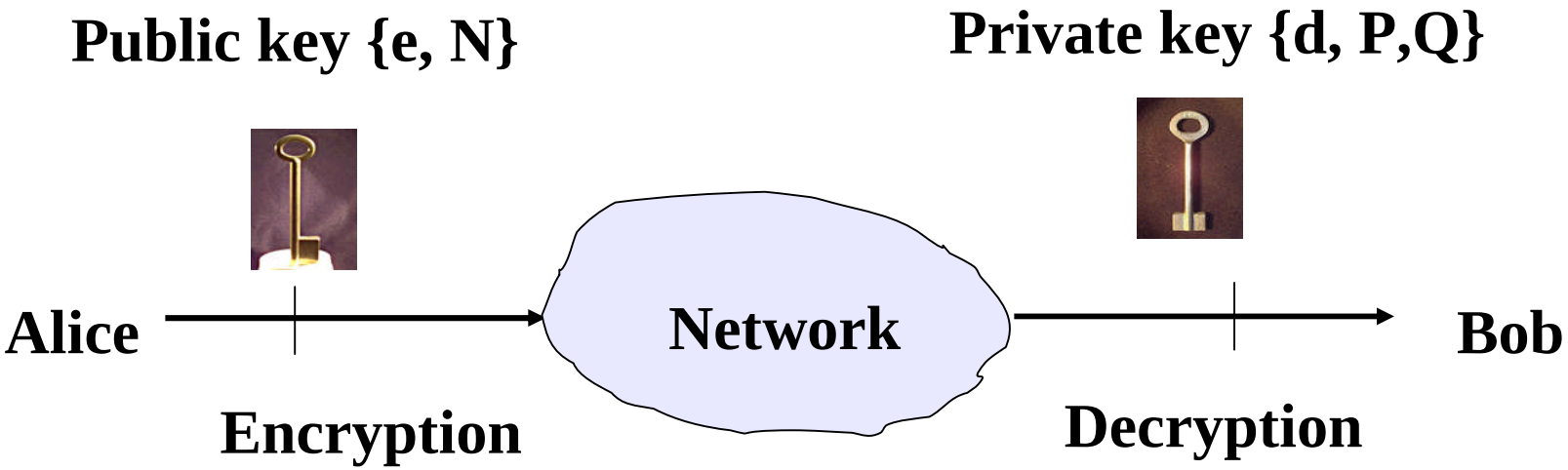
In 1977 by

Ron Rivest, Adi Shamir & Leonard Adleman

developed the first public key cryptosystems, they called RSA

RSA

Major Public Key Cryptosystem



{ e, N } ⟵ ✕ ⟶ **{ d, P, Q }**

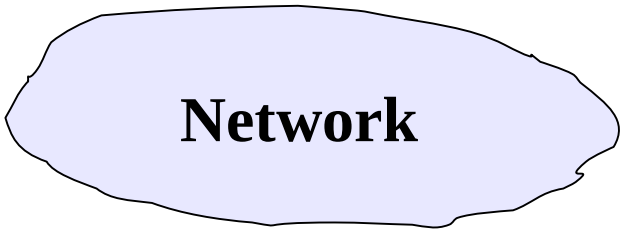
N = P·Q **P, Q - large prime factors**

$$e \cdot d \equiv 1 \mod ((P-1)(Q-1))$$

Common Applications of RSA

Secure WWW, SSL

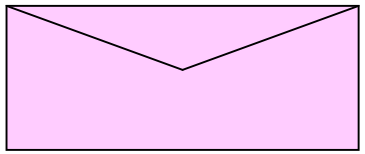
Browser



WebServer

S/MIME, PGP

Alice



Bob

Recommended key sizes for RSA

Size of the RSA key: size of N=P·Q

Old standard:

Individual users ~~**512 bits** (155 decimal digits)~~

New standard:

Short-term use (up to 2010) **1024 bits**

Long-term use **2048 bits**

Factoring 512-bit number

512 bits = 155 decimal digits

old standard for key sizes in RSA

17 March - 22 August 1999

Group of Herman te Riele
Centre for Mathematics and Computer Science
(CWI), Amsterdam

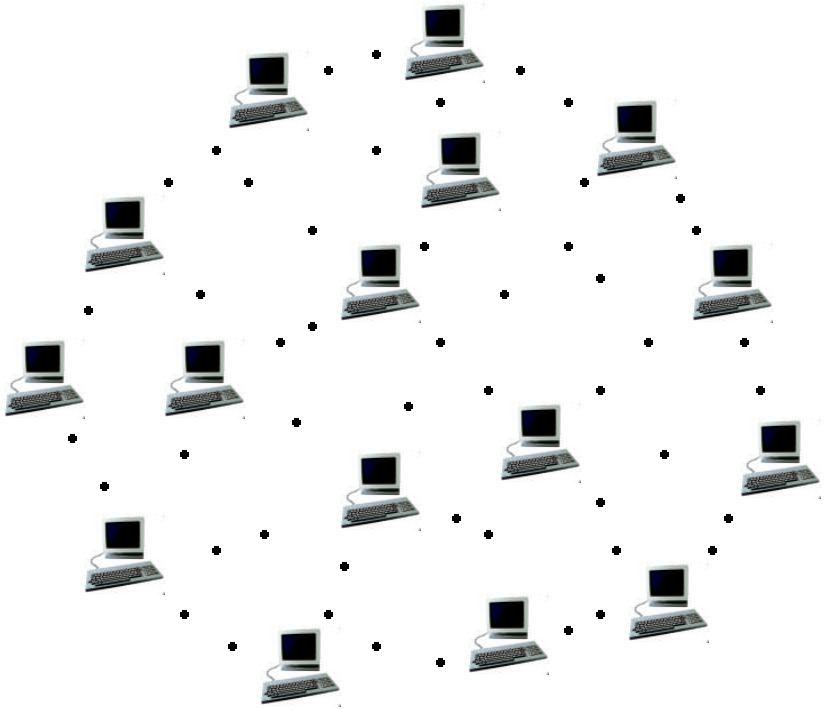
First stage	Several hundreds of workstations
	2 months
Second stage	
	Cray C916
	~2 weeks

Estimated Difficulty of factoring a 1024-bit number

according to RSA Security, Inc.

342 million PCs, 500 MHz
170 GB RAM

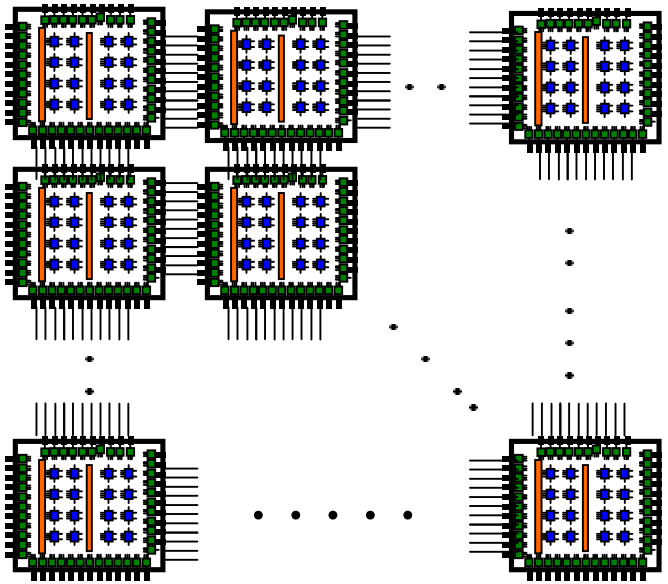
1 year



Our Task

Determine how hard is to break RSA for factoring large key sizes using reconfigurable hardware

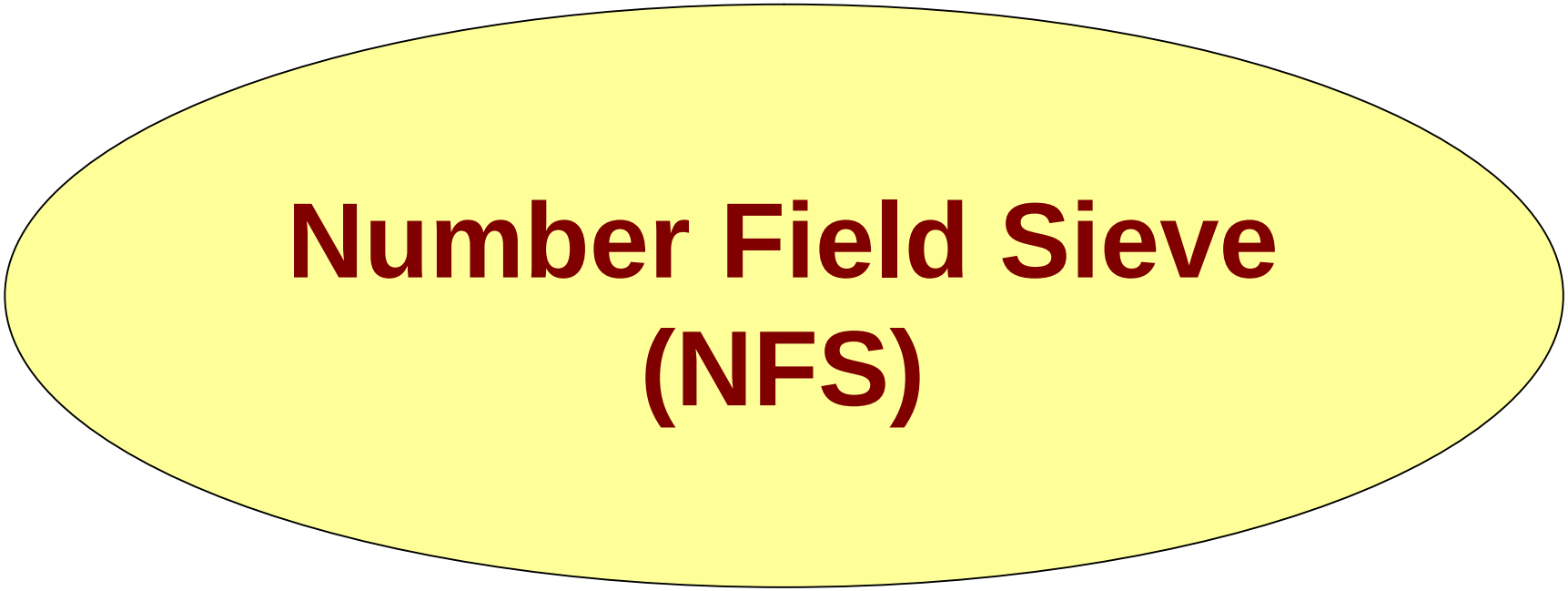
This paper



Generic Array of FPGAs

Future publications

Existing Reconfigurable SuperComputers

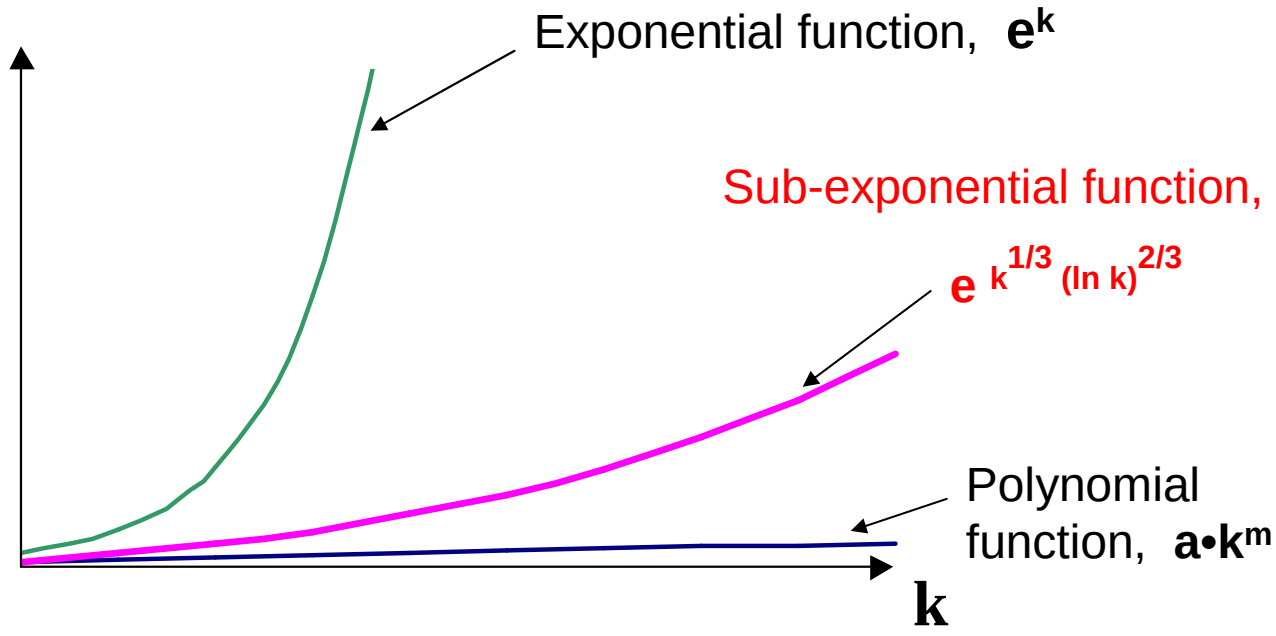

Number Field Sieve
(NFS)

Best Algorithm to Factor Large Numbers

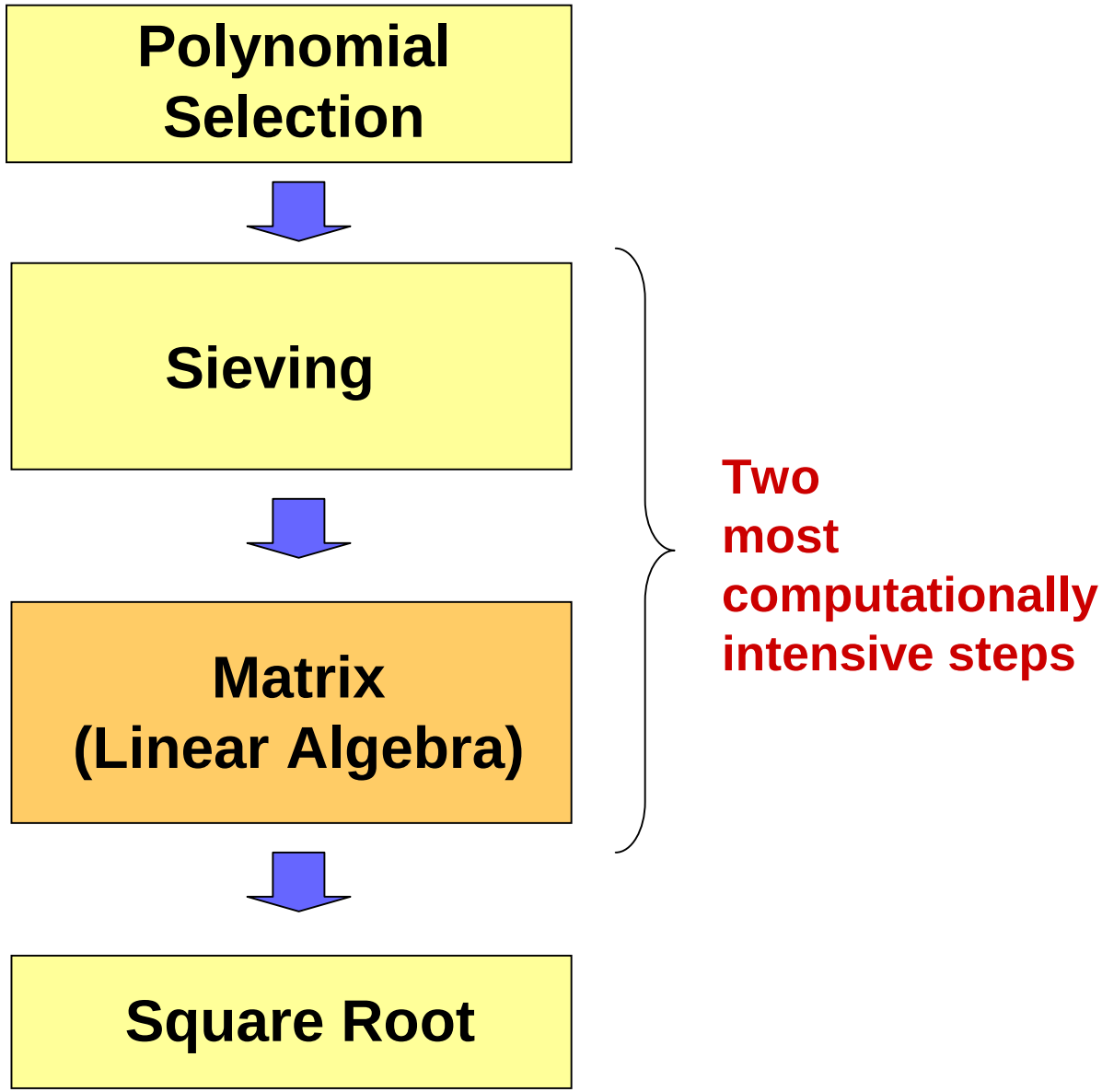
NUMBER FIELD SIEVE

Complexity: Sub-exponential time and memory

N = Number to factor,
k = Number of bits of N



Number Field Sieve (NFS) Steps



Hardware Architecture of NFS proposed to date

Daniel Bernstein

University of Illinois, Chicago

Mesh Approach

Matrix and Sieving

Mesh Sorting

Matrix

www, Fall 2001

Adi Shamir, Eran Tromer

Weizmann Institute of Science, Israel

Mesh Routing

Matrix

Asiacrypt 2002

TWIRL

Sieving

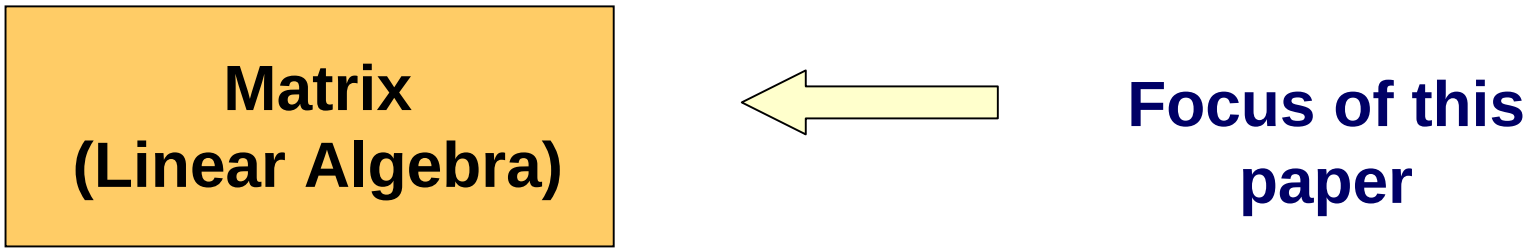
Crypto 2003,
Asiacrypt 2003

Mesh method improves *asymptotic complexity* for NFS performance

Just analytical estimations, no real implementation, no concrete numbers

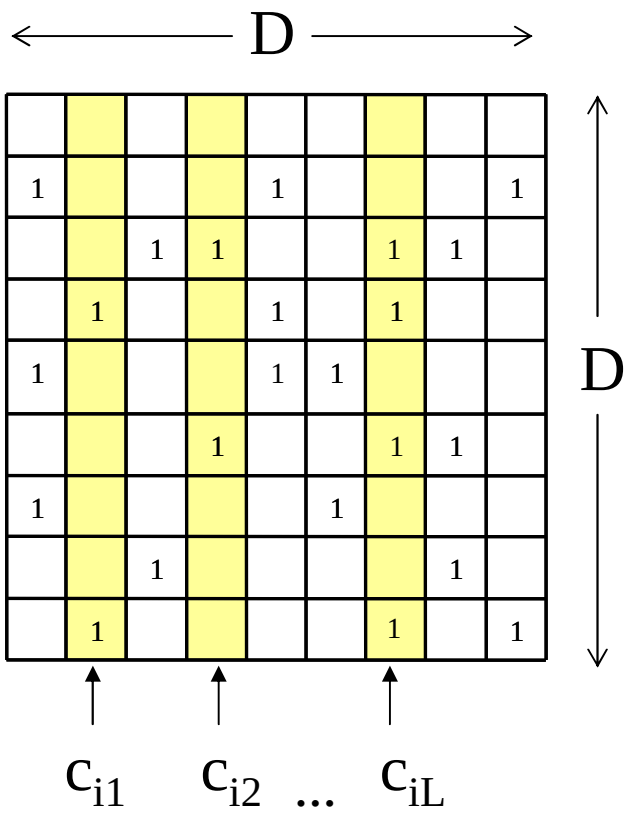
Our Objective

**Design, describe in RTL VHDL,
and estimate the real difficulty
of the Matrix Step of NFS
when implemented using current generation of
FPGA devices**



Function of the Matrix Step

Find linear dependency in the large sparse matrix obtained after sieving step



$$c_{i1} \oplus c_{i2} \oplus \ldots \oplus c_{iL} = 0$$

$$D \approx \begin{cases} 10^6 \text{ for a 512-bit N} \\ 10^7 \text{ for a 1024-bit N} \end{cases}$$

D = number of the matrix columns and rows

Block Wiedemann Algorithm for the Matrix Step of NFS

1) Uses **multiple matrix-by-vector multiplications** of the sparse matrix A with K random vectors v_i

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

where $k = 2D/K$

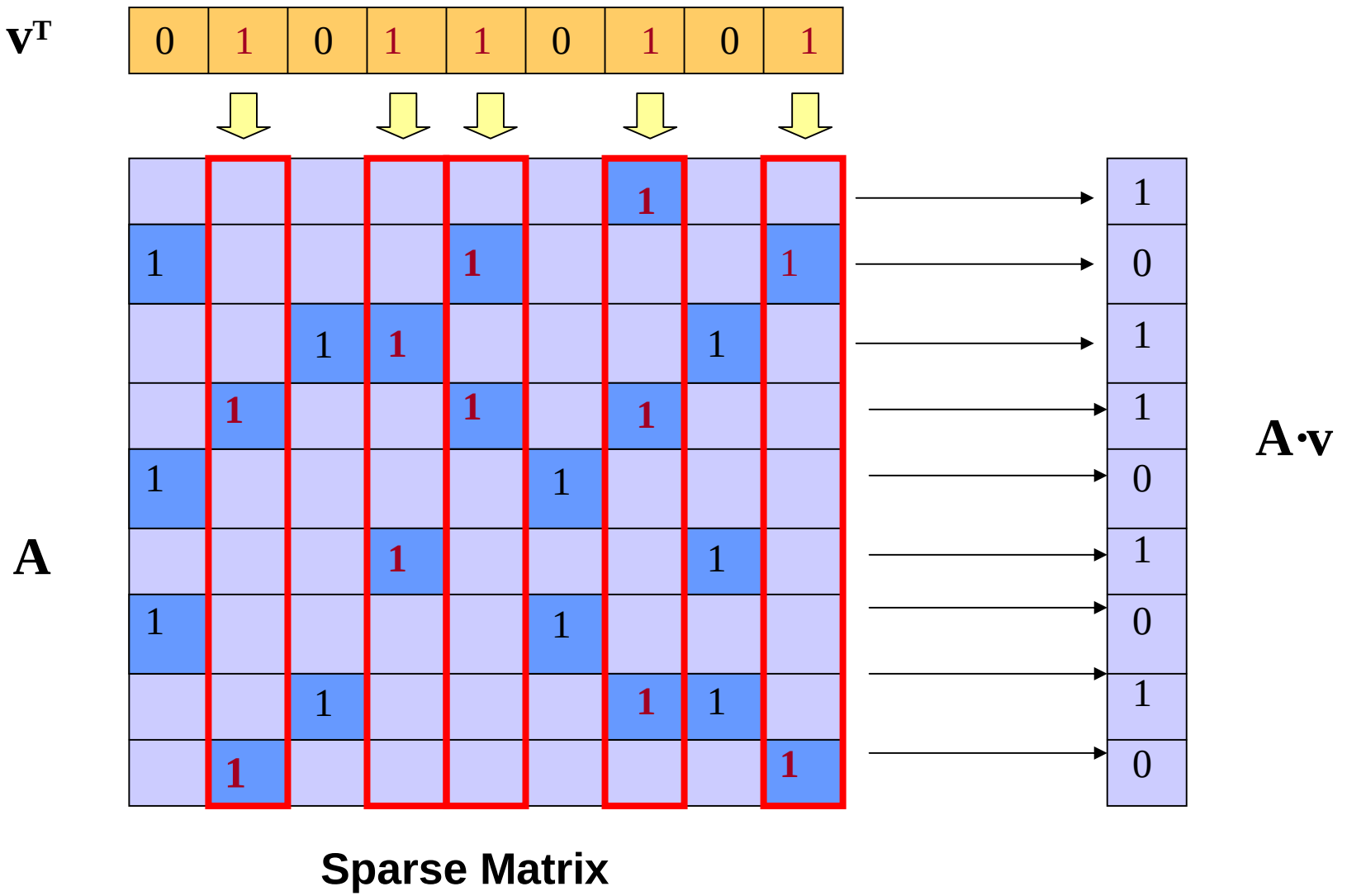
2) Post computations leading to the determination of the linear dependence of the matrix columns

Most time consuming operation:

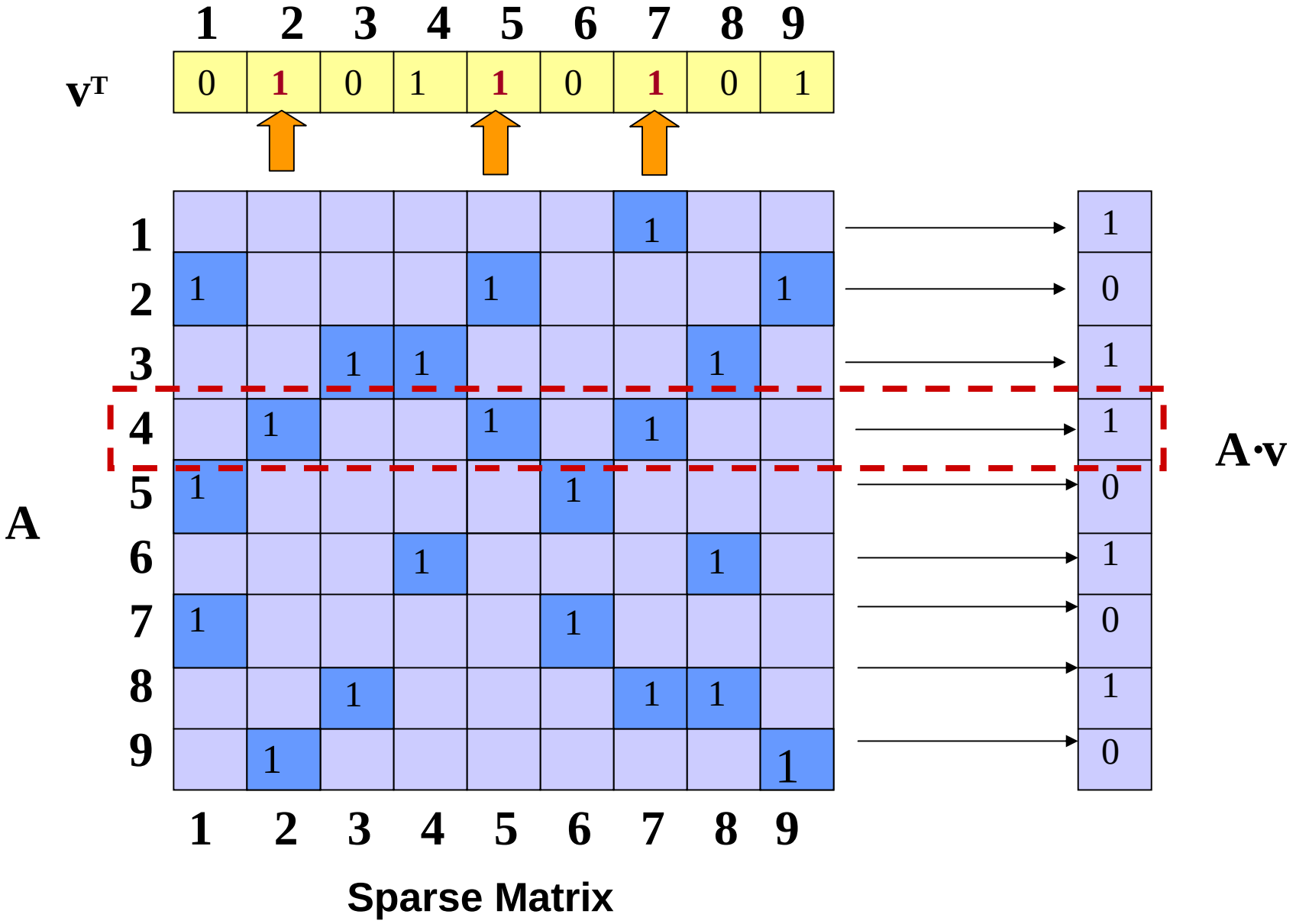
$$A_{[DxD]} \cdot v_{[Dx1]}$$

Mesh Routing Architecture for the Matrix Step

Matrix-by-Vector Multiplication

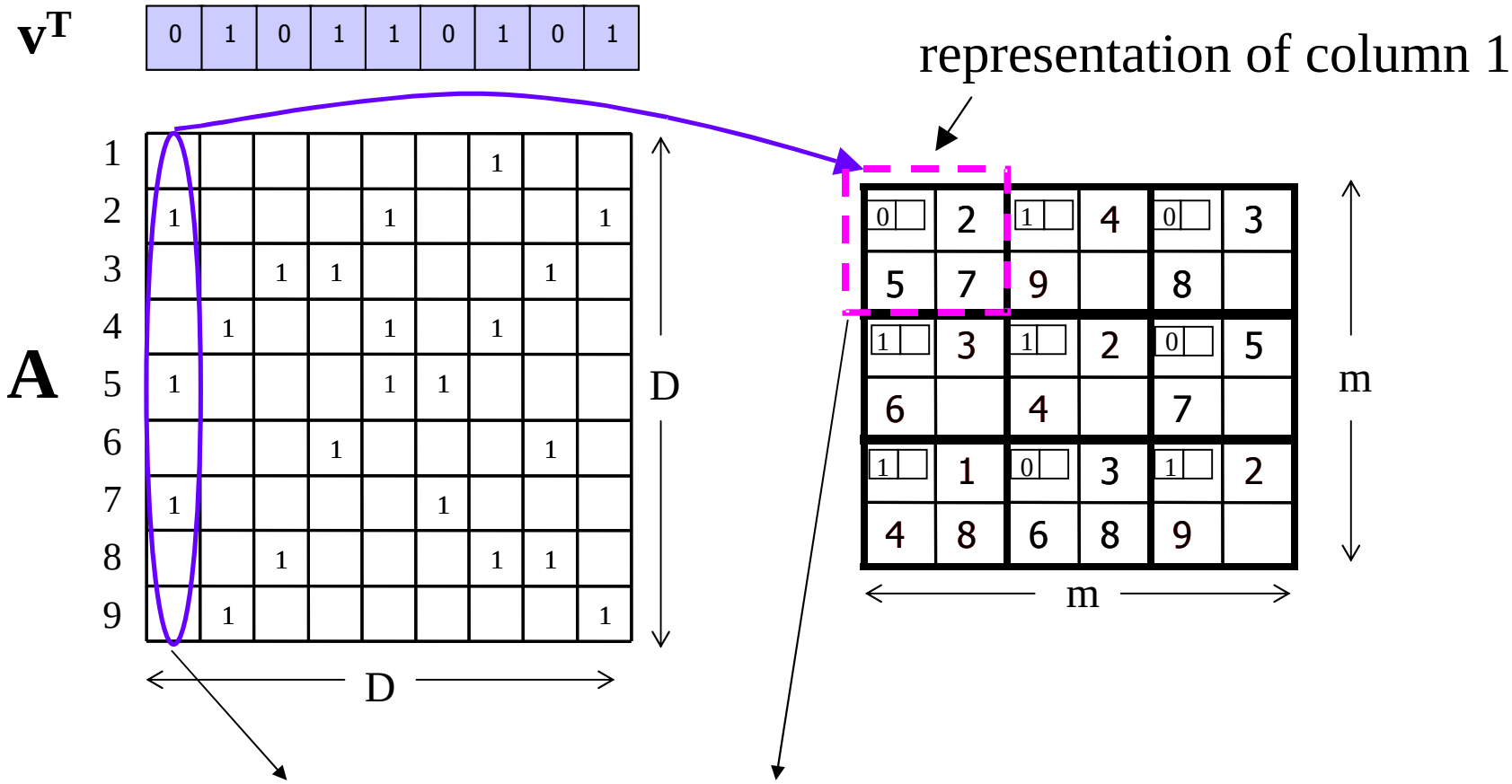


Matrix-by-Vector Multiplication for Sparse Matrices



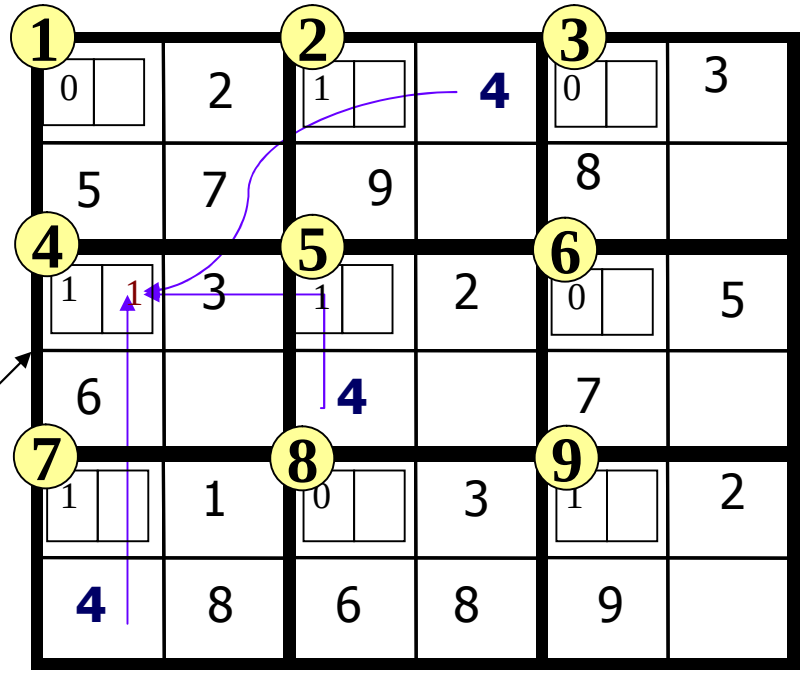
Mesh Routing

m x m mesh where m = $\sqrt{D}$



d = maximum number of non-zero entries per column

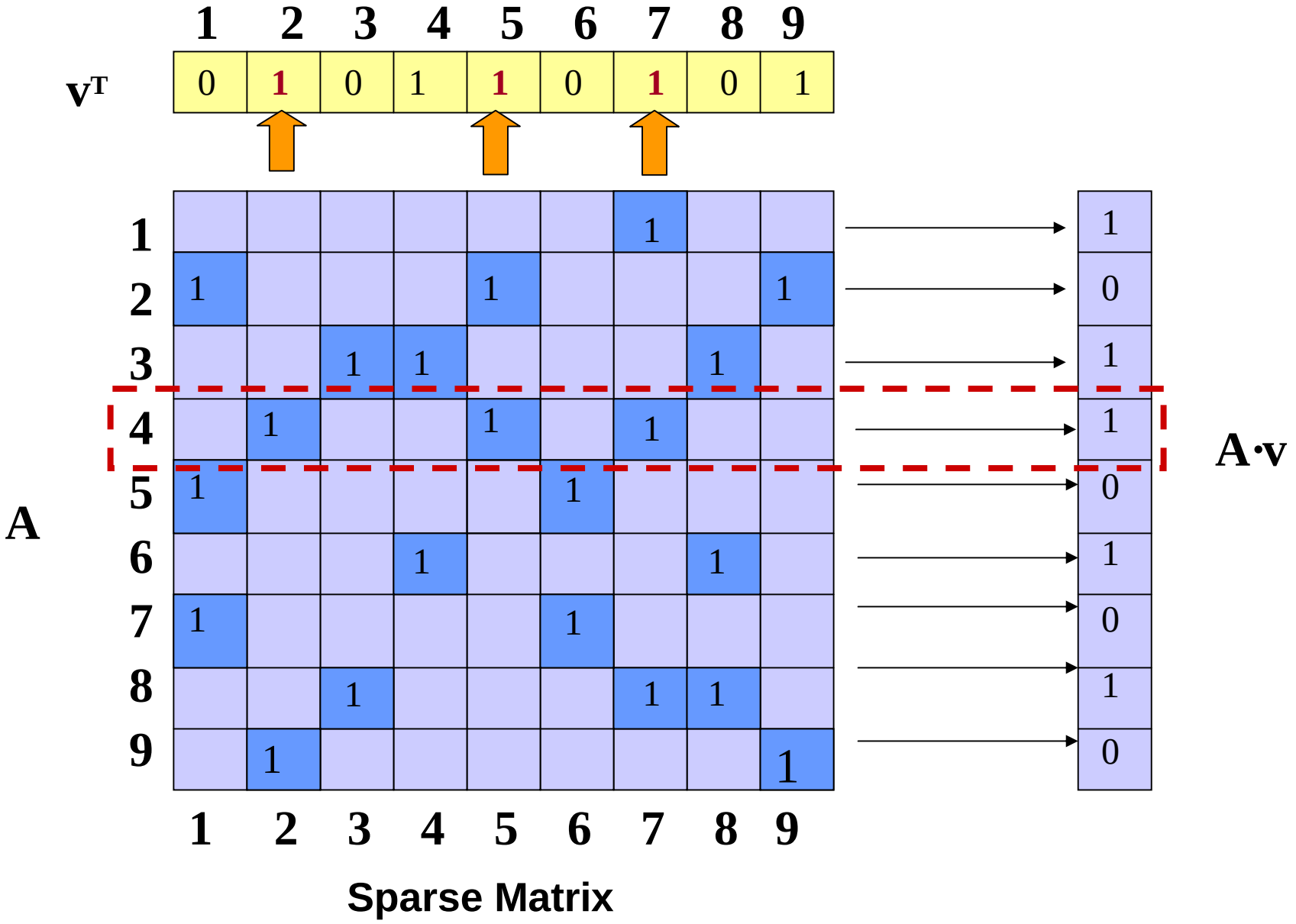
Routing in the Mesh



Fourth cell

Each time a packet arrives at the target cell, the packet's vector's bit is xored with the partial result bit on the target cell

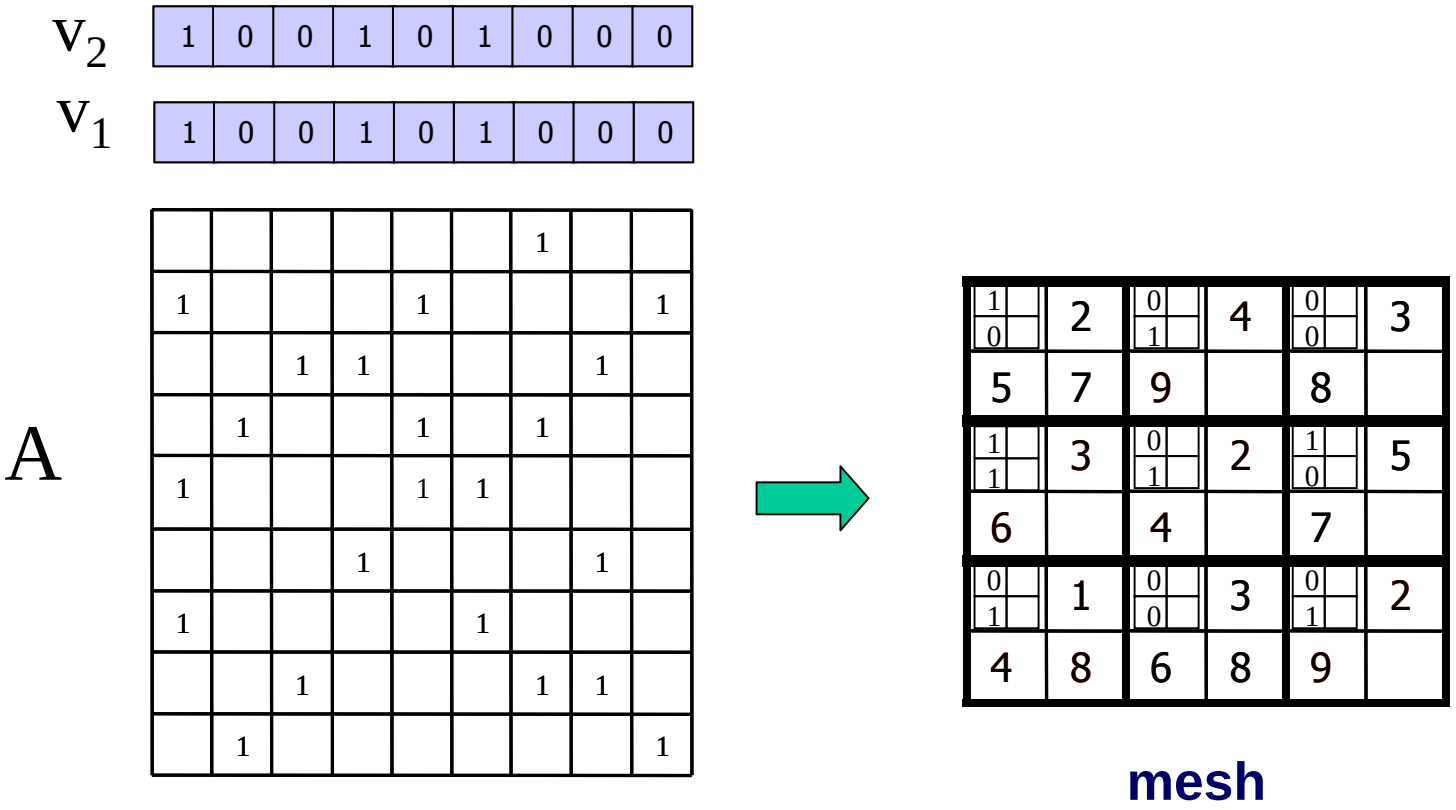
Matrix-by-Vector Multiplication for Sparse Matrices



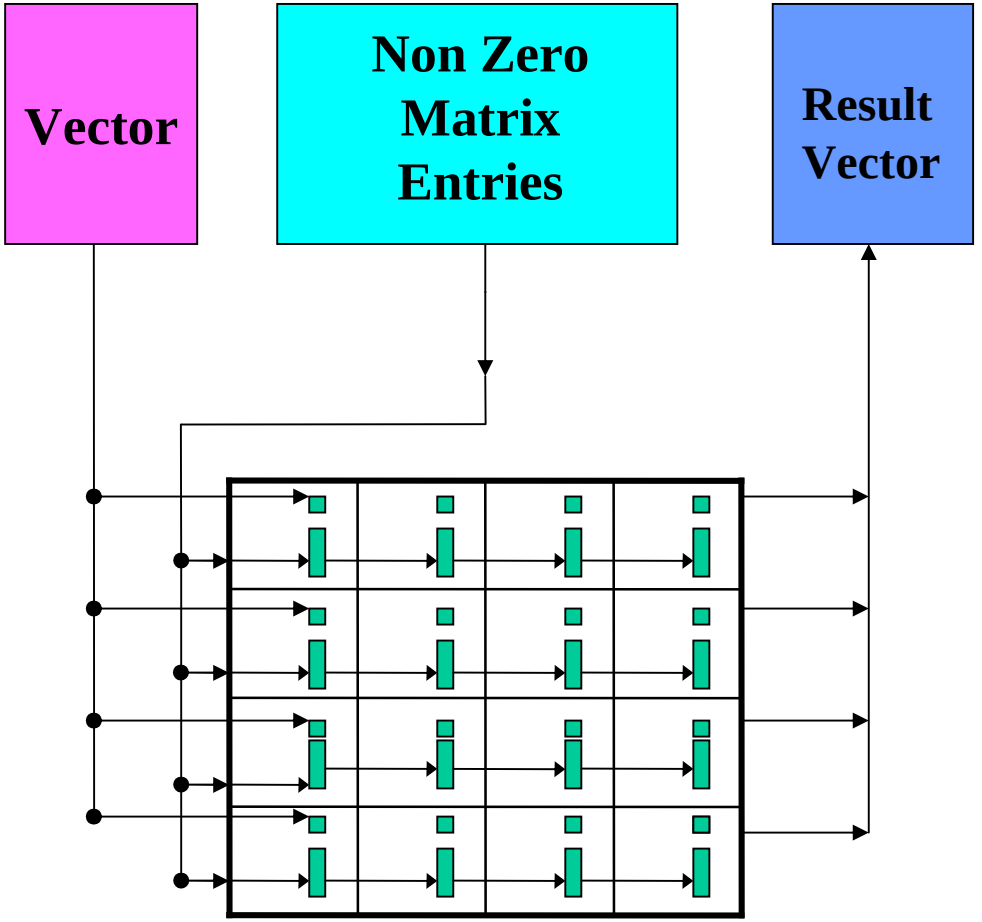
Mesh Routing with K parallel matrix by vector multiplications

Example for K=2

$A \cdot v_1$ and $A \cdot v_2$ computed in parallel

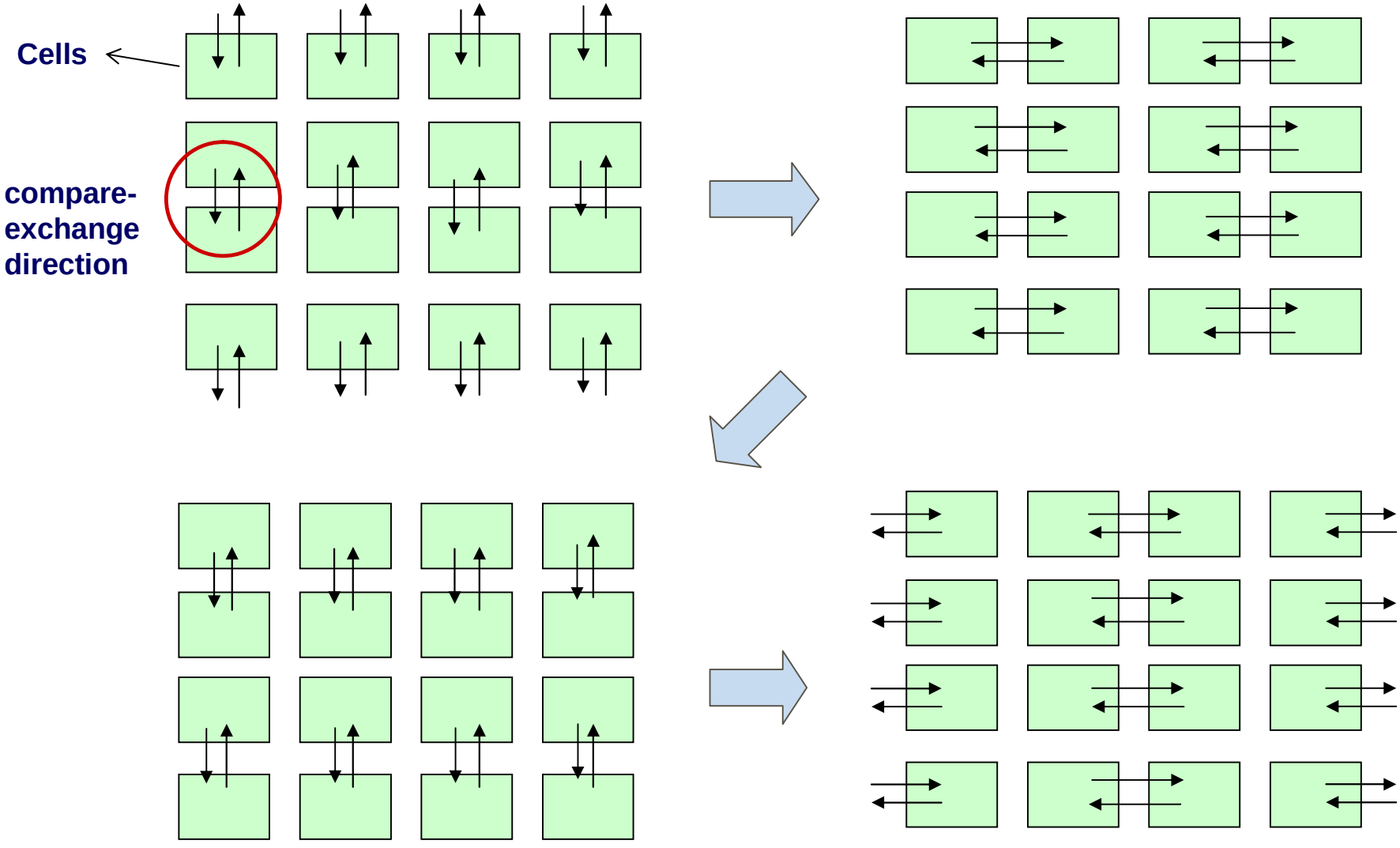


Parallel Loading & Unloading of Data



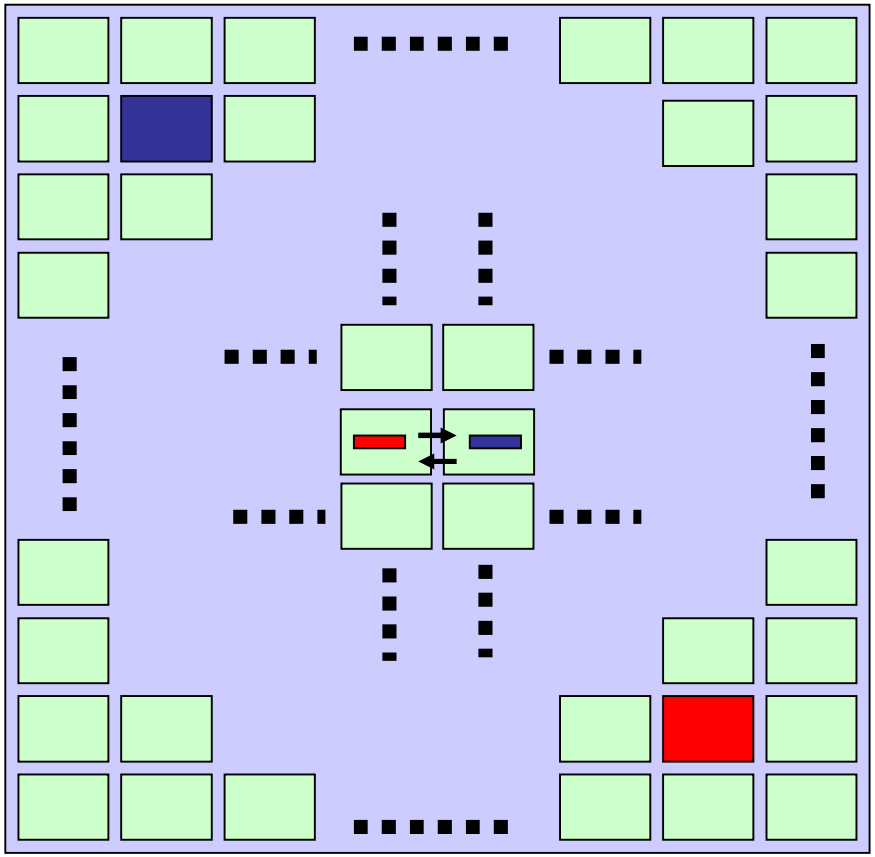
Clockwise Transposition Routing

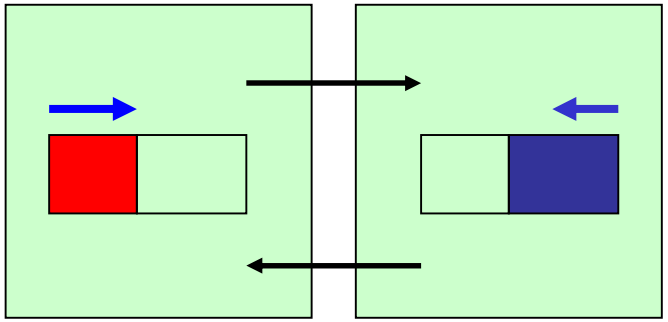
Four iterations repeated



Compare-Exchange Cases

Between Columns

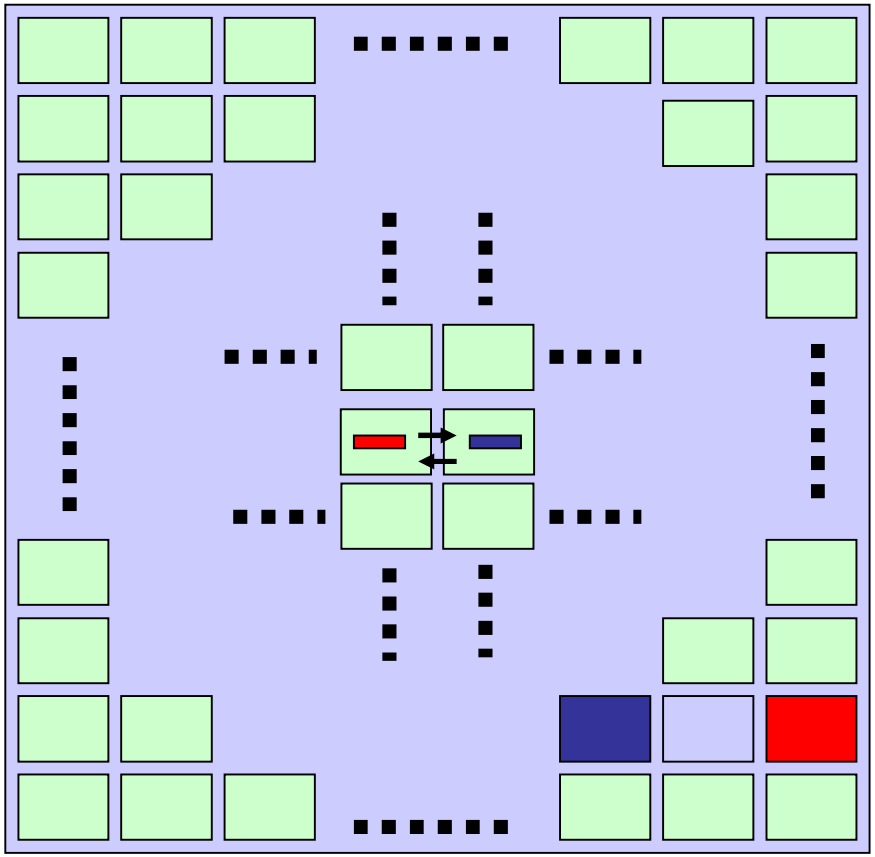


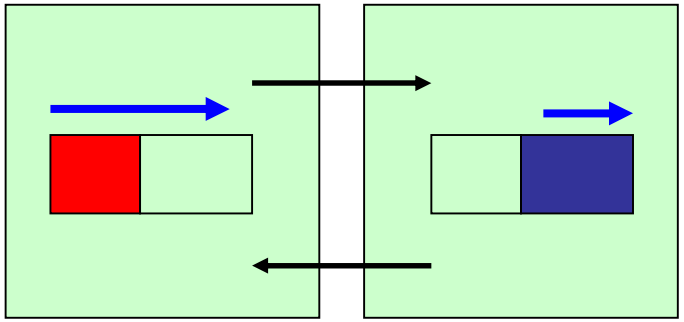


- **Direction of travel = direction of exchange**
- **Result of comparison = Exchange the packets.**

Compare-Exchange Cases

Between Columns





- **Direction of travel ≠ direction of exchange**
- **Result of comparison = Exchange the packets.**
- **Rule for Exchange: Exchange iff the distance to target of the packet which is farthest from its destination gets reduced.**

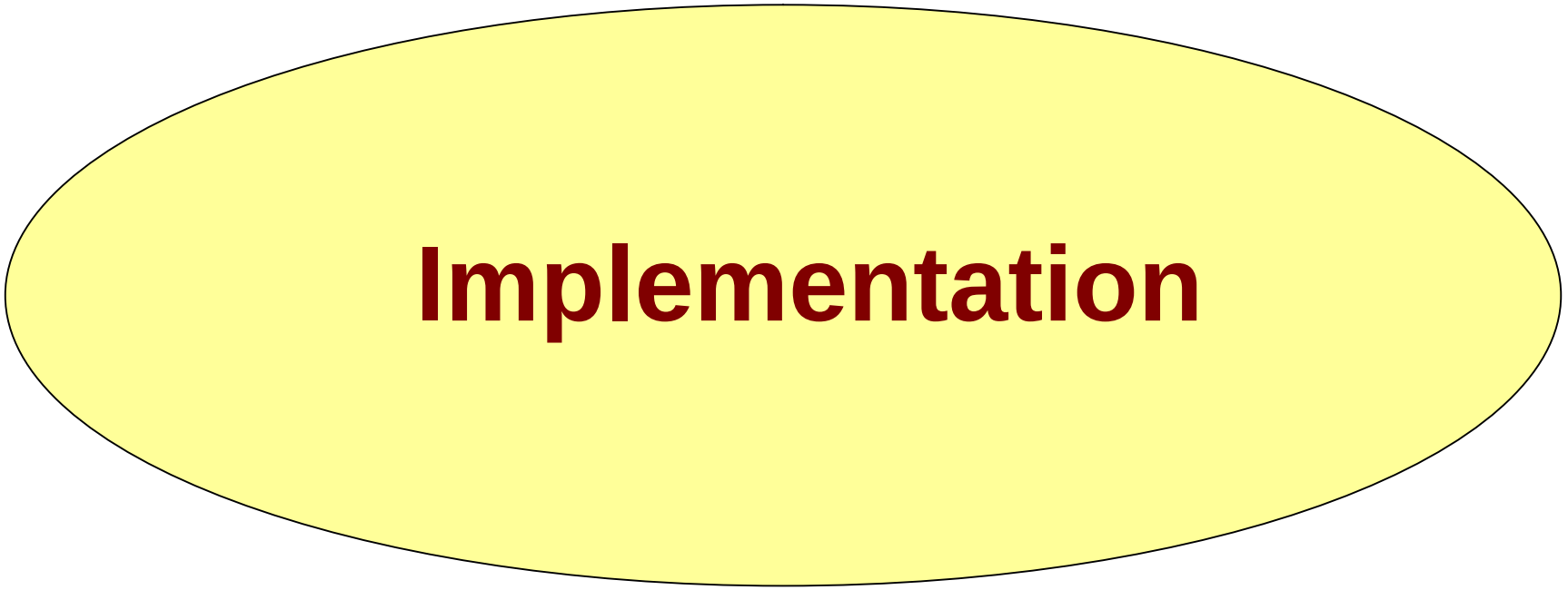
Total time of routing

- **Total routing takes maximum 4·m·d compare-exchange operations,**

  where

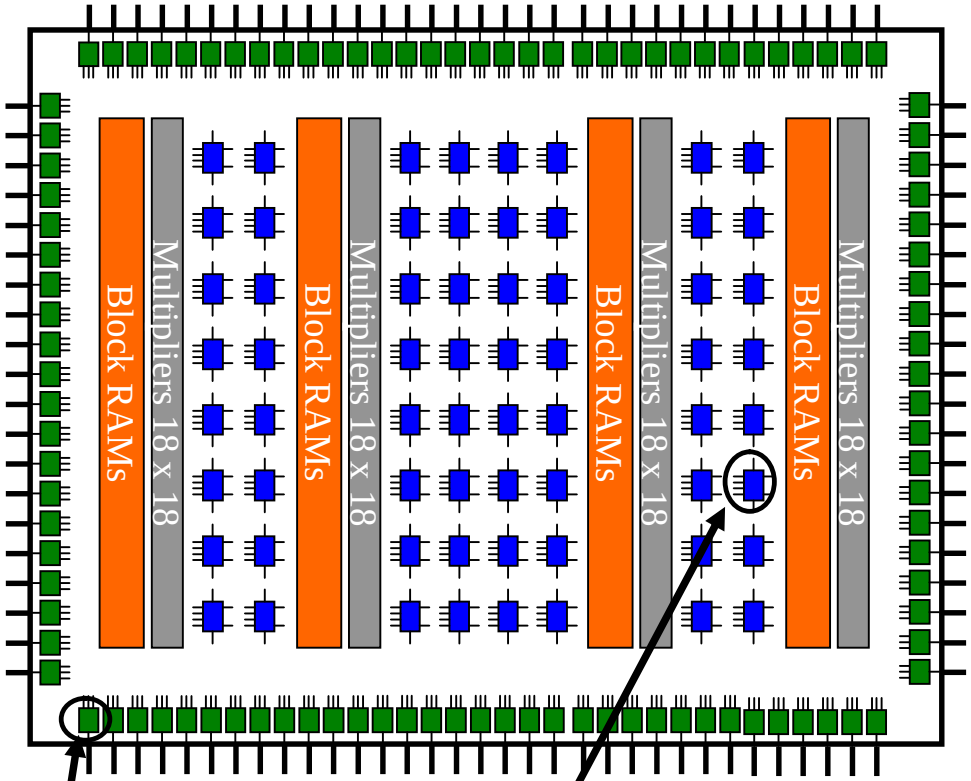
  d – matrix density = maximum number of non-zero entries per column

  m – mesh size = $\sqrt{D}$, where D is the matrix size


Implementation

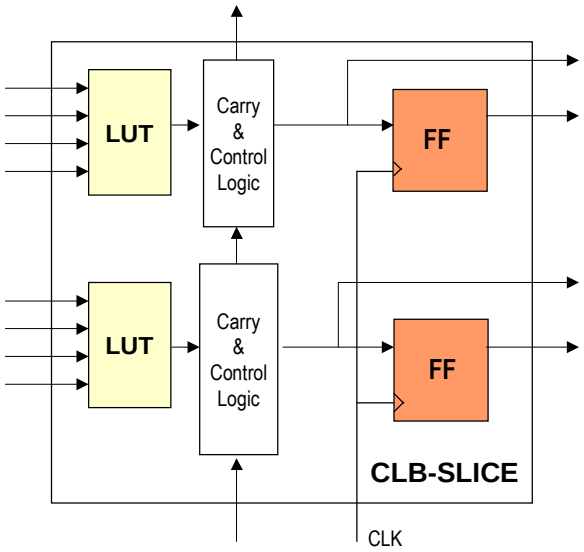
Target FPGA Devices

Xilinx Virtex II



I/O Block

CLB slice



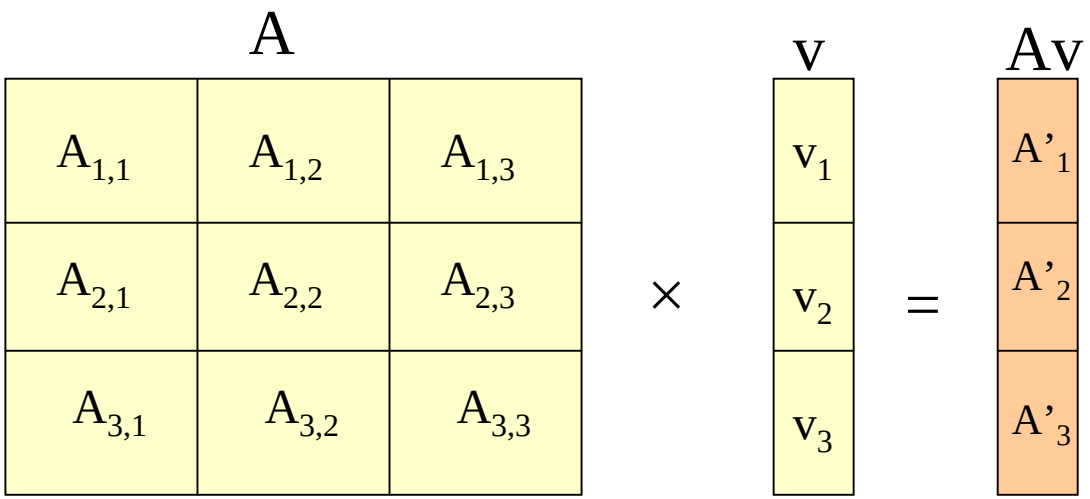
XC2V6000

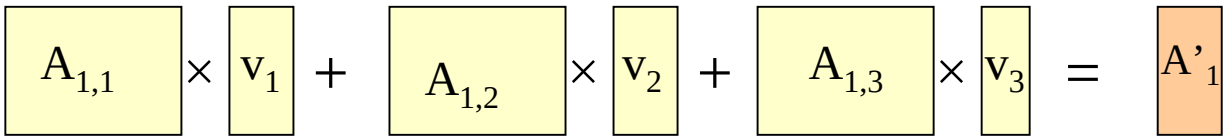
- 33,792 CLB slices
- 67,584 LUT (LookUpTable)
- 67,584 FF (Flip-Flop)

XC2V8000

- 46,592 CLB slices
- 93,184 LUT (LookUpTable)
- 93,184 FF (Flip-Flop)

Distributed Computation
(Geiselmann, Steinwandt, CHES 2003)



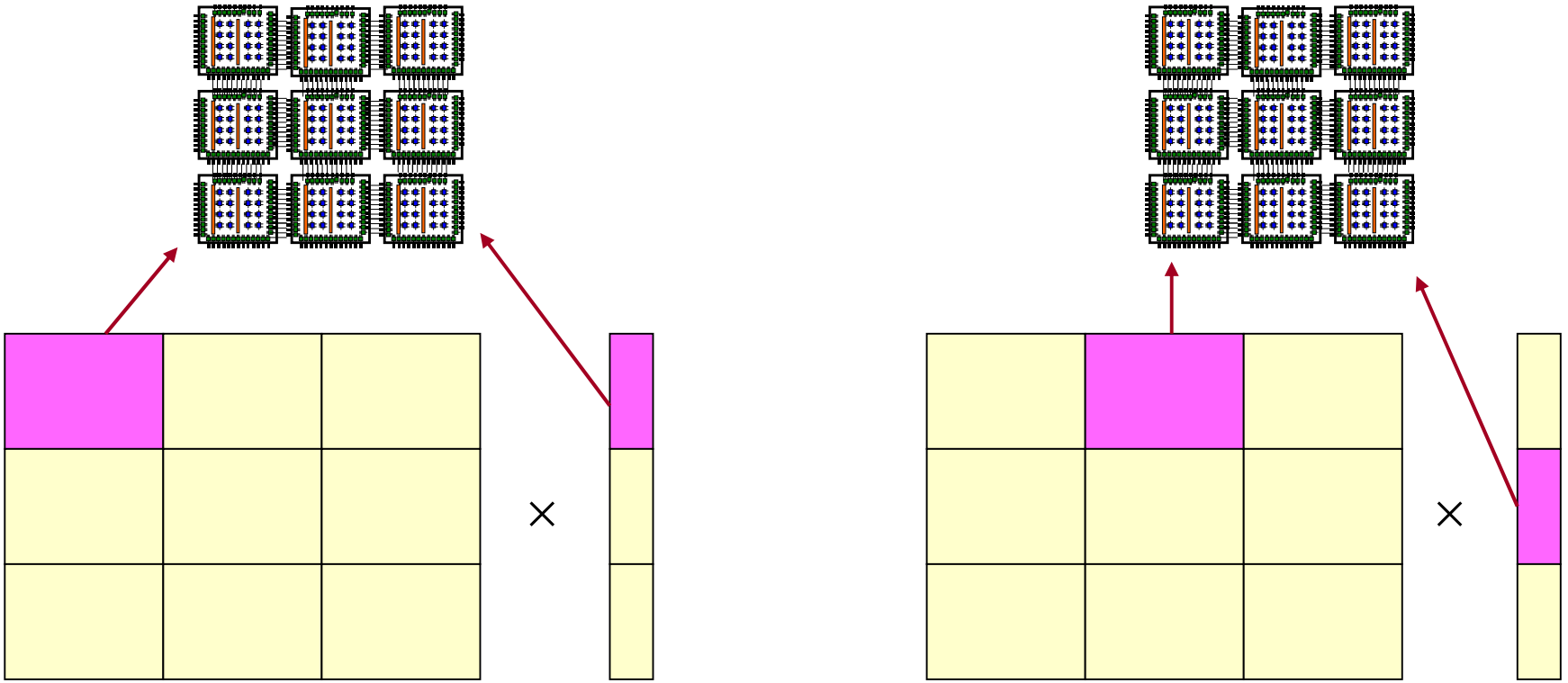


$$A \cdot v = \begin{pmatrix} \sum_{j=1}^{s} A_{1,j} \cdot v_j \\ \vdots \\ \sum_{j=1}^{s} A_{s,j} \cdot v_j \end{pmatrix}$$

Using smaller FPGA arrays to perform the entire computation

1) FPGA array performs single sub-matrix by sub-vector multiplication

2) Reuse FPGA array for next sub-computation



Results and Analysis

Time of a single matrix-by-vector multiplication

Matrix Size	K	CLB slices	LUTs	FFs	Clock Period (ns)	Time for K mult (ns)	Time per 1 mult (ns)
144x144 (Mesh 12x12)	1	8123 (17%)	15,495 (16%)	5,318 (5%)	14	672	672
144x144 (Mesh 12x12)	32	23,949 (51%)	46,944 (50%)	23,419 (25%)	16.6	797	25
144x144 (Mesh 12x12)	70	43,065 (92%)	84,836 (91%)	45,378 (48%)	17.8	854	12

K = number of concurrent matrix-vector multiplications

Time for K mult = d * 4 * m * clock period

Speedup vs. Software Implementation

Reference Optimized SW Implementation:

PC, Pentium IV, 2.8 GHz, 1 GB RAM

Matrix Size	K	One Multiplication Time in SW (ns)	One Multiplication Time in HW (ns)	Speedup
144x144 (Mesh 12x12)	70	3440	12	282

Results for a 512-bit number N

D = number of columns in matrix A
p = number of columns handled in one cell=16
n = number of times to repeat sub-multiplications = $D^2/(m^2\ p)^2$
T_K = routing time for K multiplications in the mesh = p*d*4*m*Clock period
T_{Load} = time for loading and unloading for K multiplications
T_{Total} = total time for a Matrix step = 3*D/K* n *(T_K + T_{Load})

Virtex II chips	D	m	n	T_K (ns)	T_{Load} (ns)	T_{Total} (days)	Speedup vs. 1 chip
1	$6.7x10^6$	12	$8.4 x 10^6$	13593	1568	593	1
10^2	$6.7x10^6$	120	846	$8 x10^5$	$2.1 x10^5$	4	148
16^2	$6.7x10^6$	192	129	$1.3 x10^6$	$3.8 x10^5$	0.96	617
32^2	$6.7x10^6$	384	8	$2.6 x10^6$	$9 x10^5$	0.132	4492

Comparison vs. Cray Implementation

512-bit number, Improved Mesh Routing Design

Cray C916	Array of 1024 Virtex II 8000 FPGAs
9.3 days = 224 hours	0.13 days = 3.2 hours

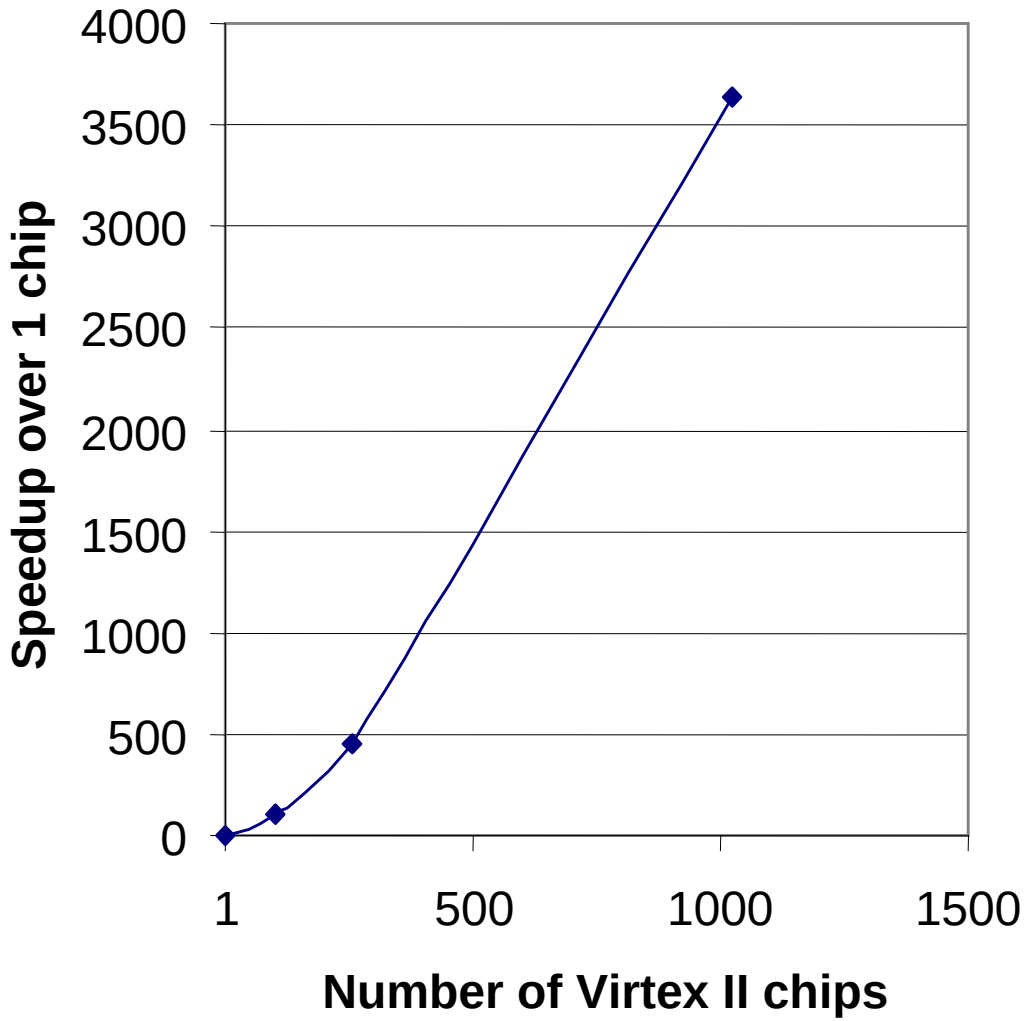
Speed-up = 70

Results for a 1024-bit number N

D = number of columns in matrix A
p = number of columns handled in one cell=16
n = number of times to repeat sub-multiplications = $D^2/(m^2\ p)^2$
T_K = routing time for K multiplications in the mesh = p*d*4*m*Clock period
T_{Load} = time for loading and unloading for K multiplications
T_{Total} = total time for a Matrix step = 3*D/K* n *(T_K + T_{Load})

Virtex II chips	D	m	n	T_K (ns)	T_{Load} (ns)	T_{Total} (days)	Speed up vs. 1 chip
1	4 x 10^7	12	3.0 x 10^8	13593	1568	126851	1
10^2	4 x 10^7	120	3.0 x 10^4	8 x10^5	2.1 x10^5	864	147
16^2	4 x 10^7	192	4599	1.3 x10^6	3.8 x10^5	210	604
32^2	4 x 10^7	384	287	2.6 x10^6	9 x10^5	27	4698

Speedup vs. number of FPGA chips



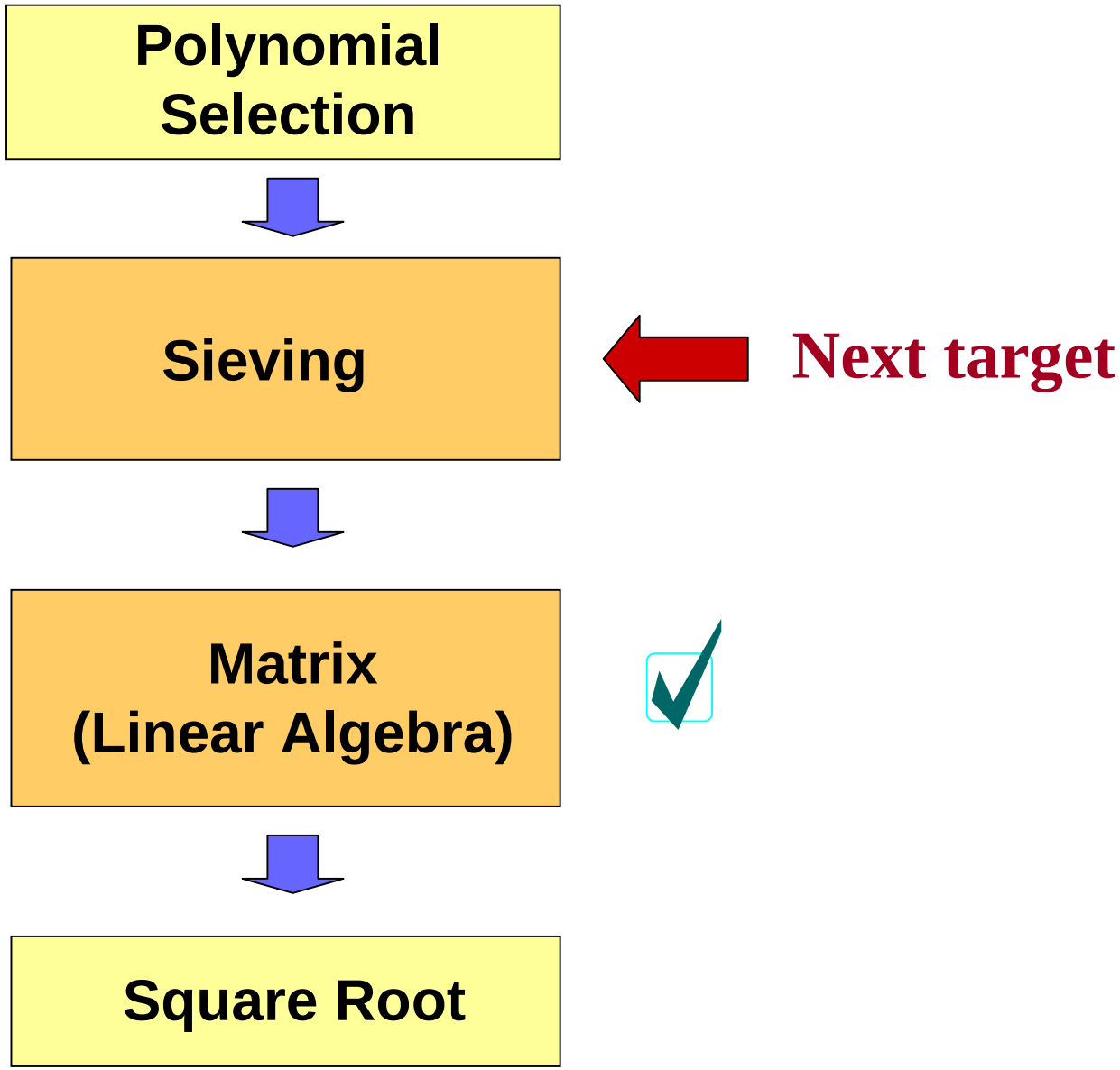
~ $(\#FPGA)^{3/2}$

Conclusions

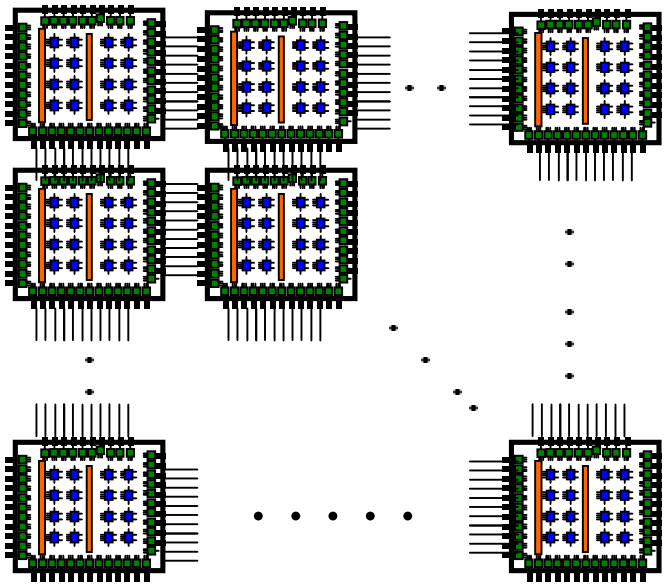
Summary & Conclusions

- First practical hardware implementation of Mesh Routing for the Number Field Sieve implemented and verified using timing simulation
- Practical numbers, based on post-placing & routing static timing analysis, obtained for an array of Xilinx Virtex II 8000 FPGAs
- Matrix step for a 1024-bit number can be performed using 1024 Xilinx Virtex II 8000 chips in 27 days
- A two-dimensional array of Virtex II chips can perform computations faster than a single FPGA by a factor proportional to $(\text{number of FPGAs})^{3/2}$

Future research (1)



Future research (2)



Next target