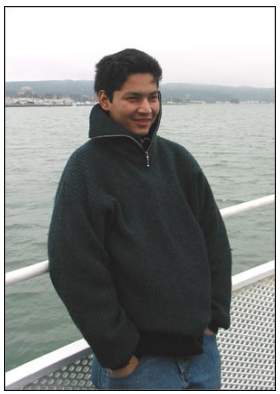
Reconfigurable Hardware Implementation of Mesh Routing in the Number Field Sieve Factorization



Sashisu Bajracharya,

Deapesh Misra,

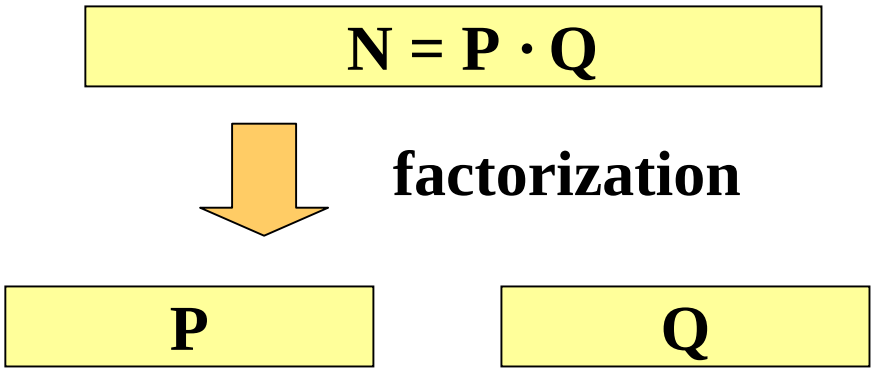
Kris Gaj

George Mason University

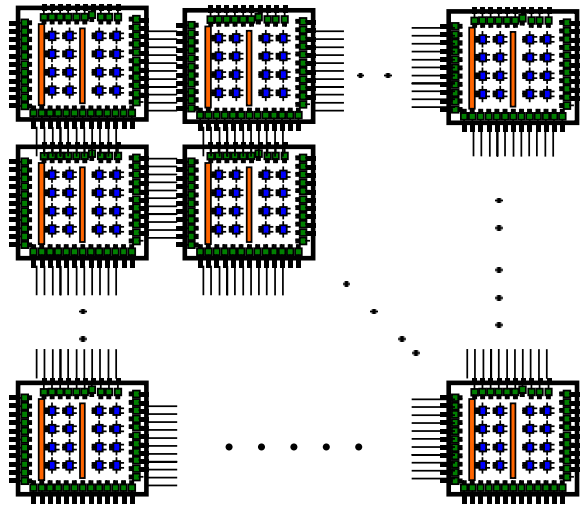
Tarek El-Ghazawi

The George Washington University

Objective & Outline



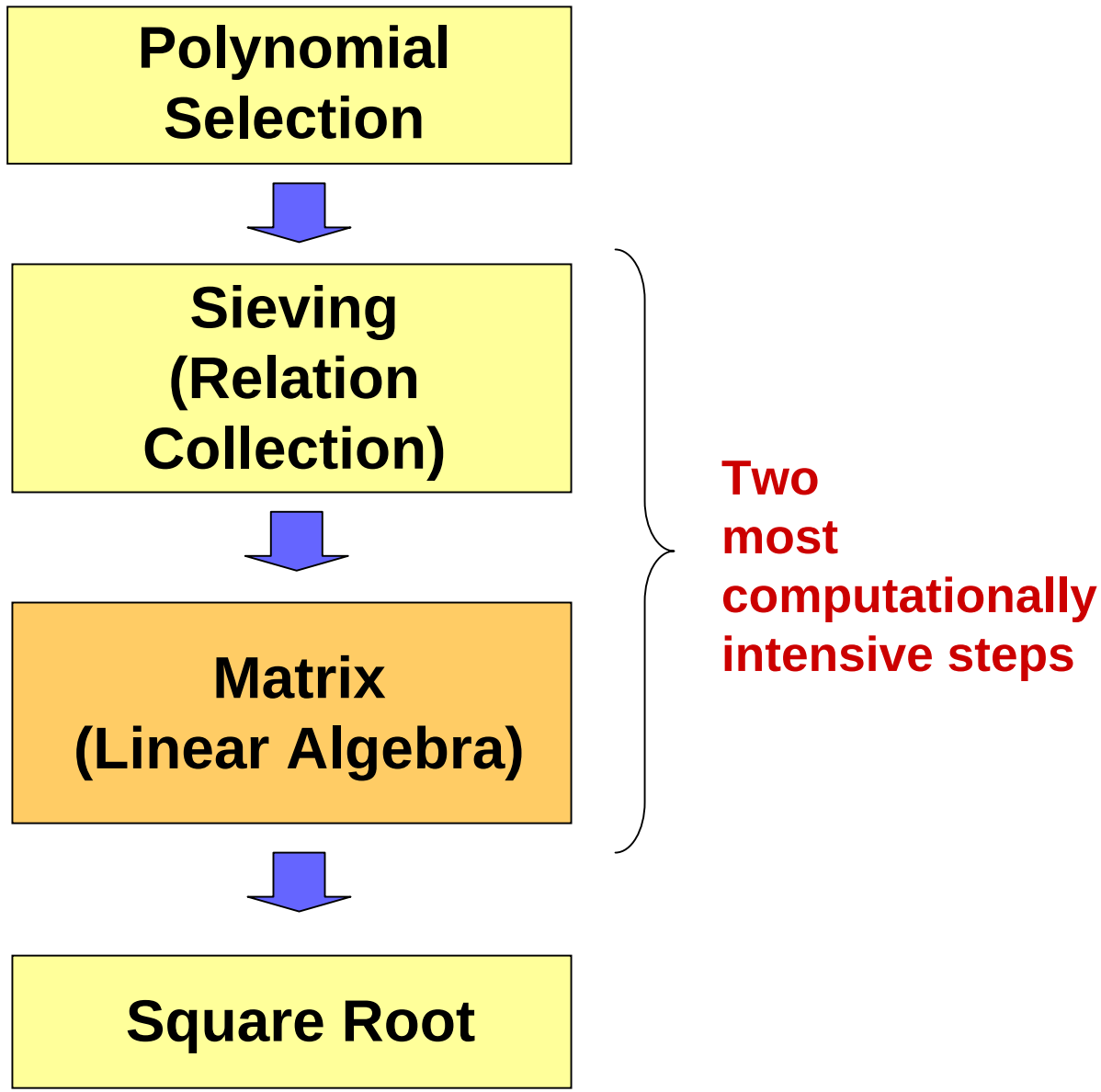
P, Q – large integers



FPGA Array

1. Number Field Sieve (NFS) Factorization
2. Mesh Routing Architecture for the Matrix Step of NFS
3. Implementation
4. Results
5. Conclusions
6. Reconfigurable computers & future work

Number Field Sieve (NFS) Steps



Previous work

2001

D. J. Bernstein, “Circuits for integer factorization: a proposal”

Mesh approach to the sieving and matrix steps improves *asymptotic complexity* for NFS performance

2002

A. K. Lenstra, A. Shamir, J. Tomlinson, E. Tromer, “Analysis of Bernstein's Factorization Circuit,” Asiacrypt 2002

Detailed design for *mesh routing*

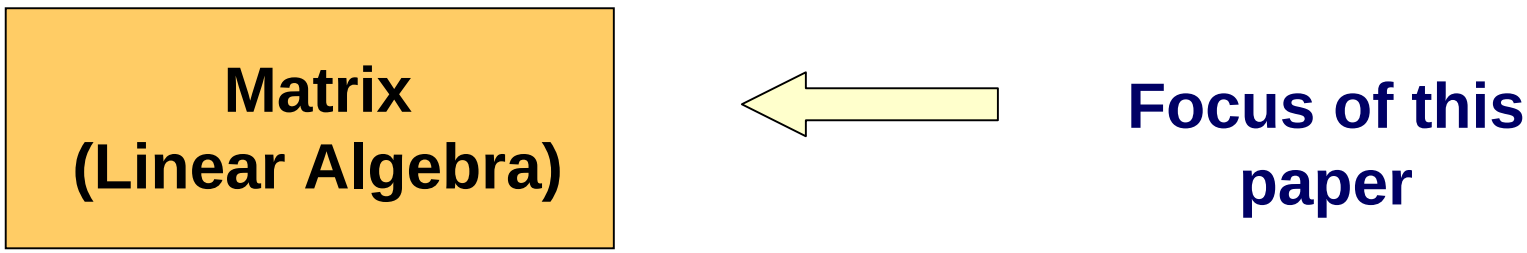
2003

W. Geiselmann, R. Steinwandt, “Hardware to solve sparse systems of linear equations over GF(2)”, CHES 2003

***Distributing computations* among multiple nodes**

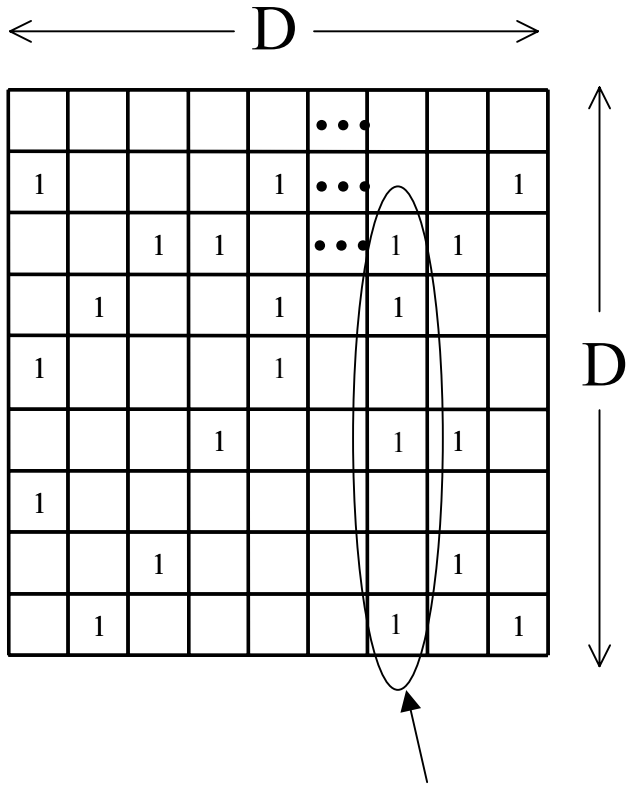
Our Objective

**Design, describe in RTL VHDL,
synthesize & simulate
existing theoretical designs
for the matrix step of NFS
using current generation of FPGA devices.**



Input to the Matrix Step

Matrix A:



D = number of the matrix columns and rows

$D \in [10^7, 10^{11}]$

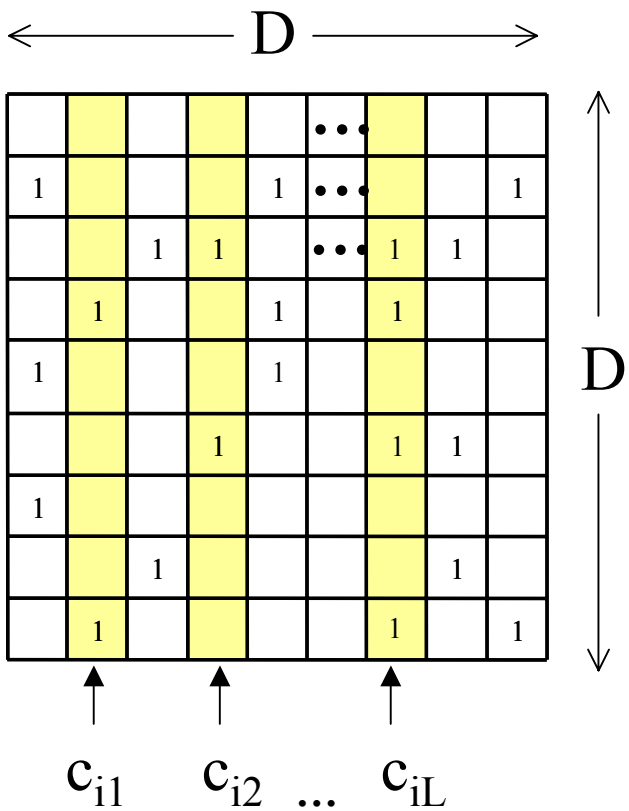
d – column density (weight) = maximum number of ones per column

d << D,

e.g., d=100 for $D=10^{10}$

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 = 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

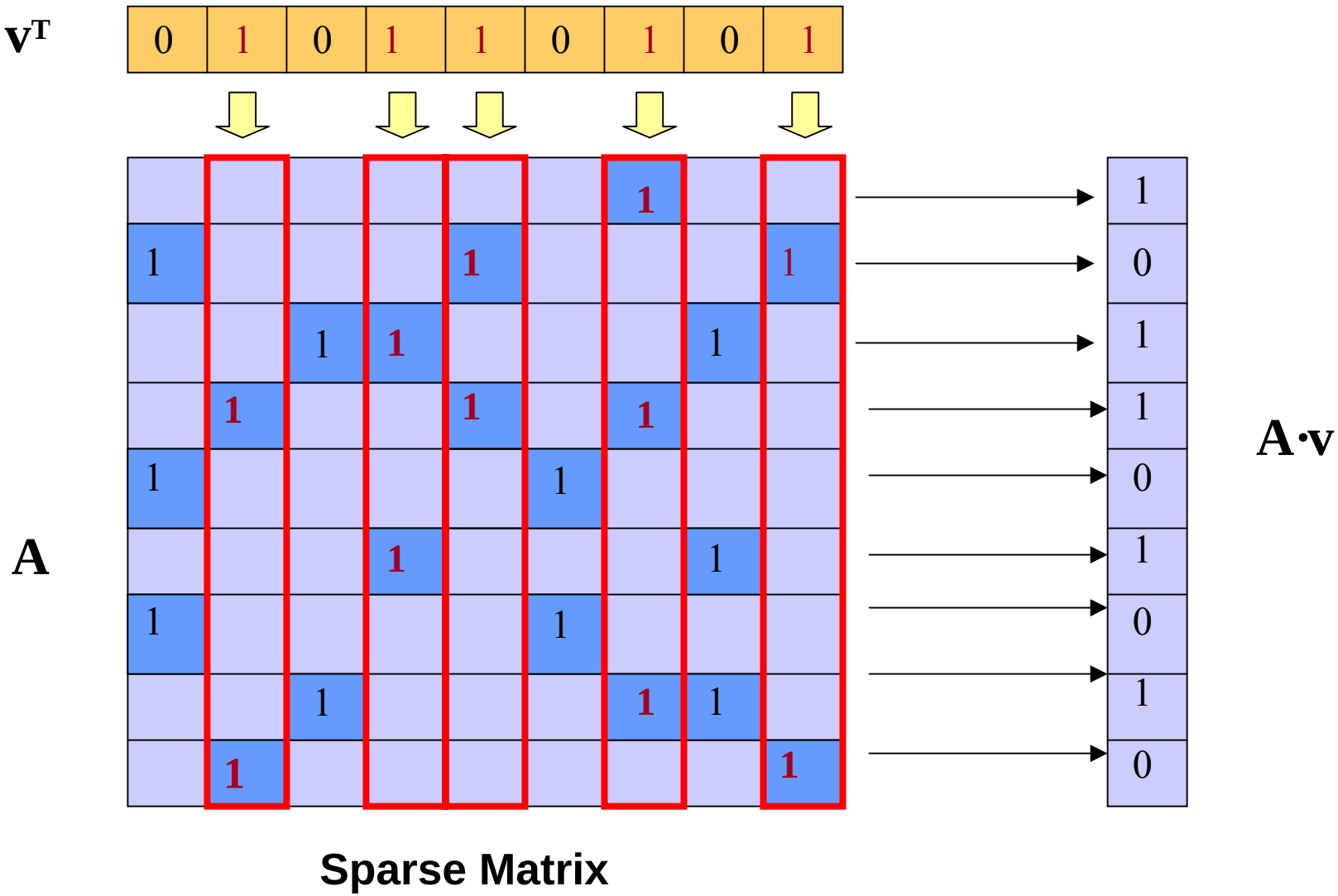
2) Post computations leading to the determination of the linear dependence of the matrix columns

Most time consuming operation:

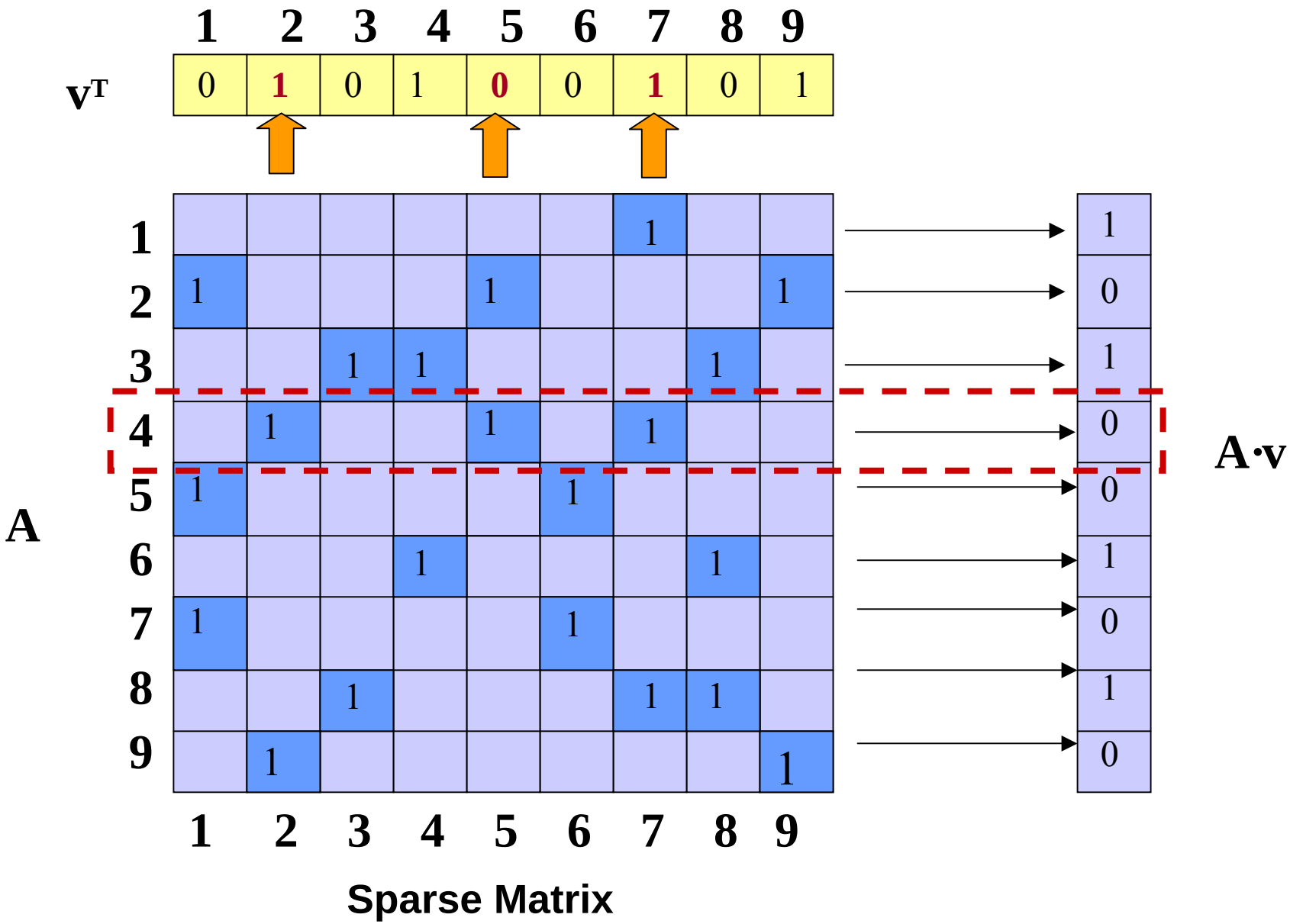
$$A_{[DxD]} \cdot v_{[Dx1]}$$

Mesh Routing Architecture for the Matrix Step

Matrix-by-Vector Multiplication

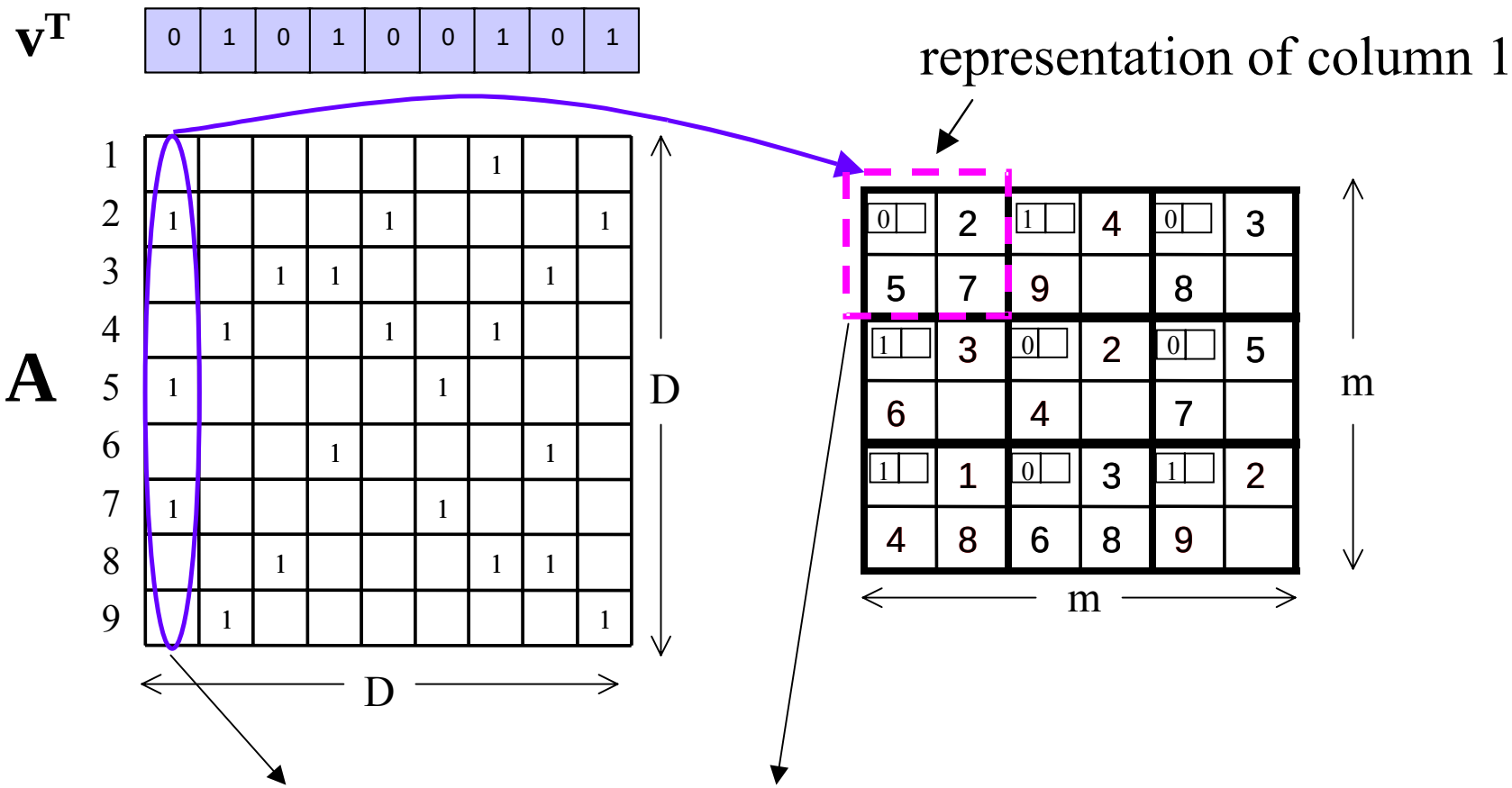


Matrix-by-Vector Multiplication for Sparse Matrices



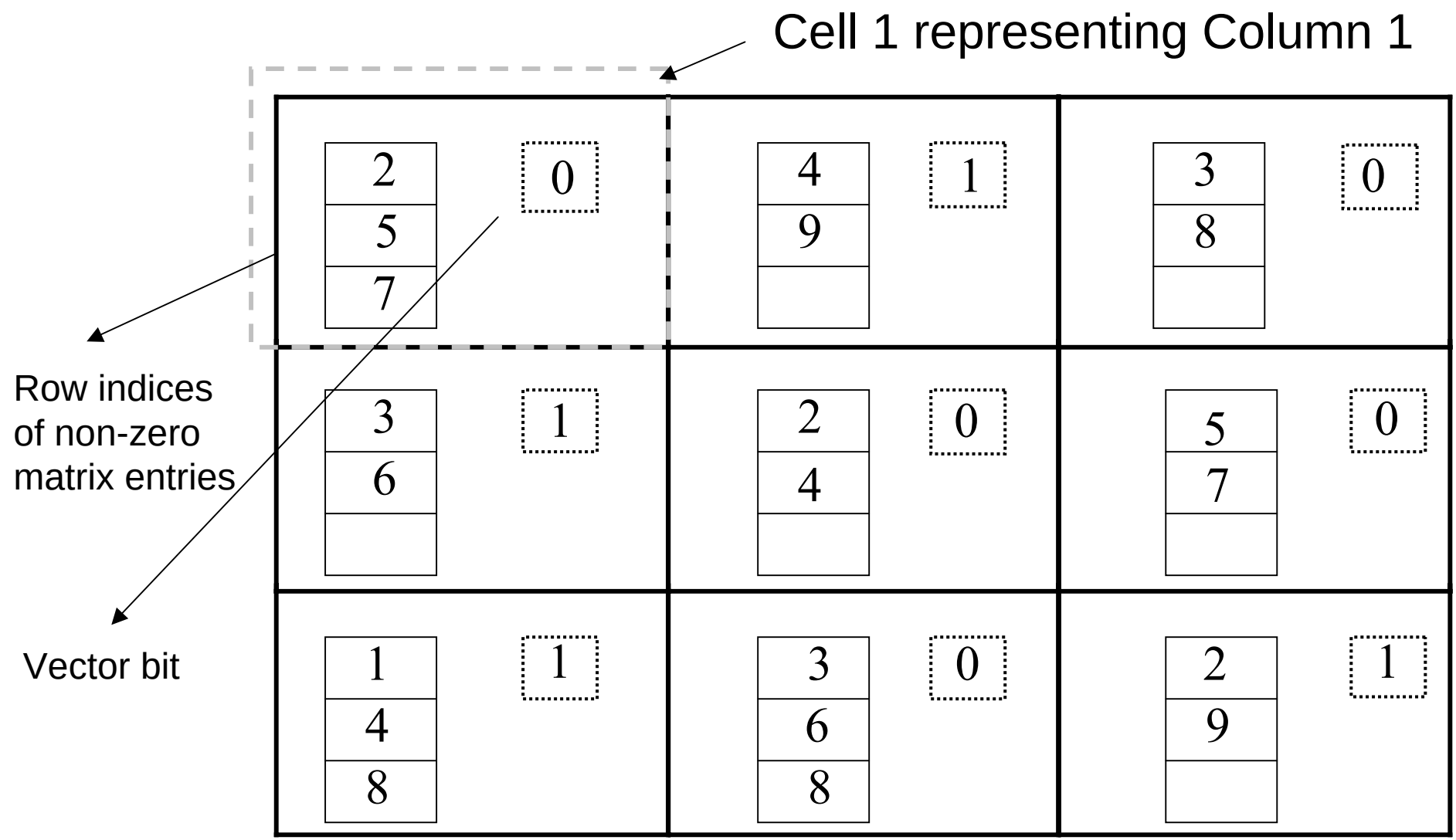
Mesh Routing

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

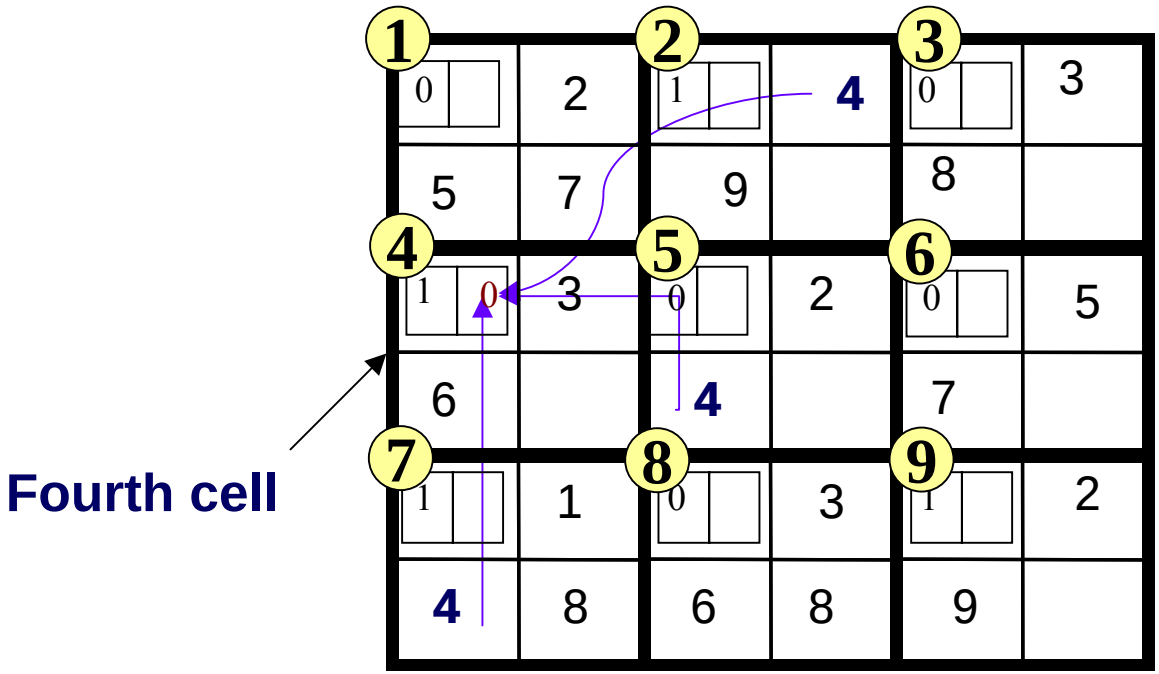


d = maximum number of non-zero entries per column

Mesh corresponding to the matrix A and vector v

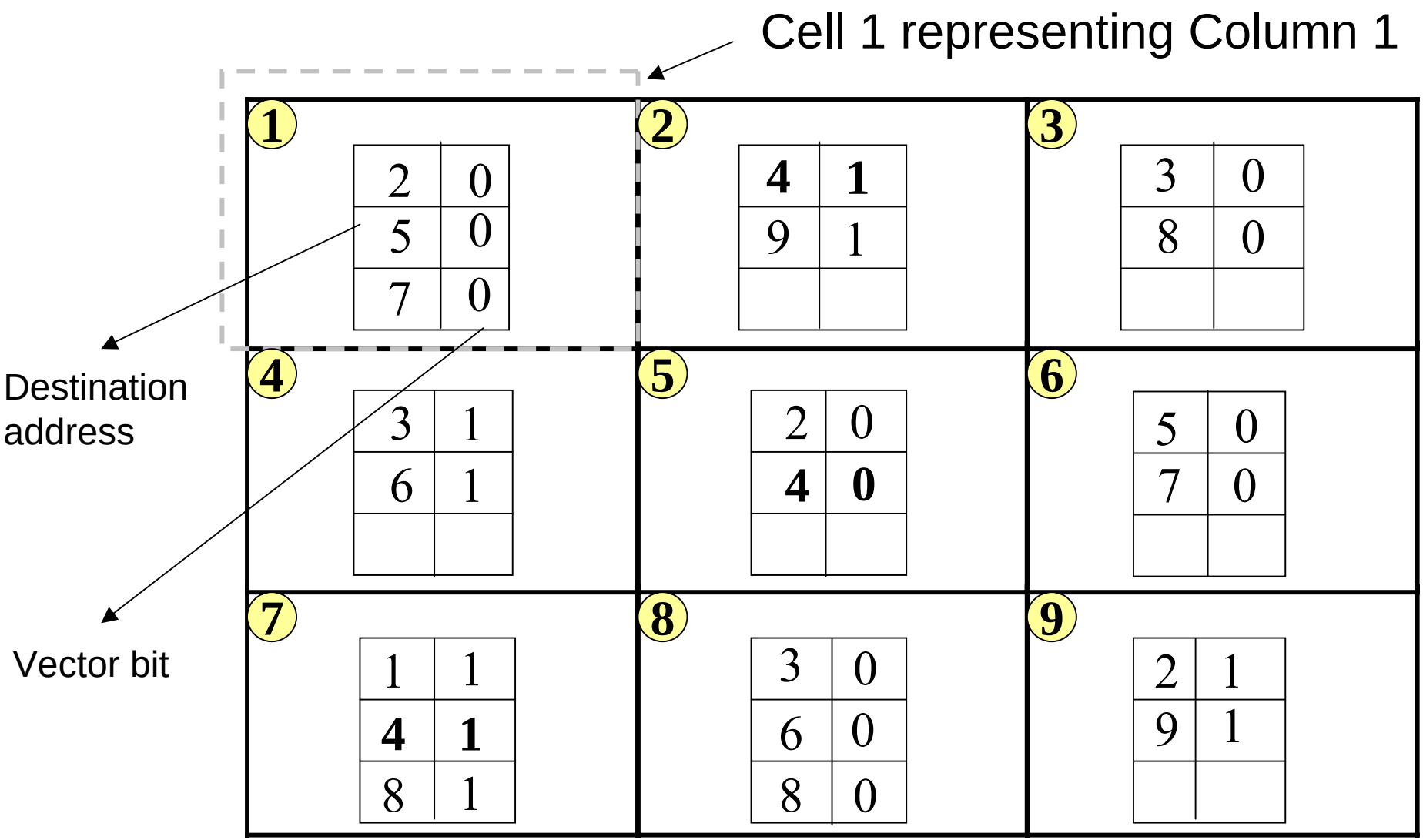


Routing in the Mesh

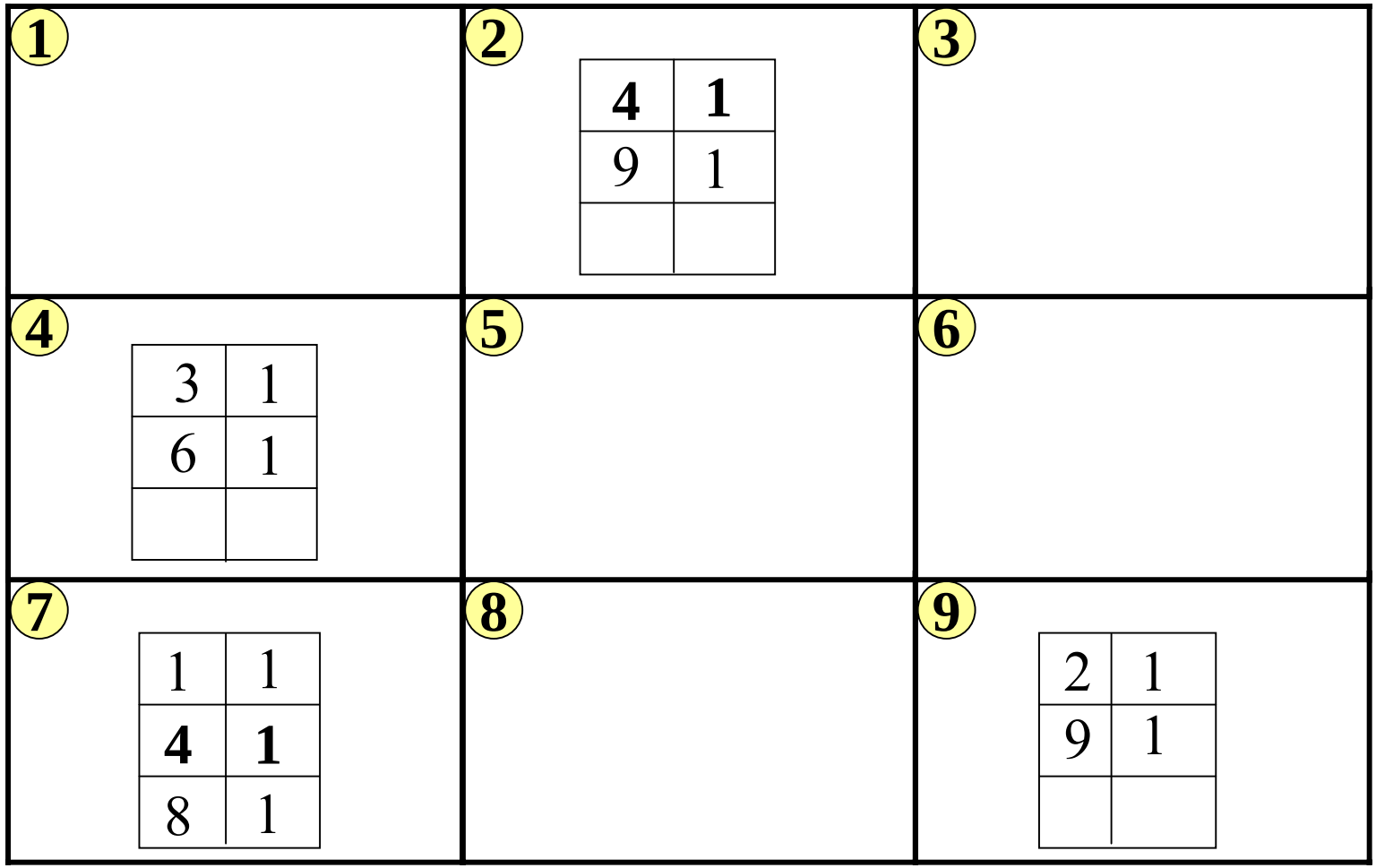


Each time a packet arrives at the target cell, the packet's vector's bit is xored with the partial result bit on the target cell

Packets generated by each cell in the mesh



Only packets with non-zero vector bits need to be routed



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

v_2

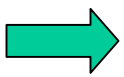
1	0	0	1	0	1	0	1	0

v_1

0	1	0	1	0	0	1	0	1

A

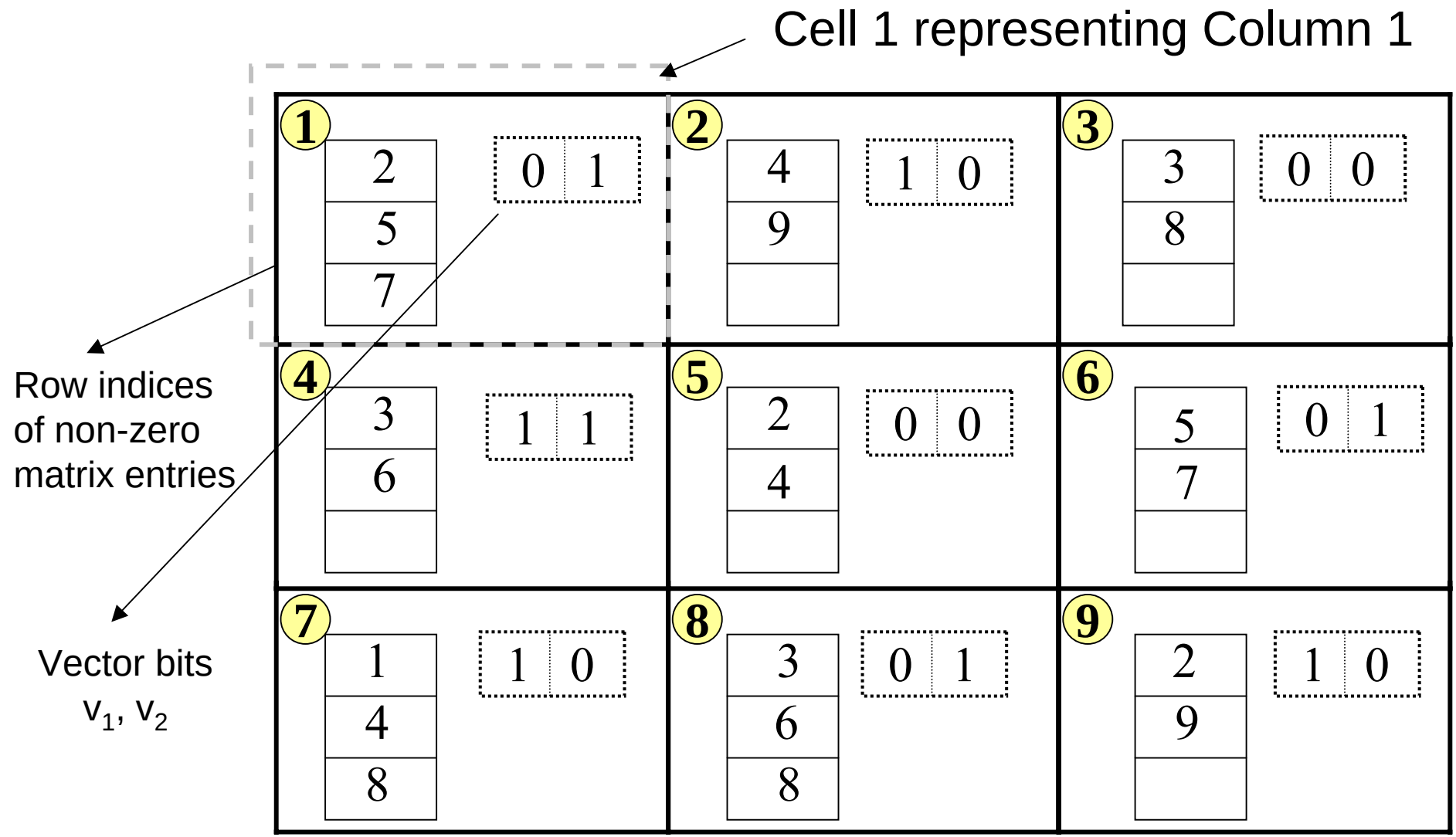
						1		
1				1				1
		1	1				1	
	1			1		1		
1				1	1			
			1				1	
1					1			
		1				1	1	
	1							1



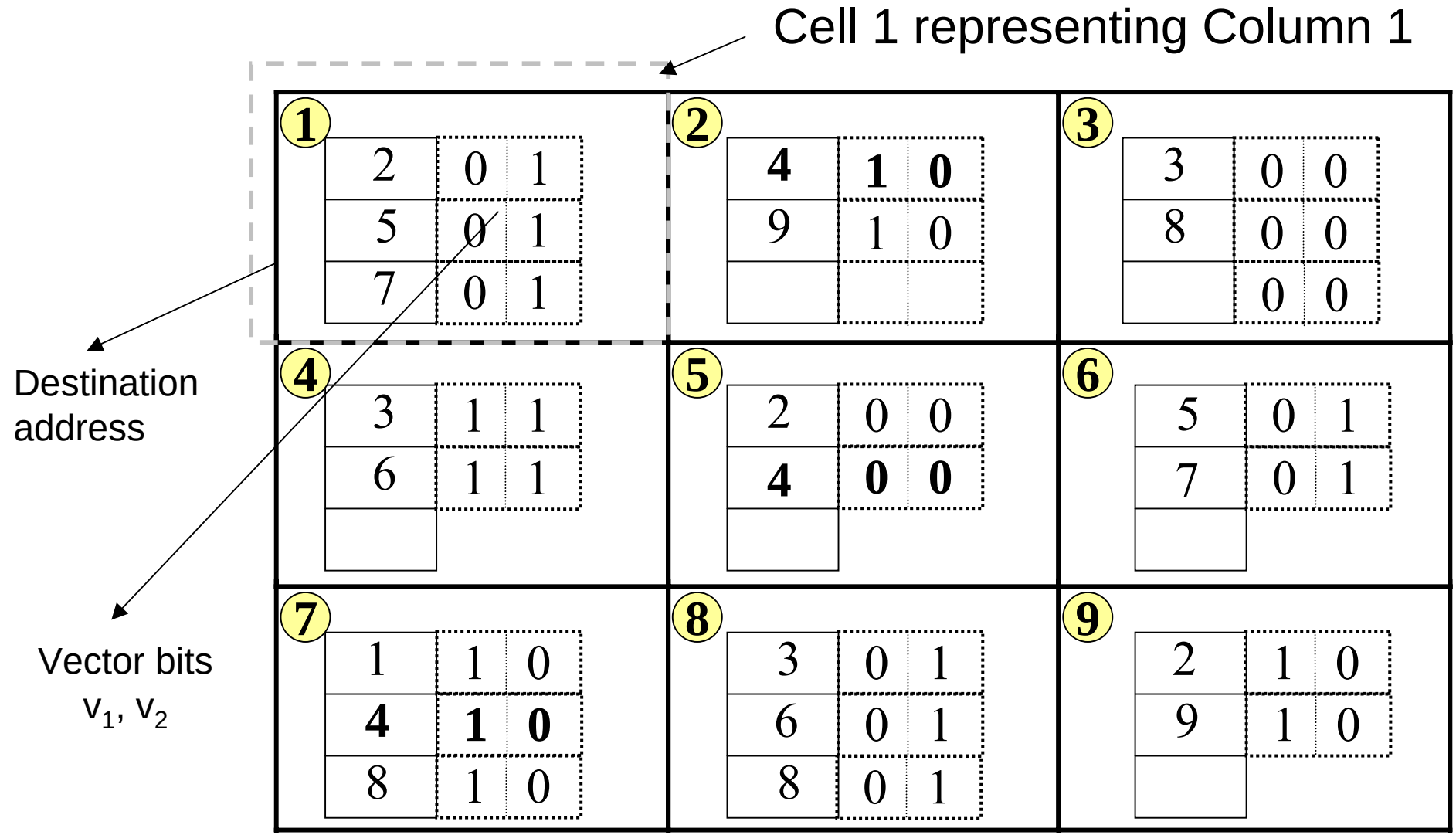
1 0	2	0 1	4	0 0	3
5	7	9		8	
1 1	3	0 0	2	1 0	5
6		4		7	
0 1	1	1 0	3	0 1	2
4	8	6	8	9	

mesh

Mesh corresponding to the matrix A and vectors v_1, v_2



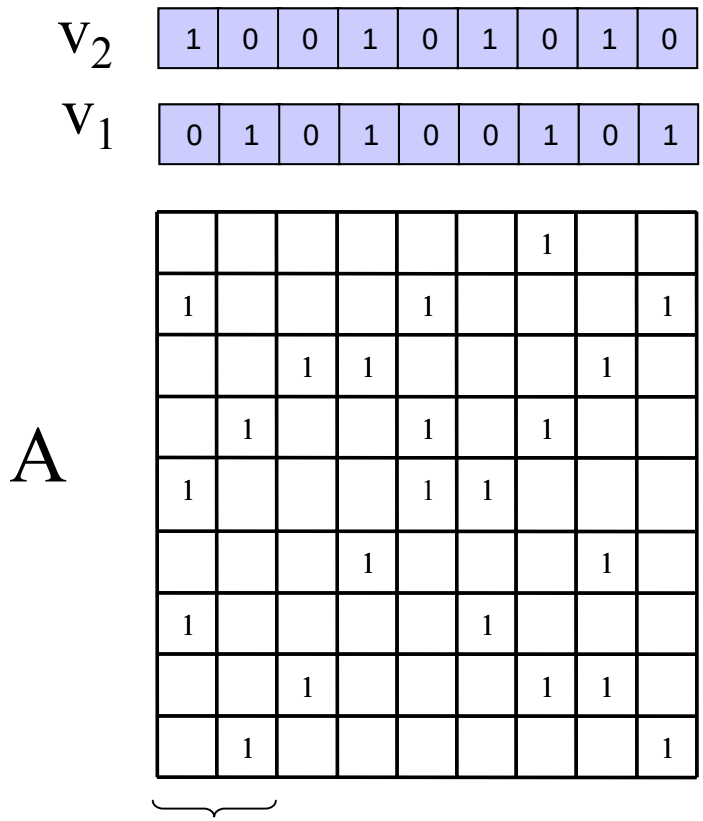
Packets with multiple vector bits generated by each cell in the mesh



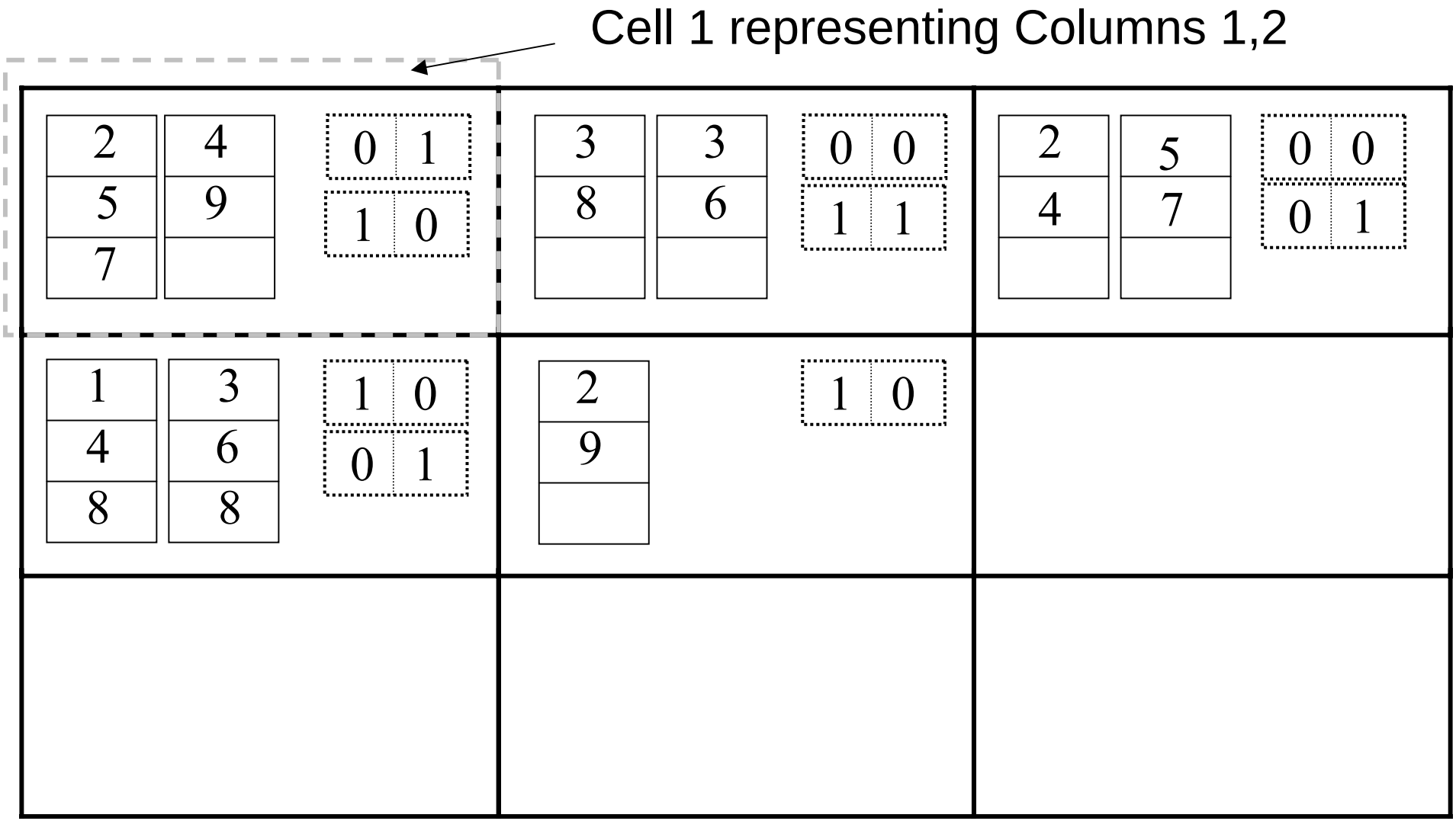
Mesh Routing with p multiple columns and K vectors

Example for p=2, K=2

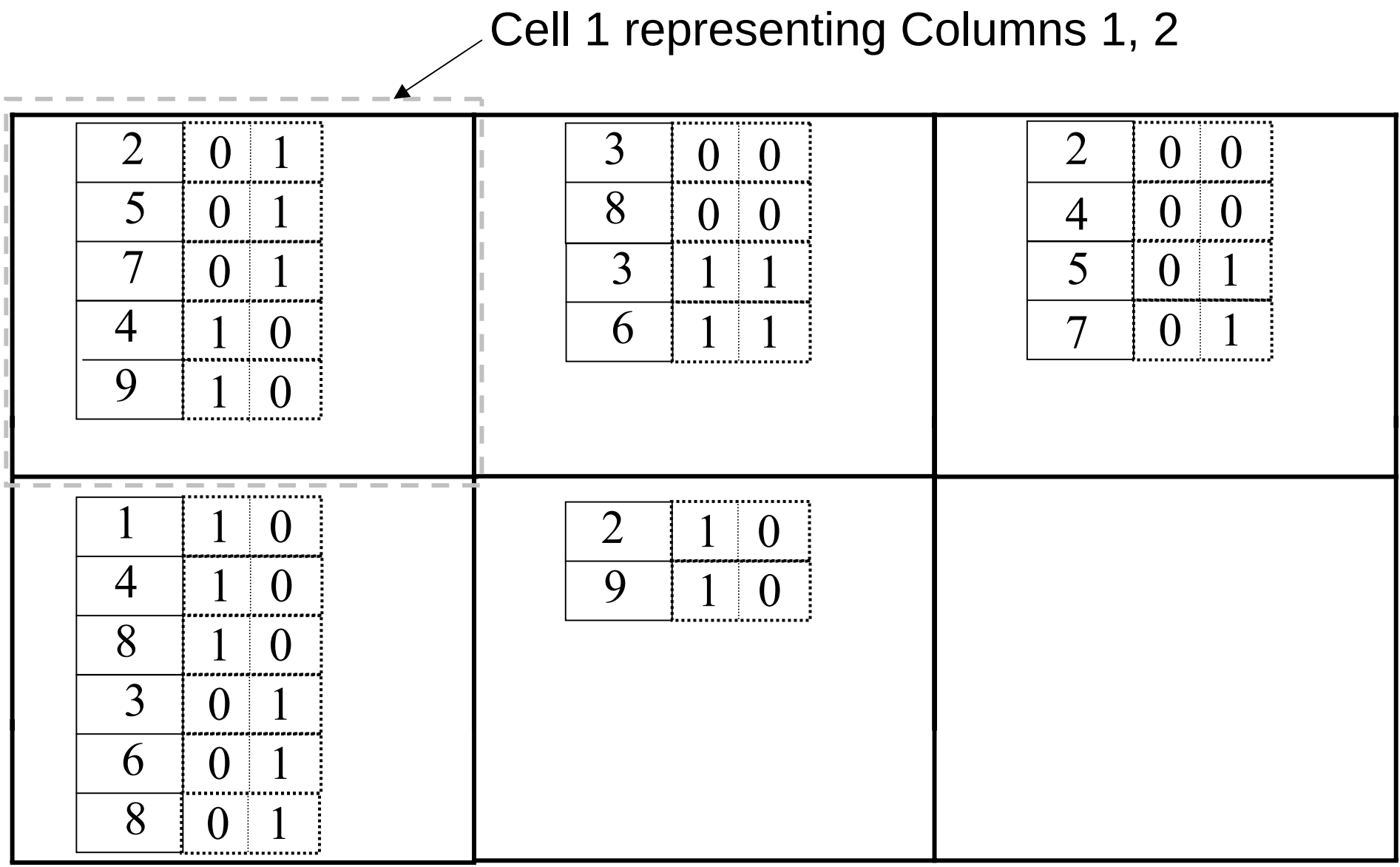
$A \cdot v_1$ and $A \cdot v_2$ computed in parallel



Mesh with multiple columns and multiple vectors per cell

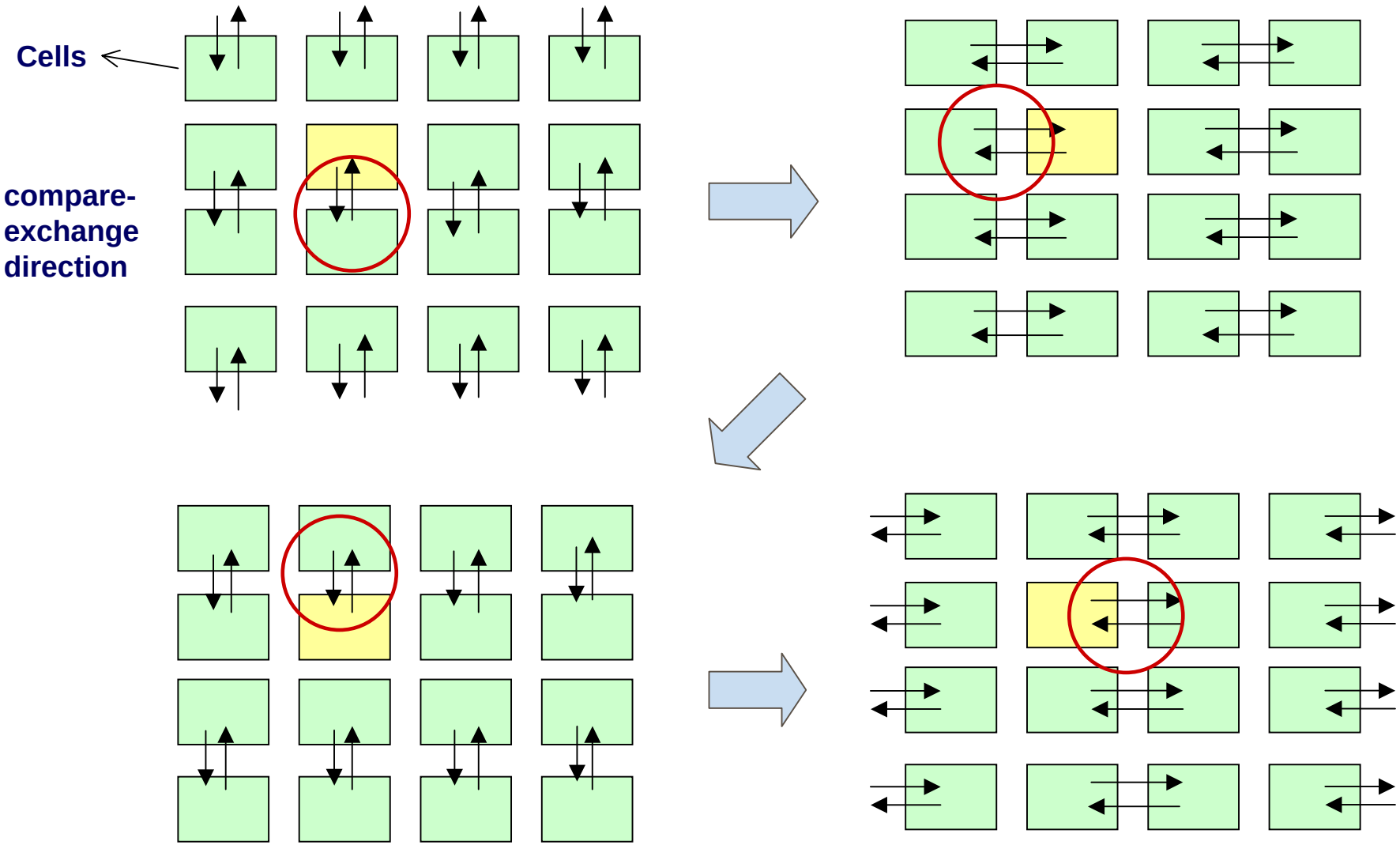


Packets generated by the mesh with multiple columns and multiple vectors per cell



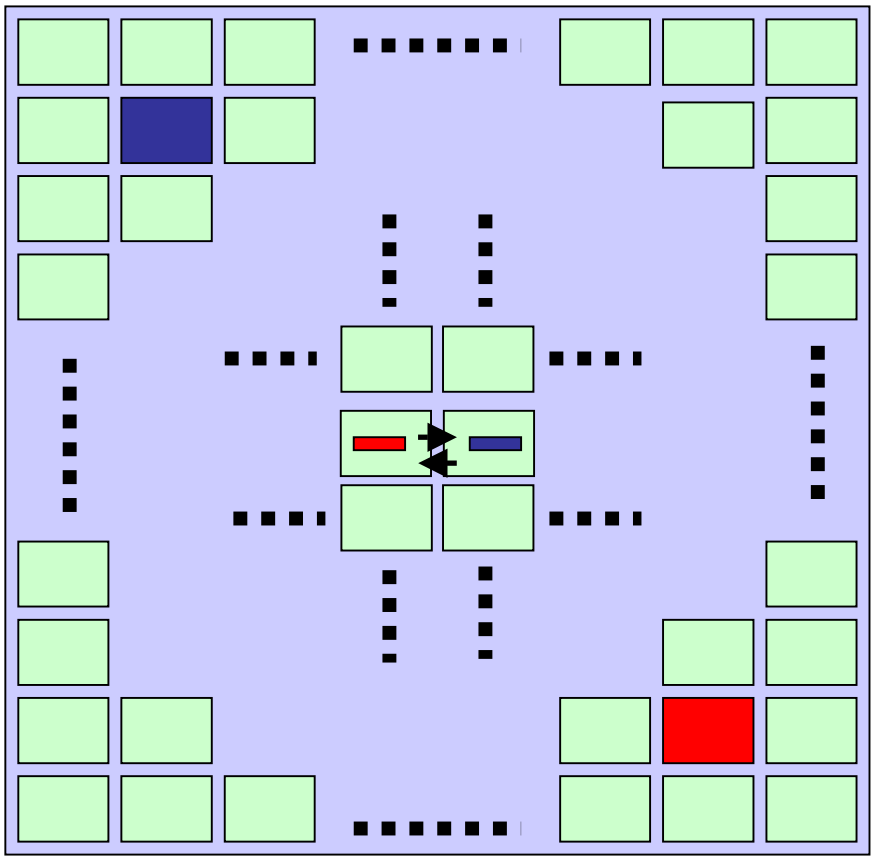
Clockwise Transposition Routing

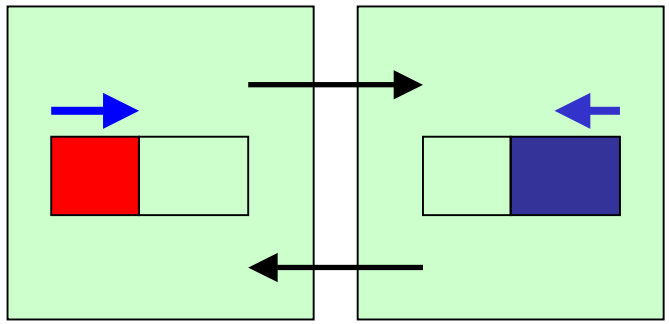
Four iterations repeated



Compare-Exchange Cases

Between Columns

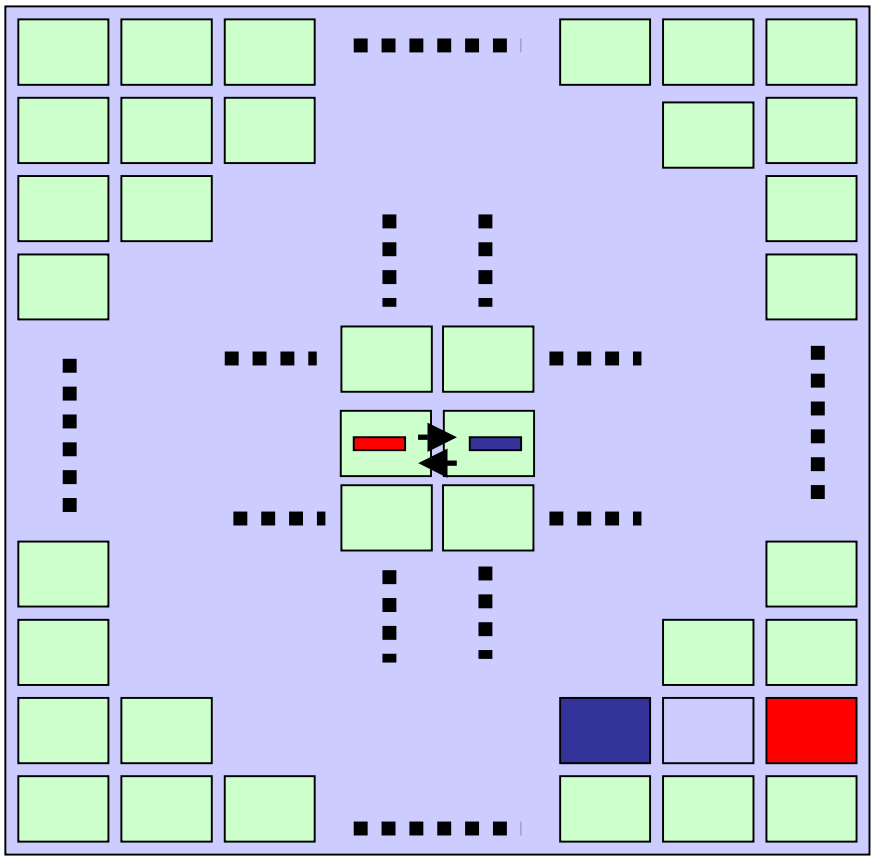


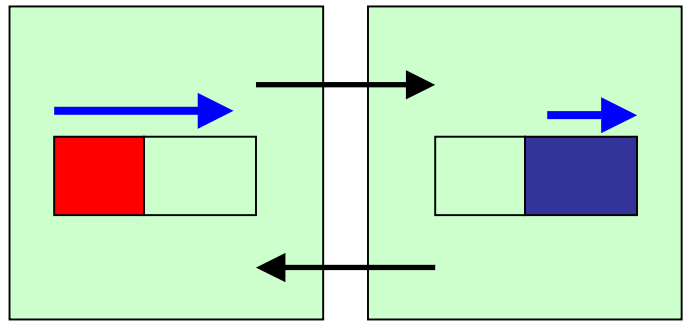


- **Direction of travel = direction of exchange**
- **Result of comparison = Exchange the packets.**

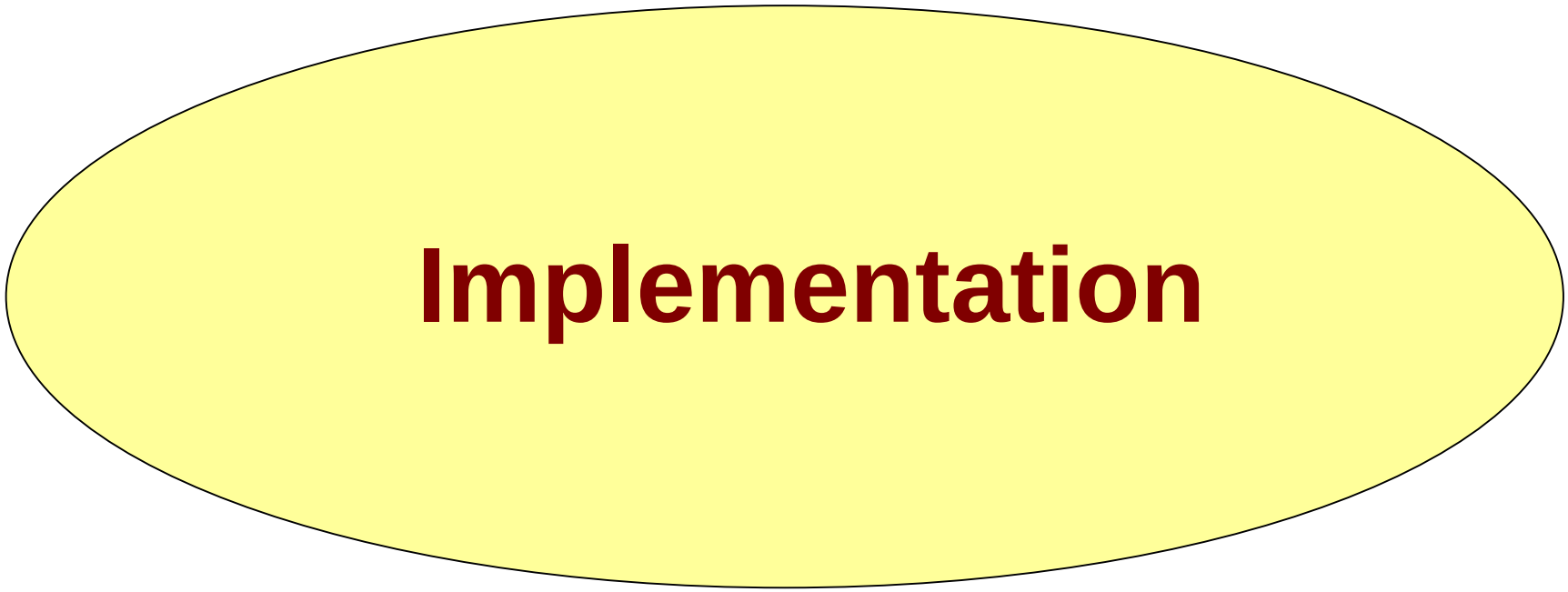
Compare-Exchange Cases

Between Columns

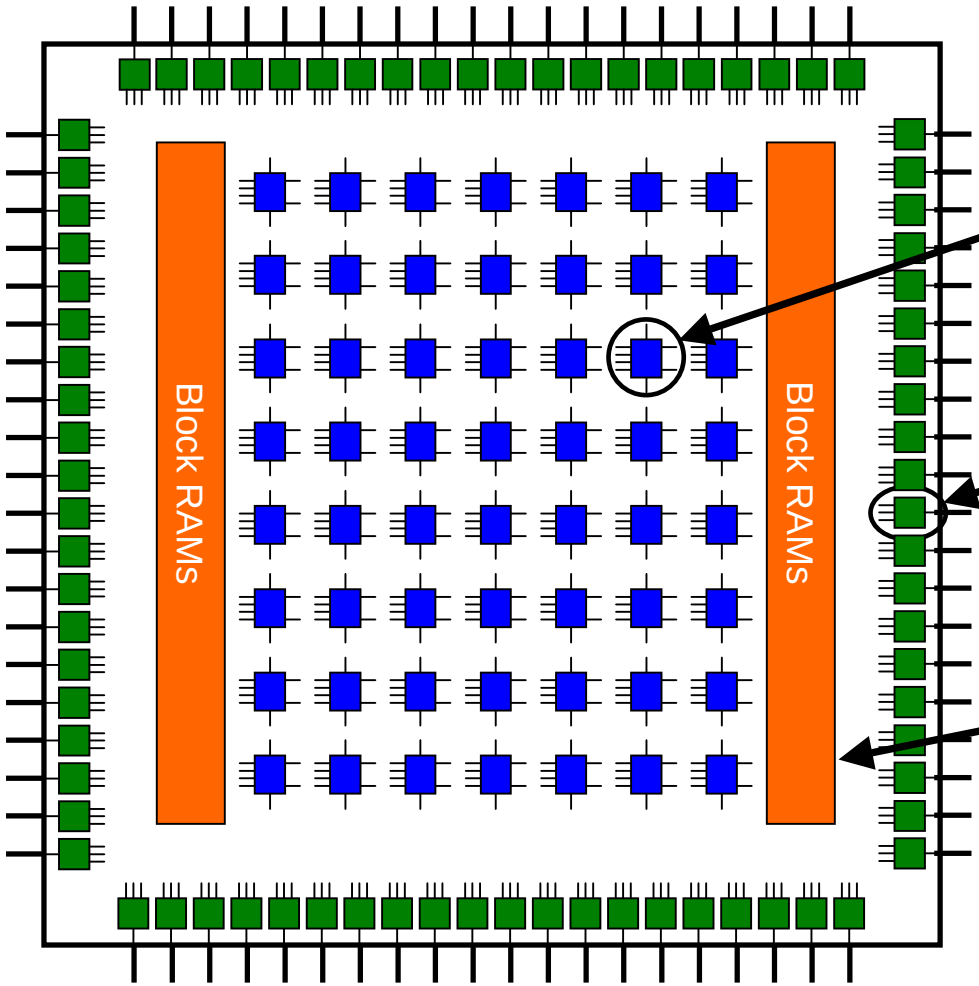




- **Direction of travel ≠ direction of exchange**
- **Result of comparison = Exchange the packets.**
- **Rule for Exchange: Exchange iff the distance to target of the packet which is farthest from its destination gets reduced.**


Implementation

Structure of the Xilinx Virtex FPGA

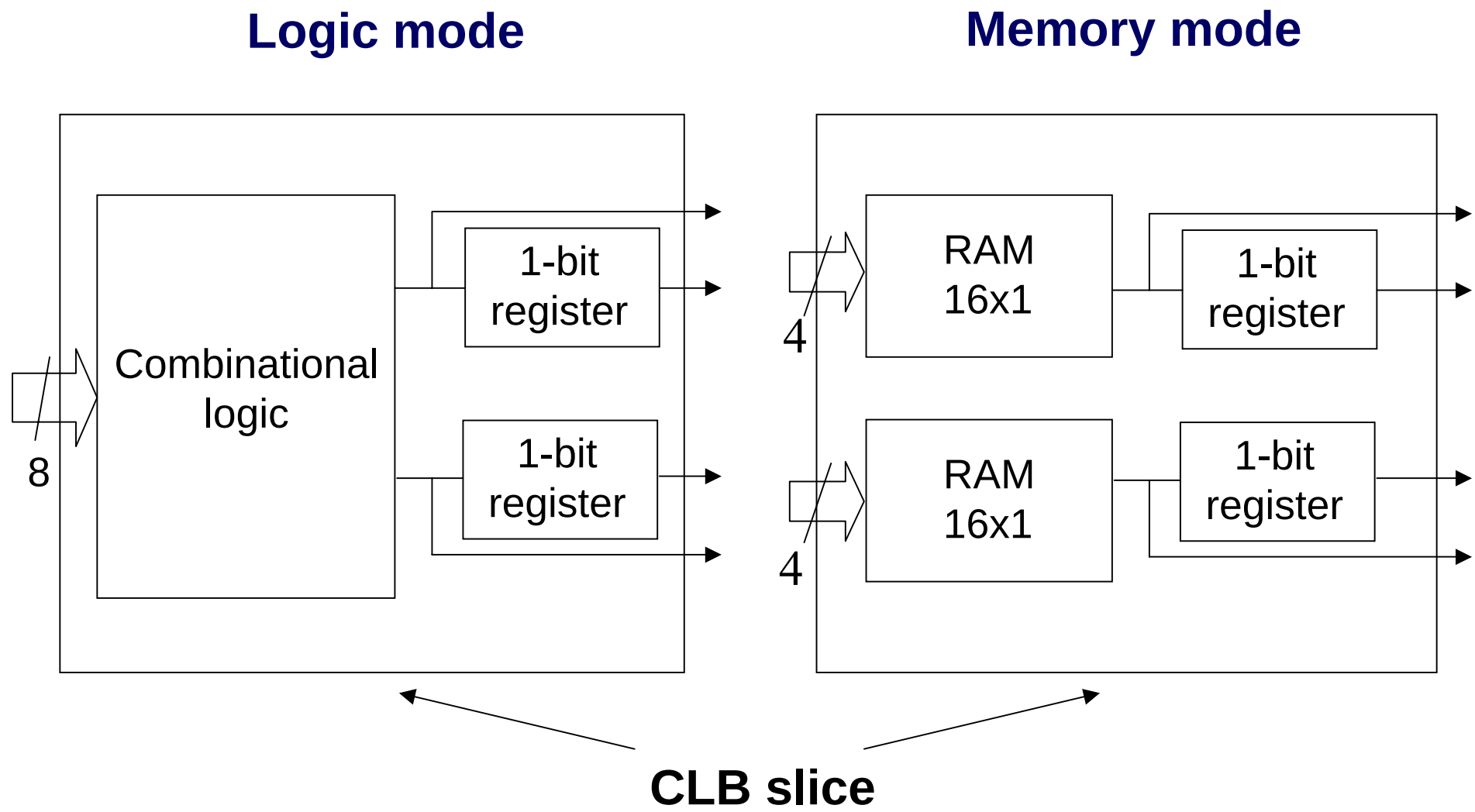


Configurable Logic Block (CLB) slices

I/O Blocks

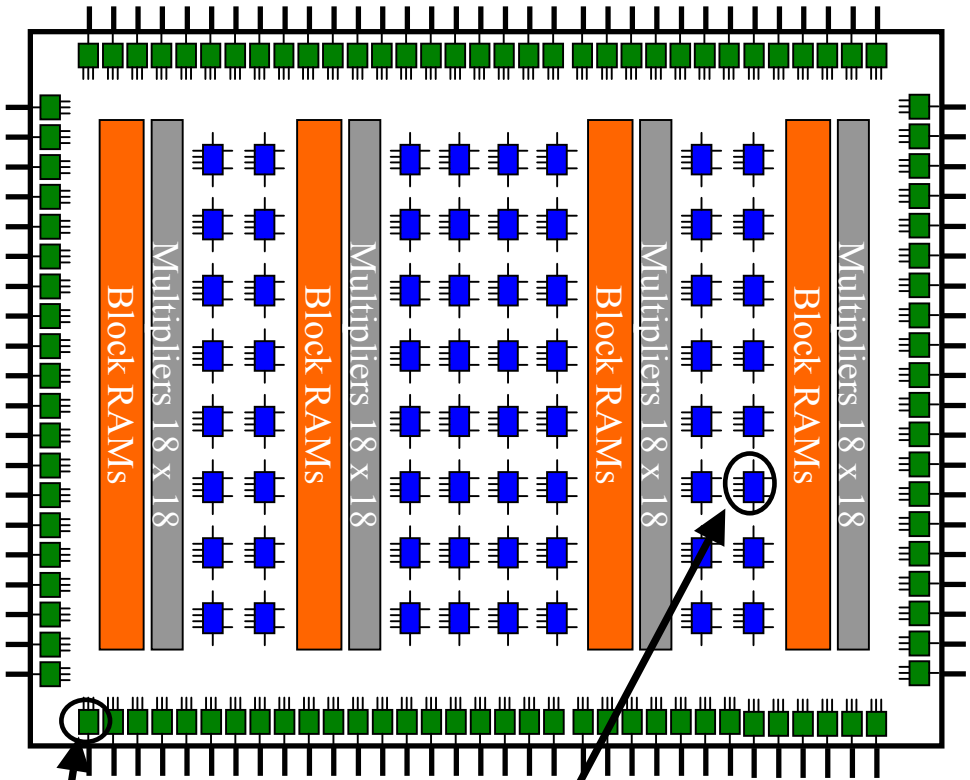
Block RAMs

Two modes of operation of CLB slices



Target FPGA Device

Xilinx Virtex II

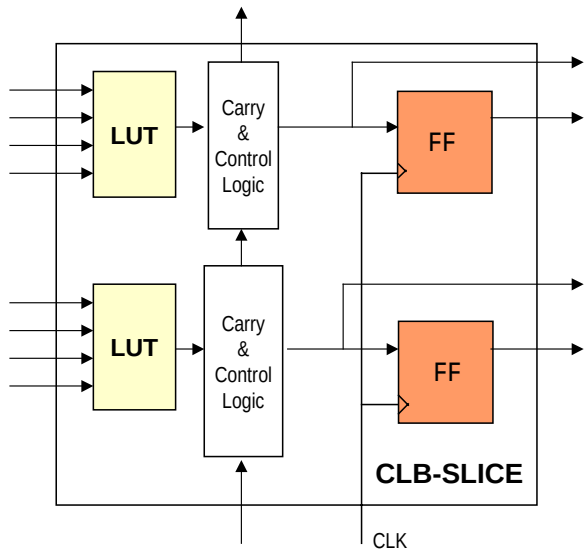


I/O Block

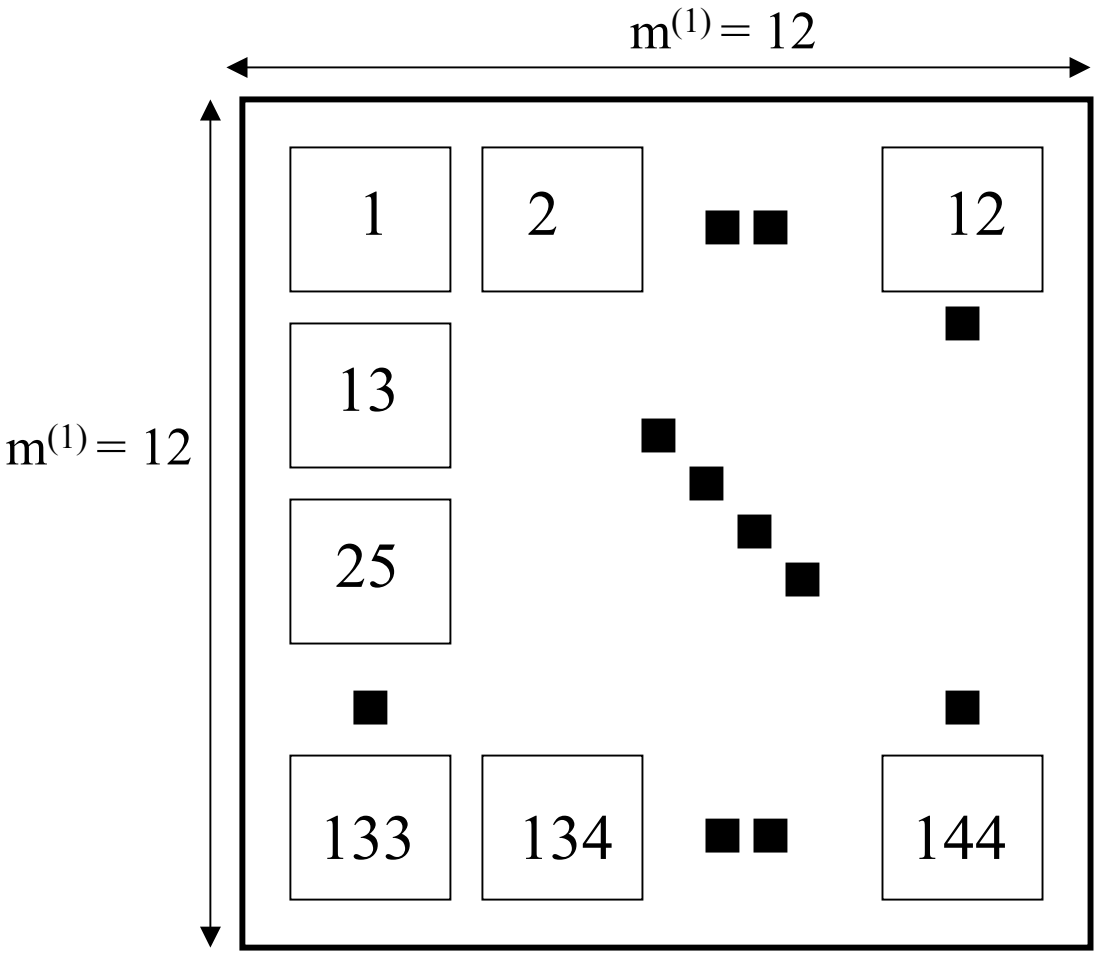
CLB slice

XC2V8000

- 46,592 CLB slices
- 93,184 LUTs
  LookUpTables
- 93,184 FF (Flip-Flop)



Mesh parameters for single FPGA



$m^{(1)} = 12$

$p = 16$

$D^{(1)} = (m^{(1)})^2 * p =$

$= (12)^2 * 16 = 2304$

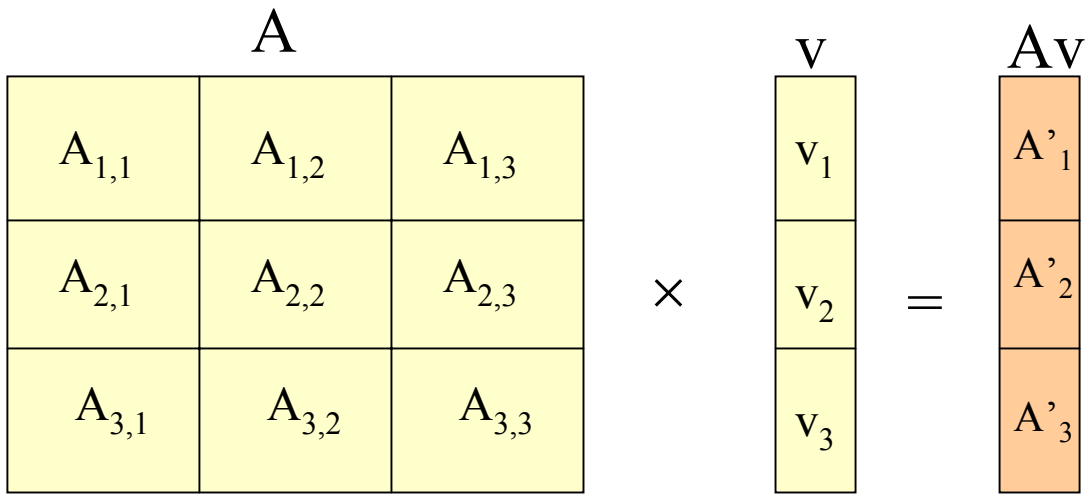
$K = 50$

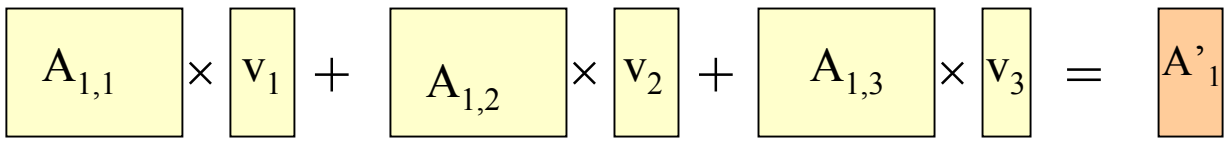
Synthesis Results on one Virtex II XC2V8000 for Improved Mesh Routing Design

Matrix Size	K	CLB slices	LUTs	FFs	Clock Period (ns)	Time for K mult (ns)	Time per 1 mult (ns)
2304x2304 (Mesh 12x12, p=16)	1	6738 (14%)	10,438 (11%)	6,279 (7%)	14.5	11136	11136
2304x2304 (Mesh 12x12, p=16)	32	29,938 (64%)	50,983 (54%)	19,651 (21%)	16.7	12826	401
2304x2304 (Mesh 12x12, p=16)	50	43,402 (93%)	74,030 (89%)	27,406 (29%)	17.7	13593	271

$f_{CLK\text{-}ROUTE} = 55.5$ MHz $T_{CLK\text{-}ROUTE} = 18$ ns

Distributed Computation
(Geiselmann, Steinwandt, CHES 2003)



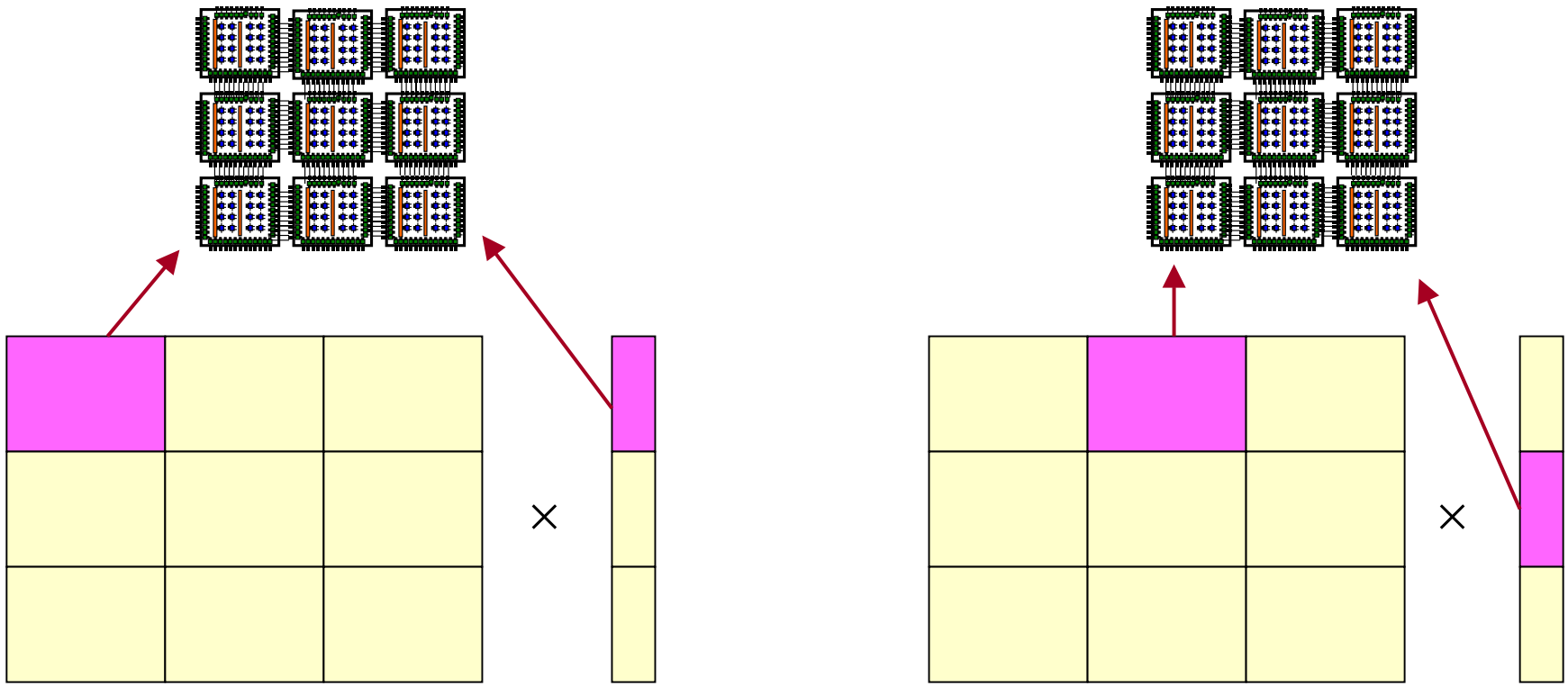


$$A \cdot v = \begin{pmatrix} \sum_{j=1}^{s} A_{1,j} \cdot v_j \\ \vdots \\ \sum_{j=1}^{s} A_{s,j} \cdot v_j \end{pmatrix}$$

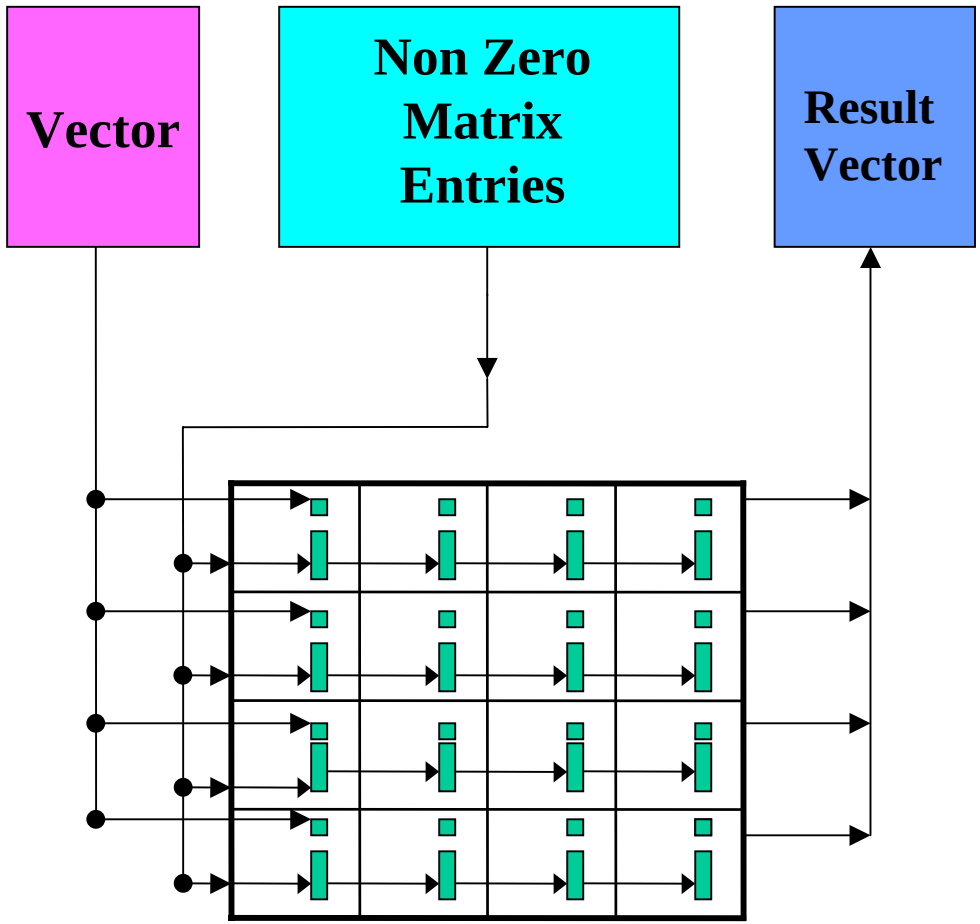
Using smaller FPGA arrays to perform the entire computation

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

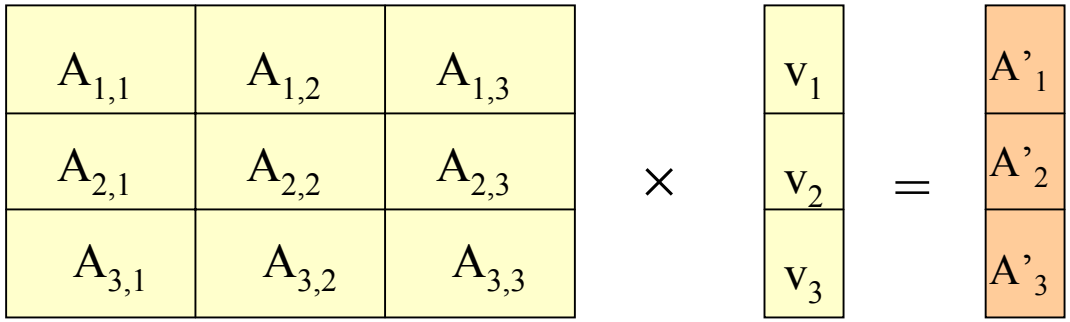
2) Reuse FPGA array for next sub-computation

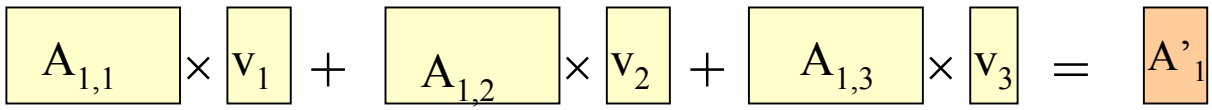


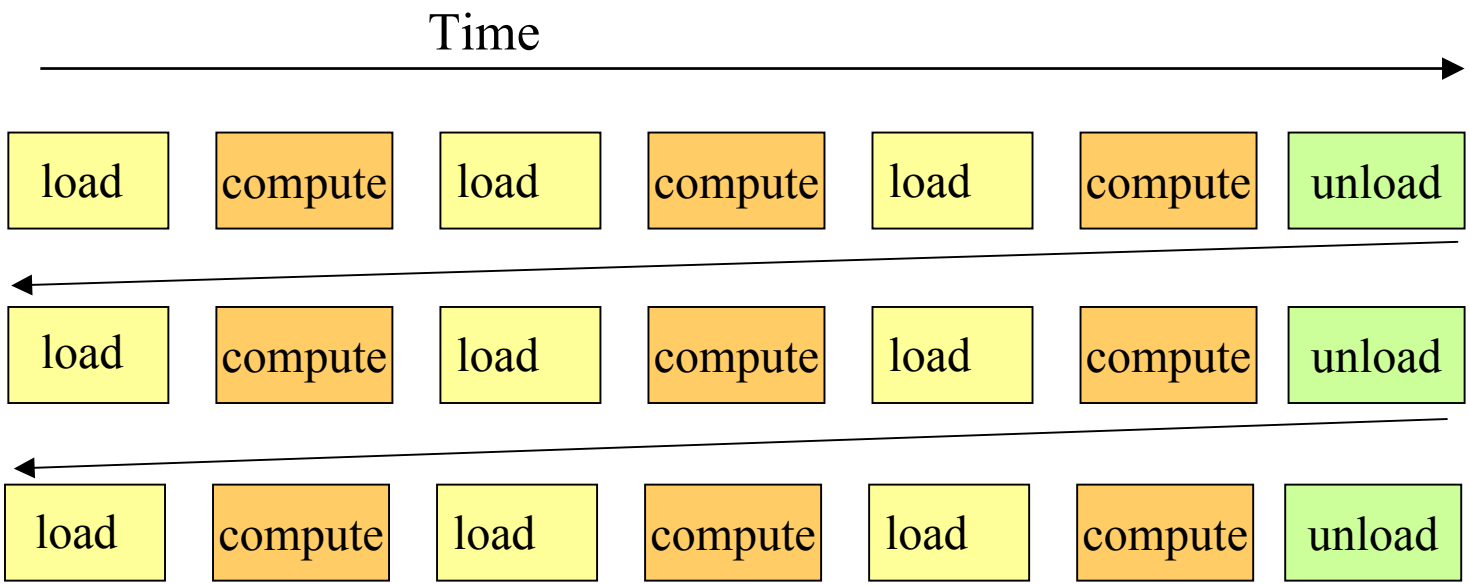
Parallel Loading & Unloading of Data



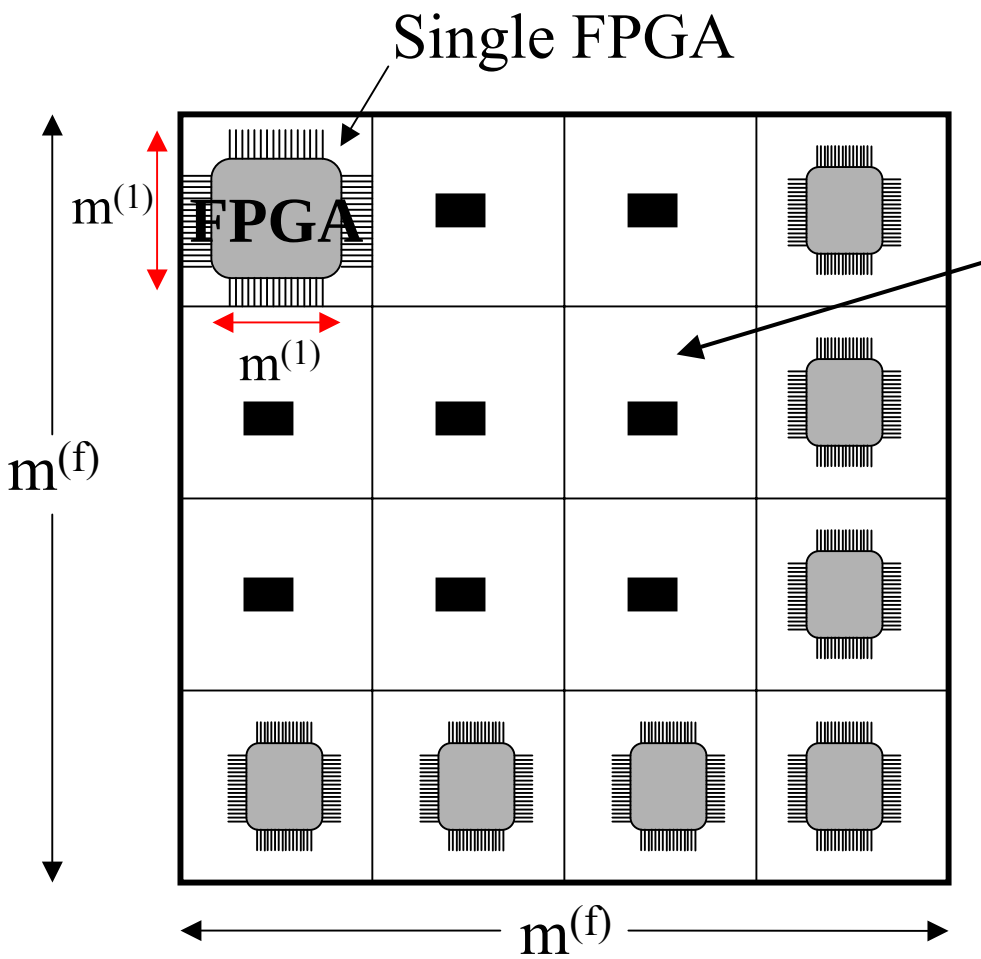
Sub matrix load-compute sequence







Size of the mesh implemented using f FPGAs

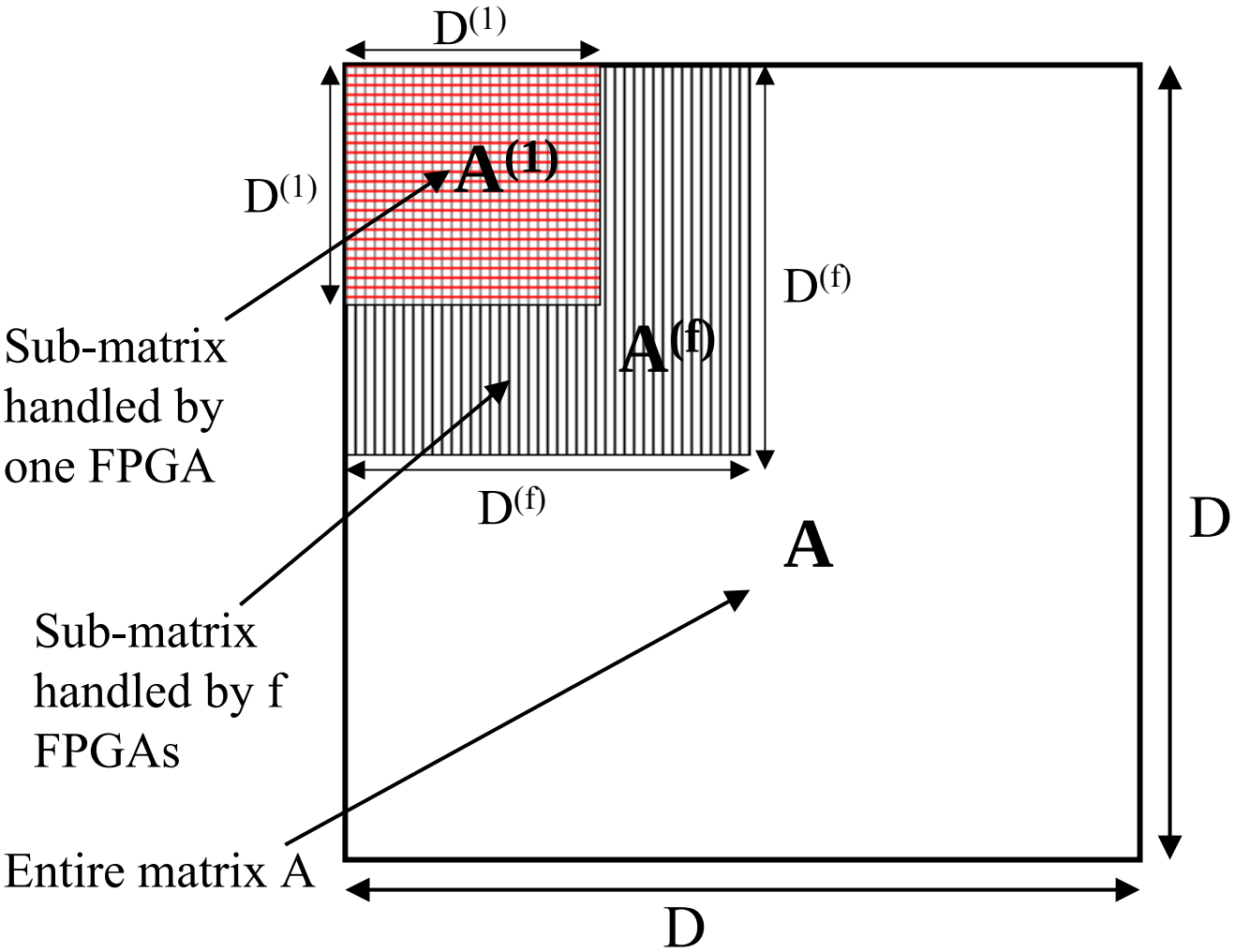


$$\#cells^{(f)} = \#cells^{(1)} \; f$$

$$(m^{(f)})^2 = (m^{(1)})^2 \; f$$

$$m^{(f)} = m^{(1)} \sqrt{f}$$

Size of the matrix handled using f FPGAs



$$D^{(1)} = (m^{(1)})^2 * p$$

$$D^{(f)} = D^{(1)} * f$$

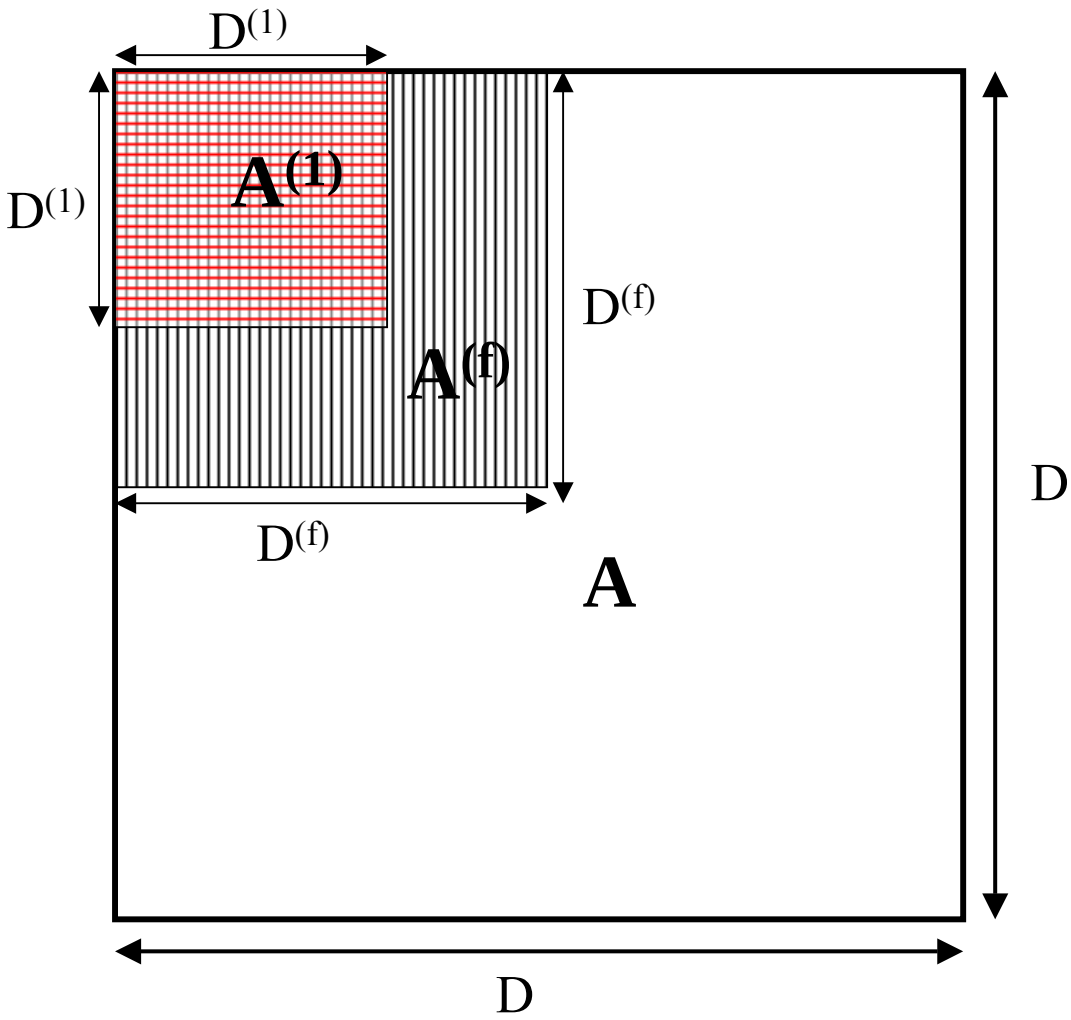
$$d^{(f)} = \lceil d * D^{(f)}/D \rceil$$

d – column density of A= maximum number of ones per column of A

$d^{(f)}$ – column density of $A^{(f)}$

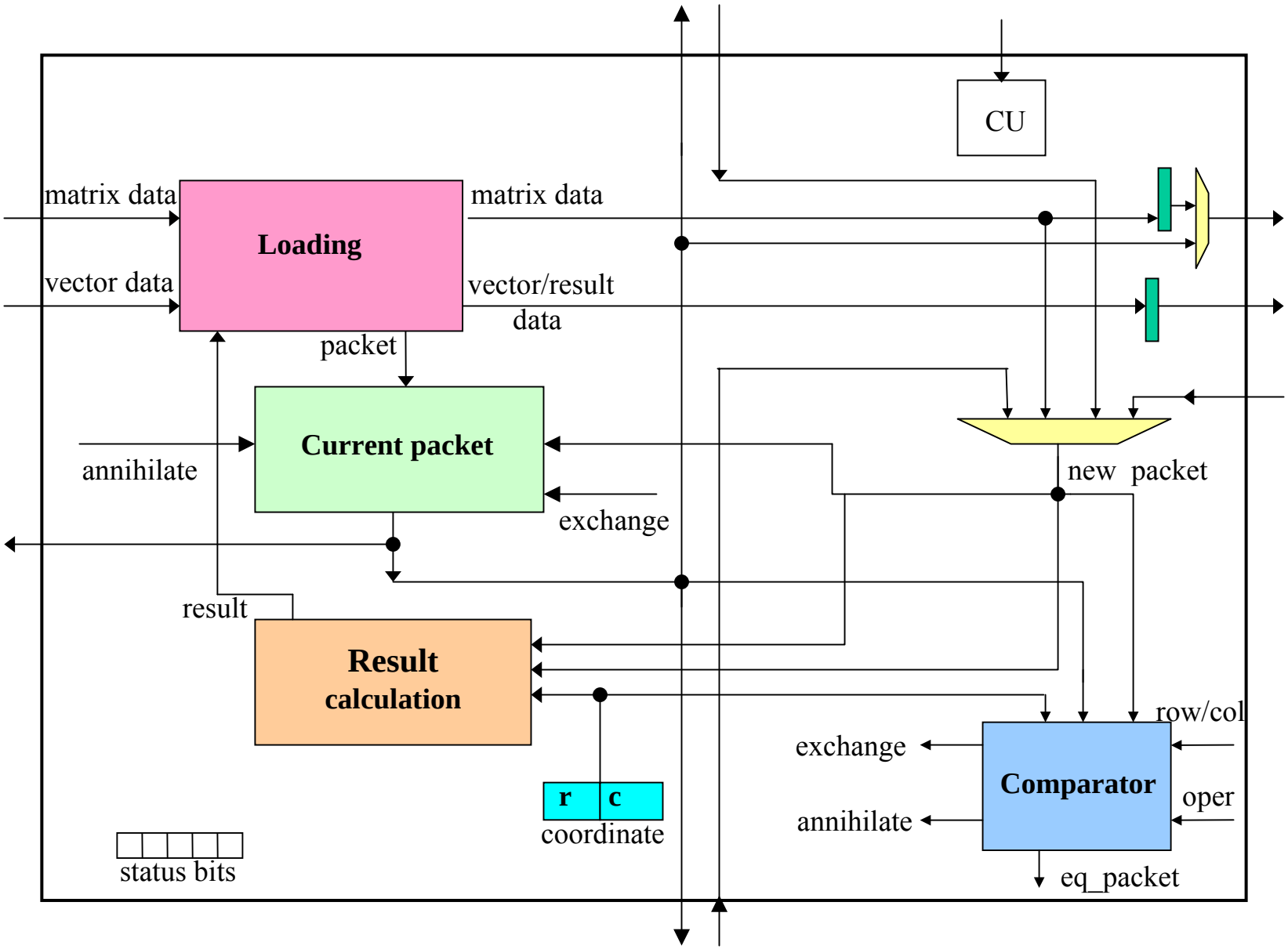
Sizes and the column densities of submatrices handled using f FPGAs

$D = 10^{10}, d = 100$



f	$D^{(f)}$	$d^{(f)}$
1	2,304	1
100	230,400	1
256	589,824	1
1024	2,359,296	1
10,000	23,040,000	1

Mesh Cell Design for Improved Mesh Routing

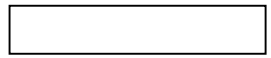


Matrix Data

Matrix data

Status	Loading Address		Routing Address	
st	r_L	c_L	r_R	c_R
1	$k^{(f)}$	$k^{(f)} + k_p$	$k^{(f)}$	$k^{(f)} + k_p$

Vector data

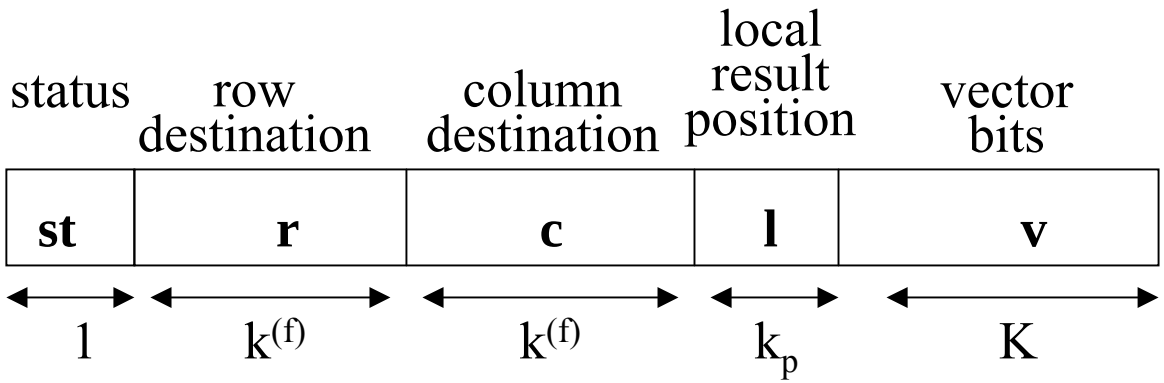


K

f (#FPGAs)	matrix data size (bits)
1	25
100	37
256	41
1024	45

$$\text{Matrix Data Size} = 1 + 4 \cdot k^{(f)} + 2 \cdot k_p = 1 + 4 \cdot \log_2 m^{(f)} + 2 \cdot \log_2 p$$

Format of the Packet



$k^{(f)} = \log_2 m^{(f)}$

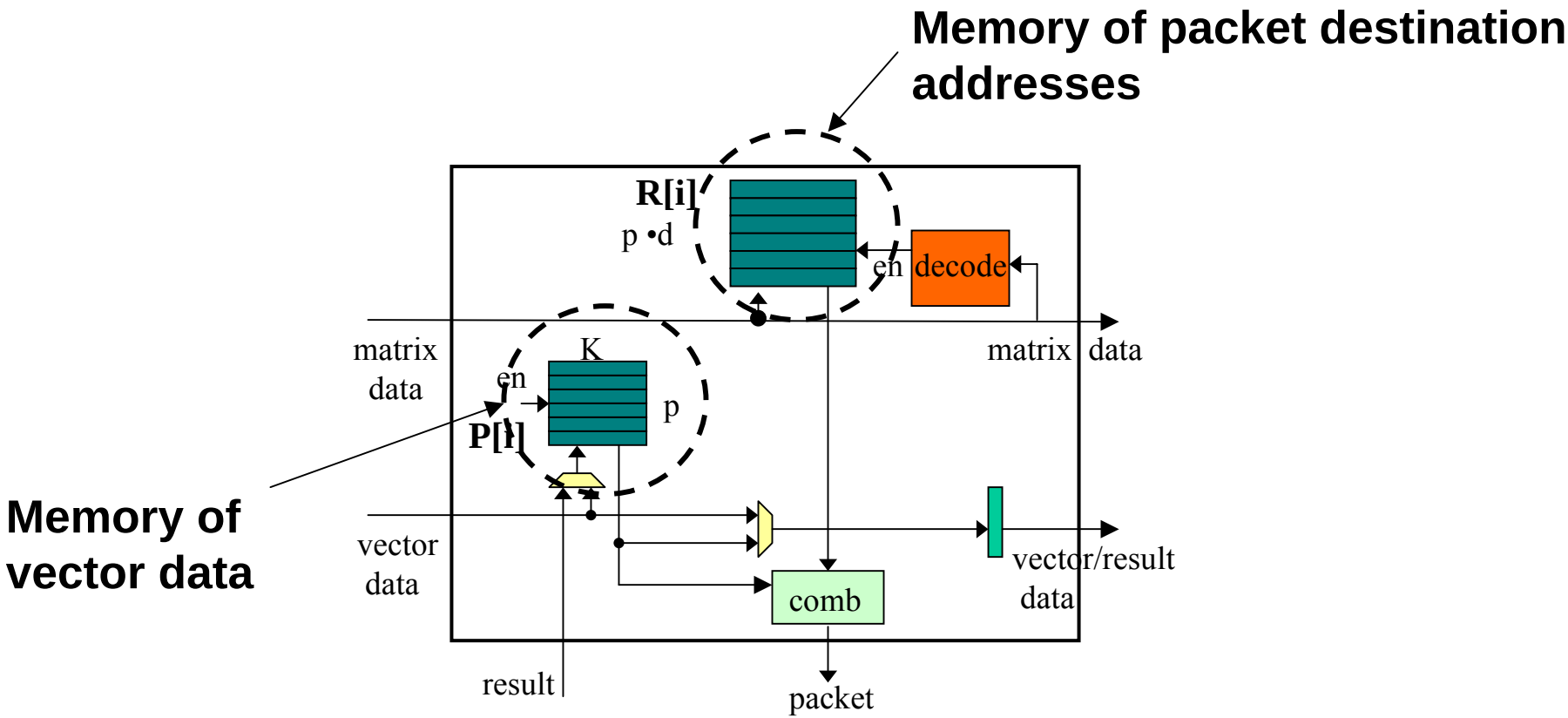
$k_p = \log_2 p$

$m^{(f)} = m^{(1)} \cdot \sqrt{f}$

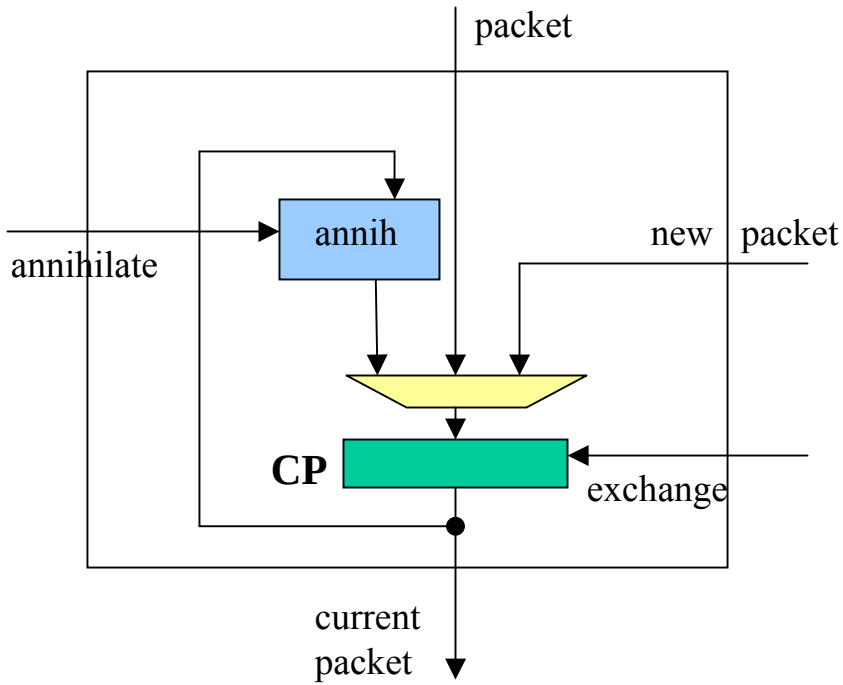
$K = 50$

# FPGAs	packet size
1	63
100	69
256	71
1024	73

Loading Unit

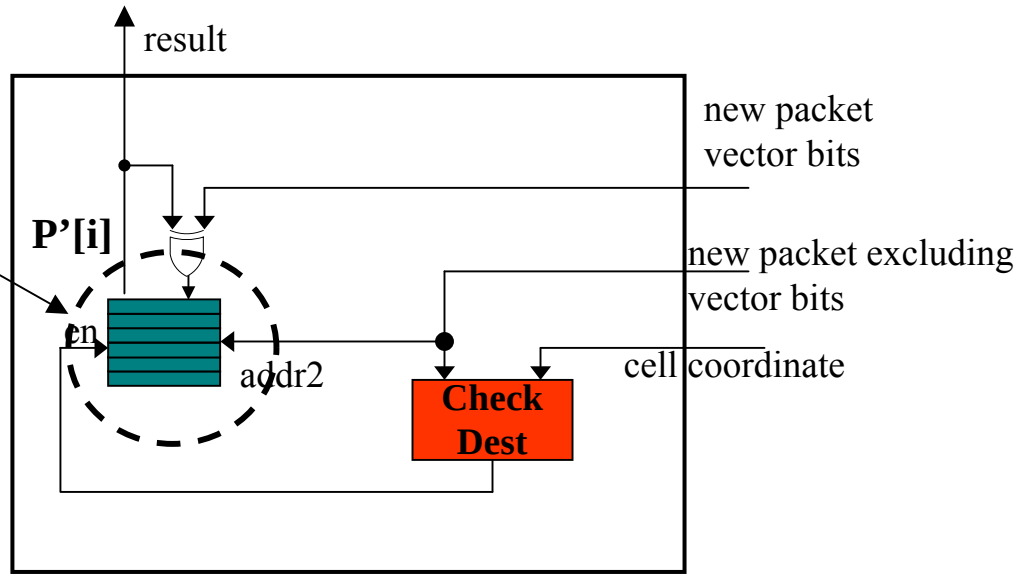


Current Packet Unit

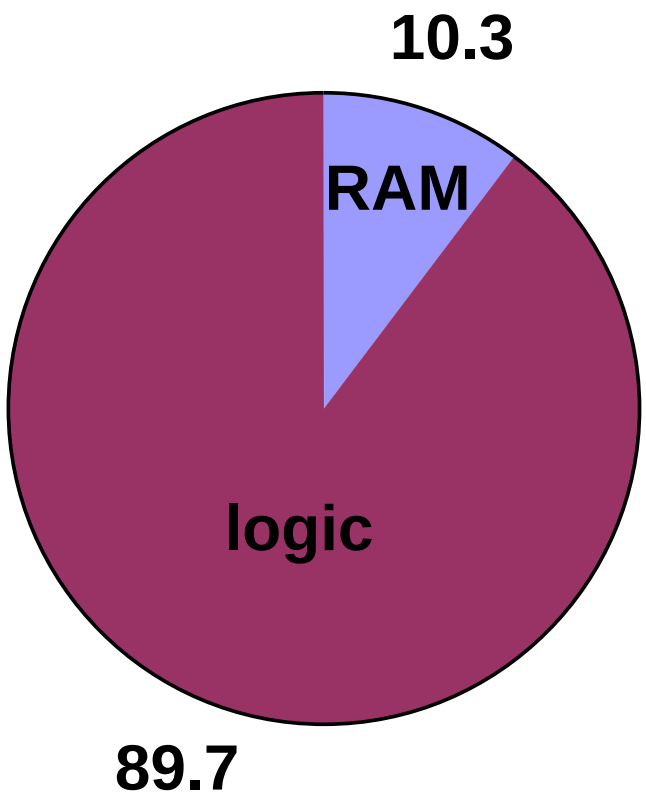


Result Calculation

Memory of the result vector bits



Resource Proportion for LUT Usage



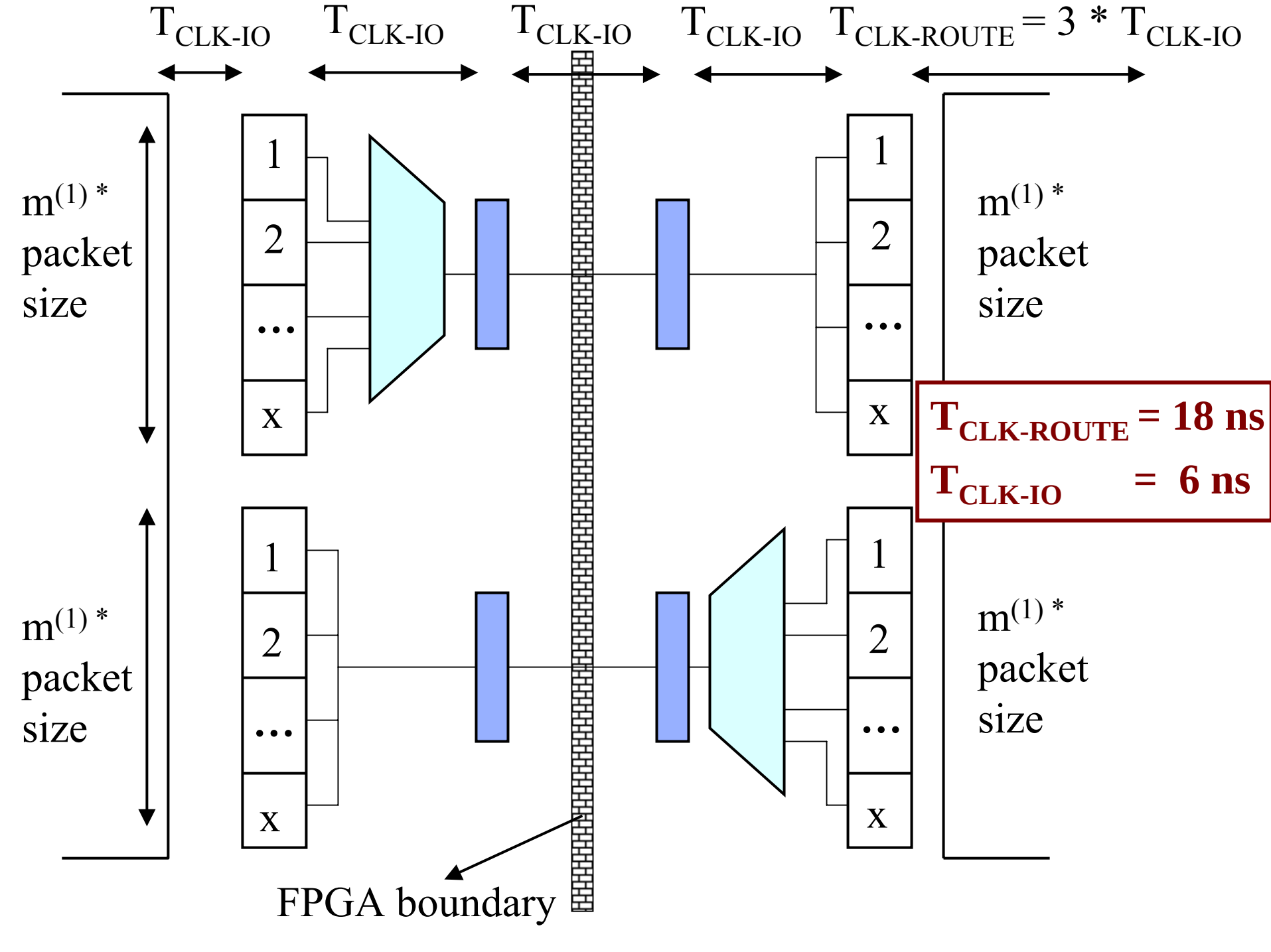
Mesh of 12x12

P=16

K=50

%LUT RAM

% LUT logic


T_{CLK-IO}
T_{CLK-IO}
T_{CLK-IO}
T_{CLK-IO}
$T_{CLK-ROUTE} = 3 * T_{CLK-IO}$
$m^{(1)}$ * packet size
1
2
...
x
$m^{(1)}$ * packet size
1
2
...
x
$T_{CLK-ROUTE}$ = 18 ns
T_{CLK-IO} = 6 ns
$m^{(1)}$ * packet size
1
2
...
x
$m^{(1)}$ * packet size
1
2
...
x
FPGA boundary

Slowdown caused by crossing the chips boundaries

$$x = \textbf{multiplexing factor} = \frac{\#\text{bits crossing the boundary}}{\#\text{ of pins per boundary}} =$$

$$= \frac{m^{(1)} * \text{packet size}^{(f)} * 2}{(\#\text{FPGA\_pins} / 4)} = \frac{12 * 73 * 2}{277} = \textbf{7}$$

for f=1024

$$\textbf{Slowdown factor} = \frac{4 \cdot T_{\text{CLK-IO}} + (x-1)\, T_{\text{CLK-IO}} + T_{\text{CLK-ROUTE}}}{T_{\text{CLK-ROUTE}}}$$

$$= \frac{10 \cdot T_{\text{CLK-IO}} + T_{\text{CLK-ROUTE}}}{T_{\text{CLK-ROUTE}}} = \textbf{4.33}$$

Results and Analysis

Results for a 512-bit number N

K= number of concurrent multiplications=50
D = number of columns in matrix A = 6.7 x10 ^{6}
$m^{(f)}$ = mesh dimension
T_{route} = time for K multiplications in the mesh
T_{Load} = time for loading and unloading for K multiplications
T_{Total} = total time for the Matrix Step = $3 \cdot (D/K) \cdot n \cdot (T_{route} + T_{Load})$
R = (#FPGAs * T_{Total})/ (1 FPGA * T_{Total} for 1 FPGA)

p =number of columns handled in one cell
$d^{(f)}$ = density of sub-matrix handled by mesh
n = number of times to repeat sub-multiplications

Virtex II chips (f)	D	p	$d^{(f)}$	$m^{(f)}$	n	T_{route} (ns)	T_{Load} (ns)	T_{Total} (days)	R
1	6.7x 10^{6}	16	1	12	8.4 x 10^{6}	13594	1568	596.6	1.00
10^2	6.7x 10^{6}	14	2	120	1105	8.9 x 10^{5}	6.1 x 10^{4}	4.9	0.82
16^2	6.7x 10^{6}	8	3	192	516	1.3 x 10^{6}	1.3x10^{5}	3.3	1.42
32^2	6.7x 10^{6}	4	6	384	129	2.5 x 10^{6}	4.3 x 10^{5}	1.8	3.07

Results for a 1024-bit number N

K= number of concurrent multiplications=50

p =number of columns handled in one cell

D = number of columns in matrix A = 10^{10}

$d^{(f)}$ = density of sub-matrix handled by mesh

$m^{(f)}$ = mesh dimension

n = number of times to repeat sub-multiplications

T_{route} = time for K multiplications in the mesh

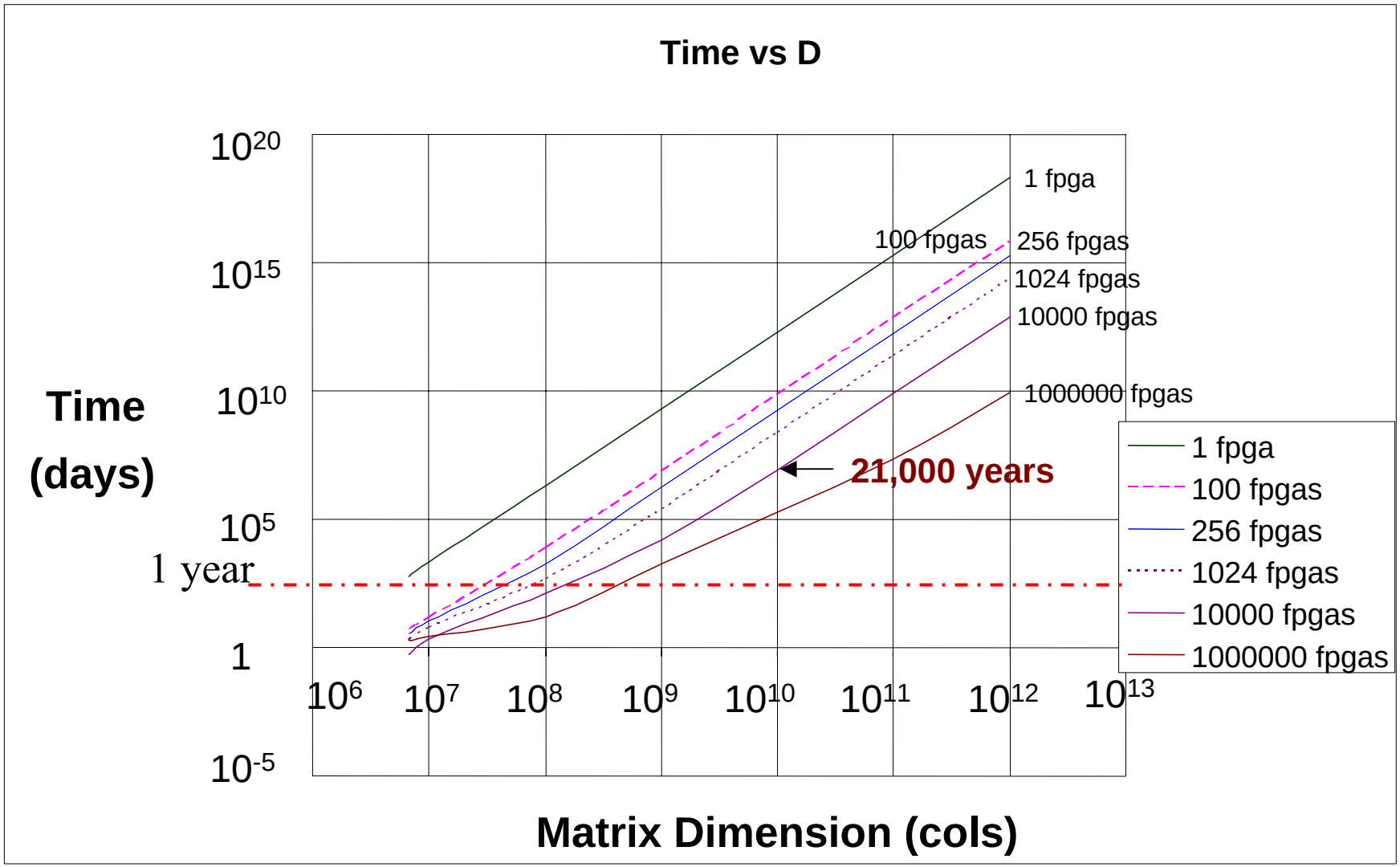
T_{Load} = time for loading and unloading for K multiplications

T_{Total} = total time for the Matrix Step = $3 \cdot (D/K) \cdot n \cdot (T_{route} + T_{Load})$

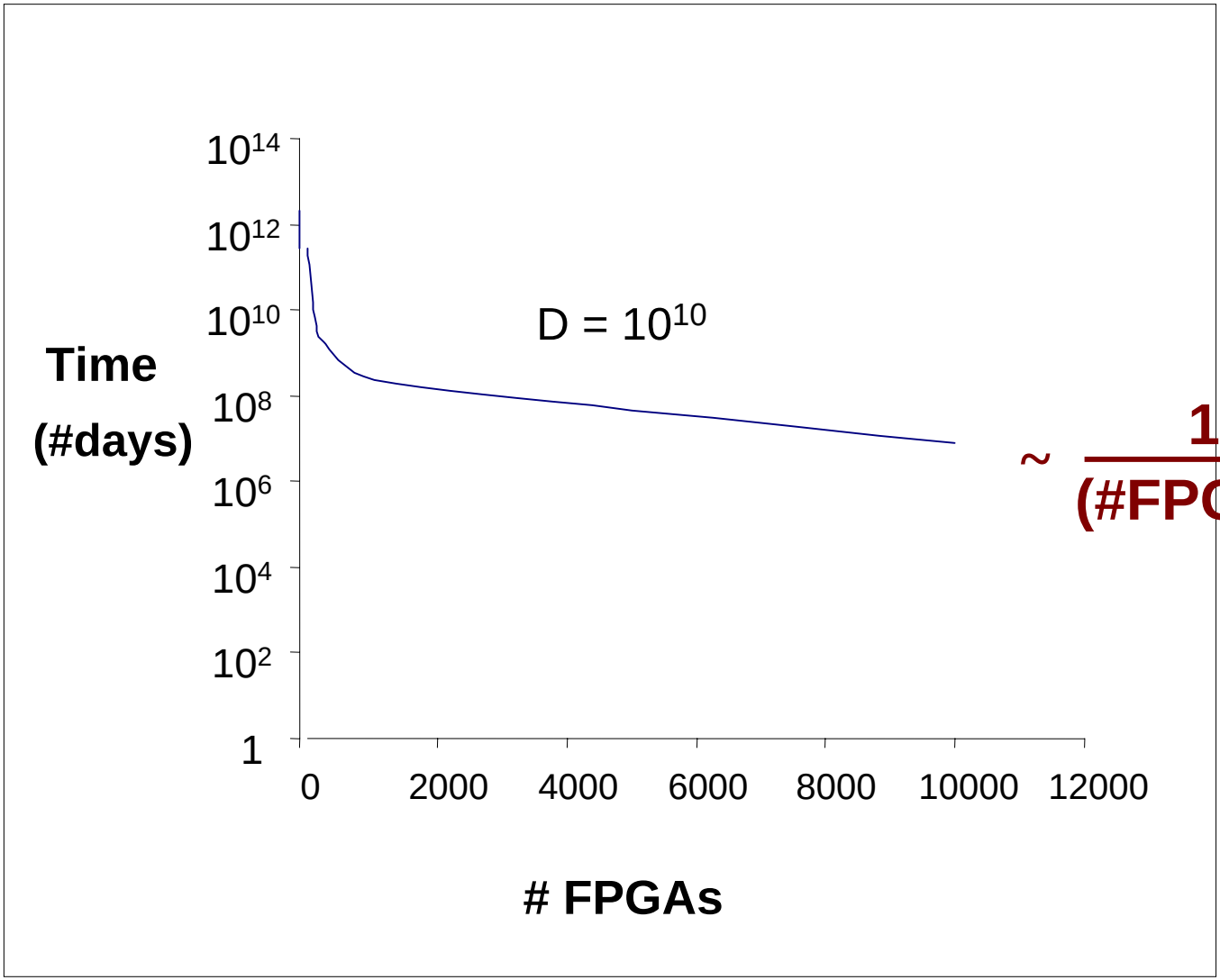
R = (#FPGAs * T_{Total})/ (1 FPGA * T_{Total} for 1 FPGA)

Virtex II chips (f)	D	p	$d^{(f)}$	$m^{(f)}$	n	T_{route} (ns)	T_{Load} (ns)	T_{Total} (days)	R
1	10^{10}	16	1	12	1.8×10^{13}	13,593	1,568	1.9×10^{12}	1.00
10^2	10^{10}	16	1	120	1.8×10^{9}	4.8×10^{5}	4.5×10^{4}	6.9×10^{9}	0.35
16^2	10^{10}	16	1	192	2.8×10^{8}	8.2×10^{5}	7.5×10^{4}	1.8×10^{9}	0.23
32^2	10^{10}	16	1	384	1.8×10^{7}	1.7×10^{6}	1.6×10^{5}	2.3×10^{8}	0.12

Time vs. D

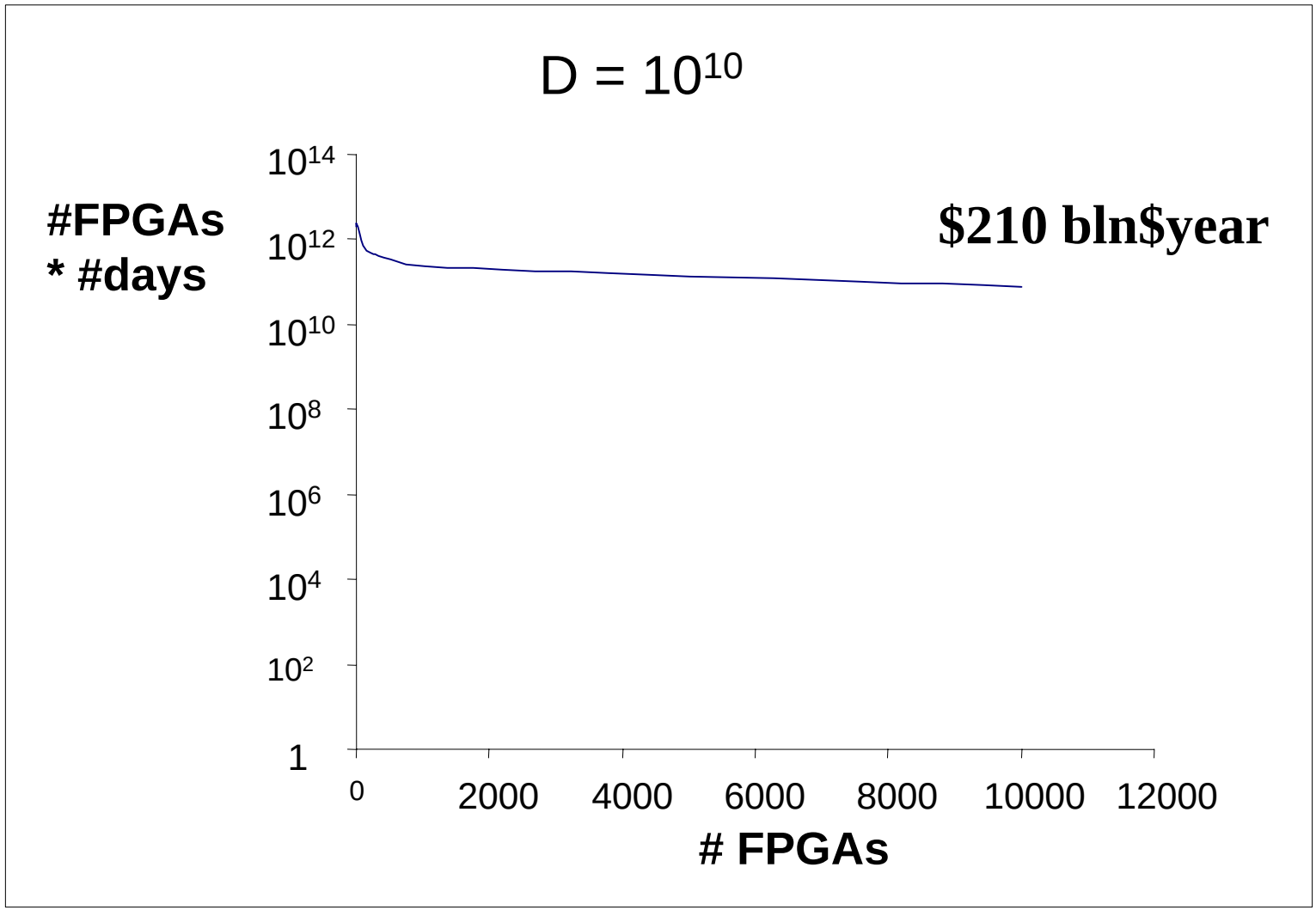

Time vs D
Time (days)
10^{20}
10^{15}
10^{10}
10^{5}
1 year
1
10^{-5}
10^{6} 10^{7} 10^{8} 10^{9} 10^{10} 10^{11} 10^{12} 10^{13}
Matrix Dimension (cols)
1 fpga
100 fpgas
256 fpgas
1024 fpgas
10000 fpgas
1000000 fpgas
21,000 years
1 fpga
100 fpgas
256 fpgas
1024 fpgas
10000 fpgas
1000000 fpgas

Time vs. #FPGAs



$$\sim \frac{1}{(\#FPGA)^{3/2}}$$

Cost*Time vs. #FPGAs



Conclusions

Summary & Conclusions (1)

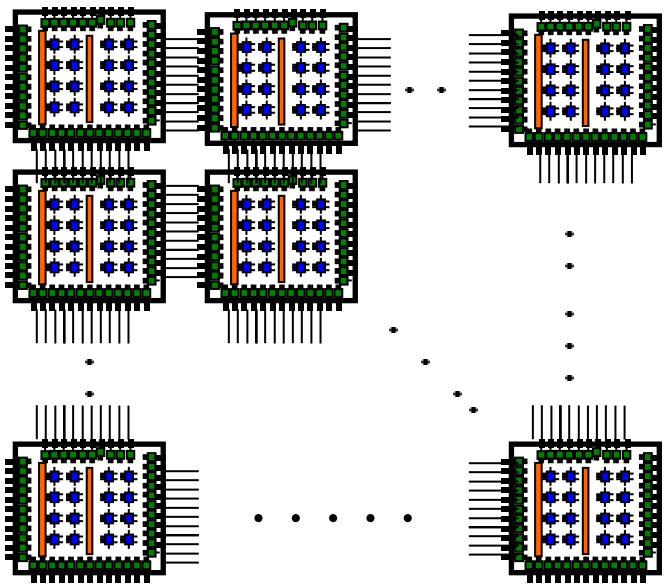
- First practical hardware implementation of Mesh Routing for the Number Field Sieve implemented and verified using timing simulation
- Practical numbers, based on post-placing & routing static timing analysis, obtained for an array of Xilinx Virtex II 8000 FPGAs
- A two-dimensional array of Virtex II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

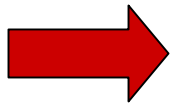
Summary & Conclusions (2)

- **Matrix step for a 512-bit RSA key takes about 5 days on the rectangular array of 100 FPGAs**

- **Assuming matrix size $D=10^{10}$ matrix step for a 1024-bit RSA key appears to be prohibitive from the point of view of full (throughput) cost**

Mapping designs to existing reconfigurable platforms

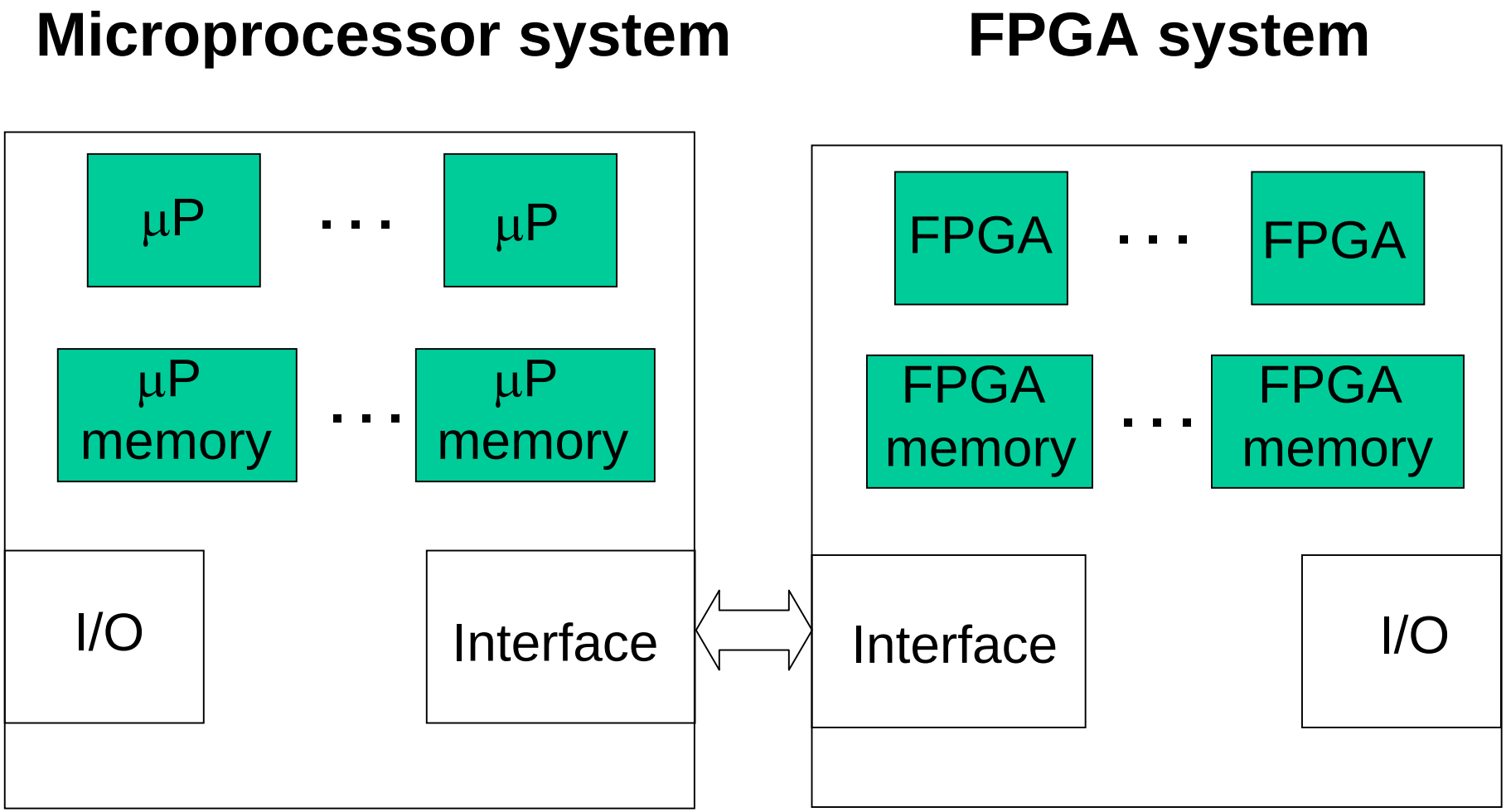




Generic Array of FPGAs

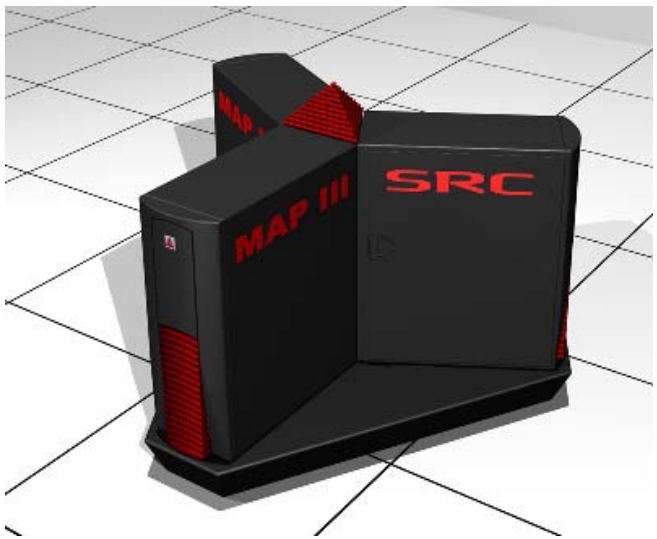
Existing General – Purpose Reconfigurable SuperComputers

What is a reconfigurable computer?



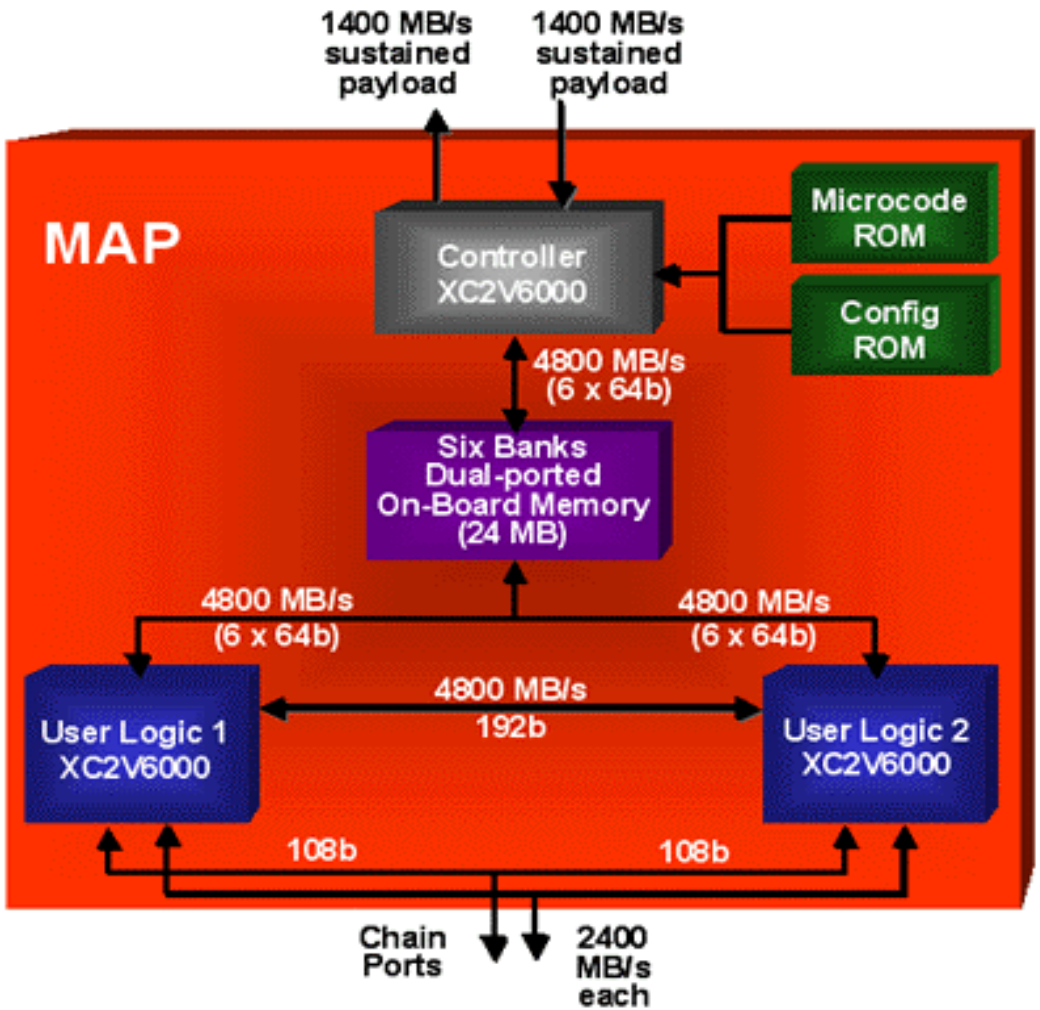
Example: SRC 6E System

http://www.srccomp.com/



Source: [SRC, MAPLD04]

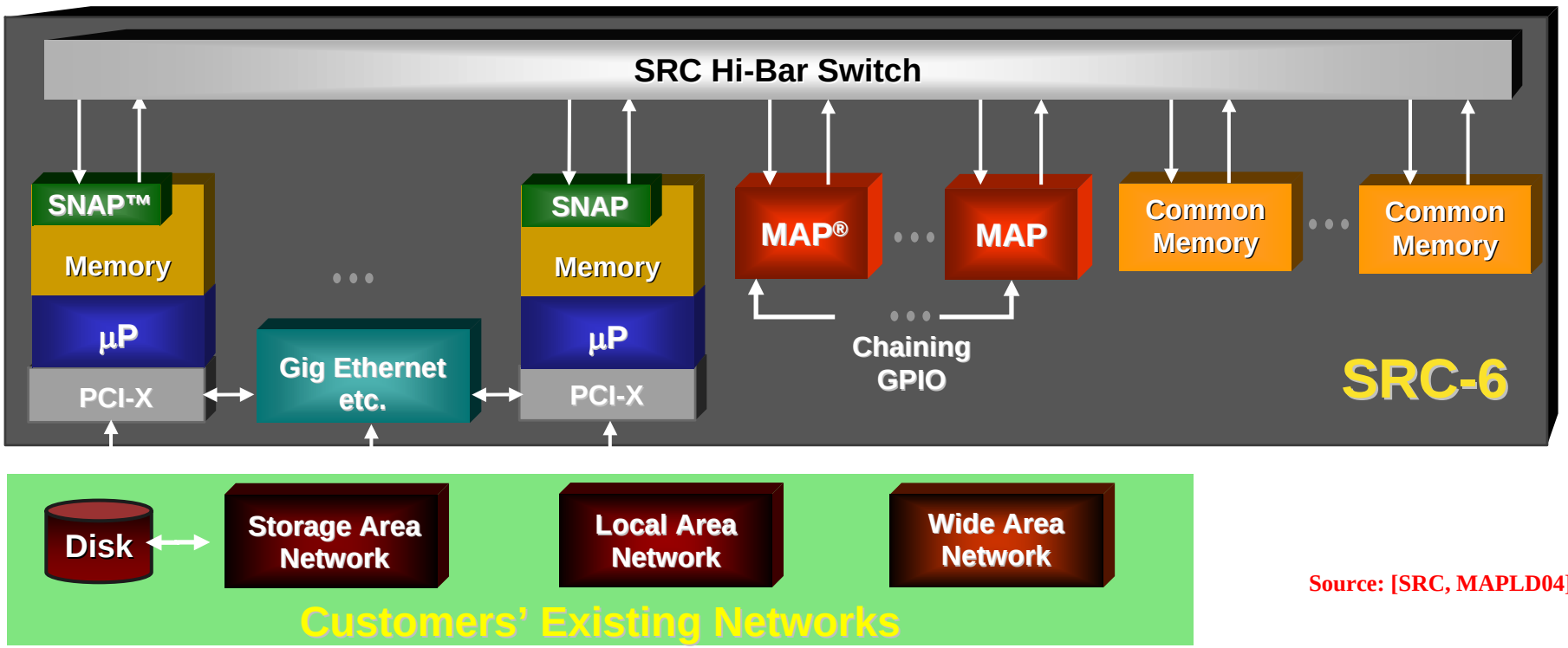
SRC MAP™ Reconfigurable Processor



Source: [SRC, MAPLD04]

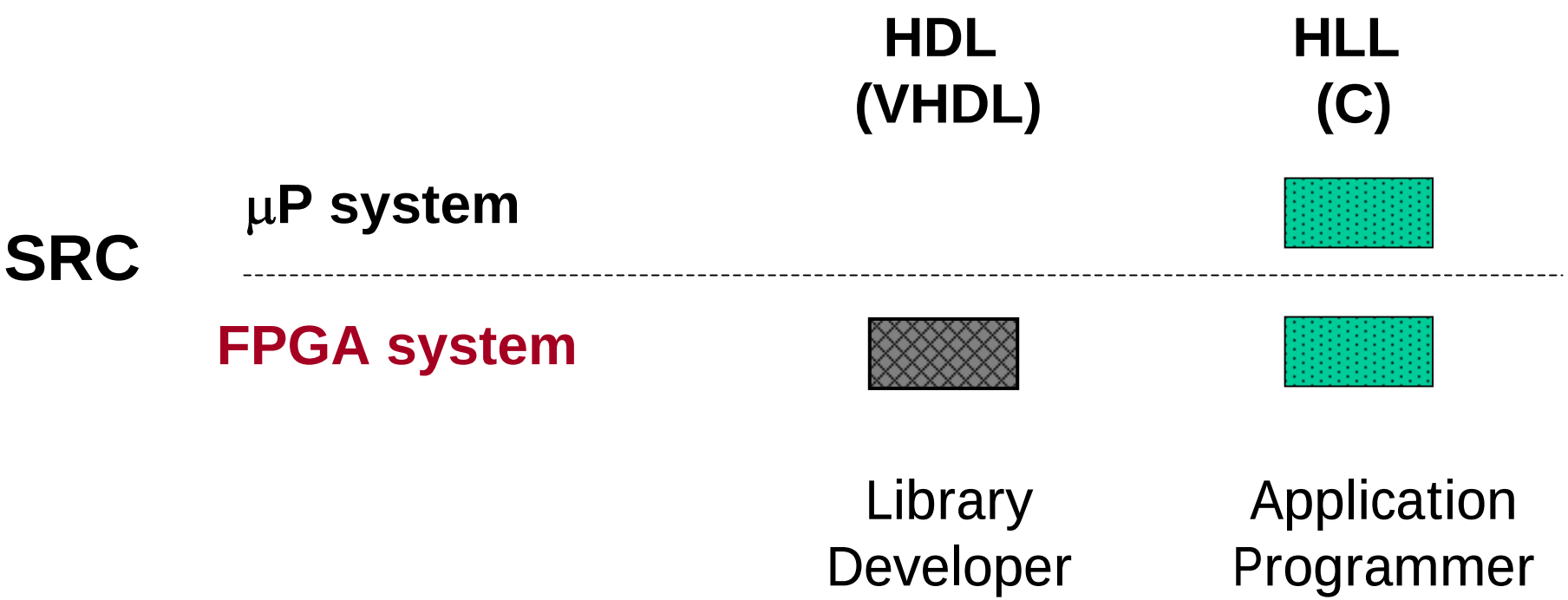
SRC Hi-Bar™ Based Systems

- Hi-Bar sustains 1.4 GB/s per port with 180 ns latency per tier
- Up to 256 input and 256 output ports with two tiers of switch
- Common Memory (CM) has controller with DMA capability
- Controller can perform other functions such as scatter/gather
- Up to 8 GB DDR SDRAM supported per CM node



Source: [SRC, MAPLD04]

SRC Programming



Other reconfigurable supercomputers

- **Cray XD1 (formerly Octiga Bay 12 K) from Cray Inc.**
- **SGI Altix 3000 from Silicon Graphics**
- **Star Bridge Hypercomputer from Star Bridge Systems**

Advantages of reconfigurable computers

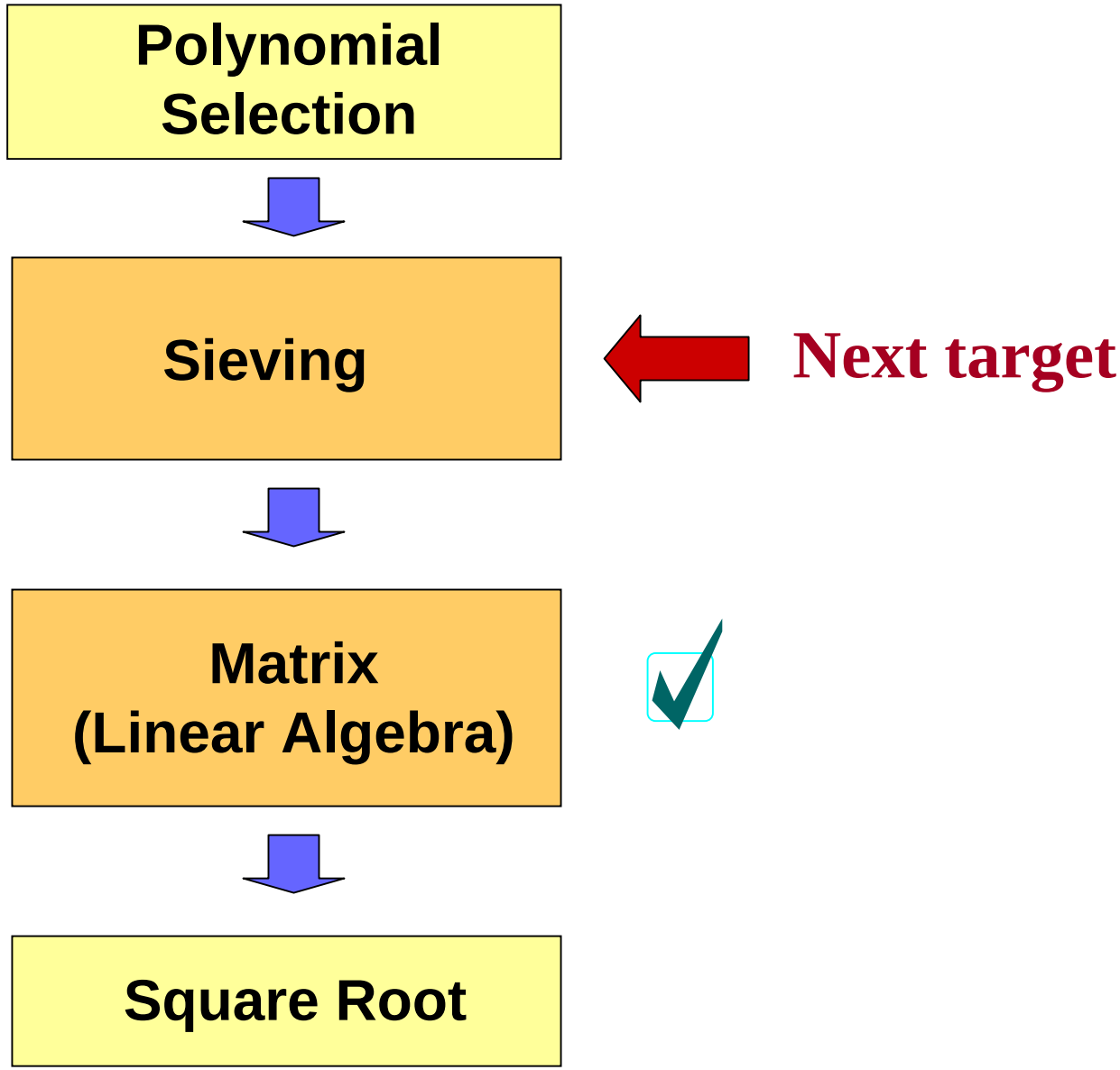
- **general-purpose: cost distributed among multiple users with different needs**

- **behave like hardware:**
  - **parallel processing**
  - **distributed memory**
  - **specialized functional units, etc.**

- **can be programmed by mathematicians themselves using traditional programming languages or GUI environments**

- **encourage innovation and experimentation**

Our future goal



Questions?

Backup slides

Sources of optimism

- emergence of new companies supporting reconfigurable supercomputing, including major players in the area of traditional supercomputing, such as Cray Inc. and SGI

- constant progress in the capabilities, performance, and flexibility of existing reconfigurable computing platforms

- constant progress in the compiler technology, and logic synthesis of high level programming languages.

Parameters

f = number of FPGAs in the FPGA array

$m^{(1)}$ = mesh dimension for one FPGA = 12

$m^{(f)}$ = mesh dimension for f FPGAs = $m^{(1)} \cdot \sqrt{f}$

p = number of columns of matrix handled in one cell of the mesh

$D^{(f)}$ = matrix dimension handled by f FPGAs $= (m^{(f)})^2 \cdot p$

$= (m^{(1)} \cdot \sqrt{f})^2 \cdot p$

$= f \cdot (m^{(1)})^2 \cdot p$

$= f \cdot D^{(1)}$

Parameters

D = number of columns in matrix A

$D^{(1)}$ = number of columns of sub-matrix of A handled by one FPGA in the mesh

$= (m^{(1)})^2 \cdot p$

$D^{(f)}$ = number of columns of sub-matrix of A handled by f FPGAs in the mesh

$= (m^{(f)})^2 \cdot p = (m^{(1)})^2 \cdot f \cdot p = D^{(1)} \cdot f$

d = column density of matrix A

$d^{(1)}$ = density of sub-matrix handled by one FPGA

$= d \cdot D^{(1)} / D$

$d^{(f)}$ = density of sub-matrix handled by f FPGAs

$= d \cdot D^{(f)} / D$

Routing Parameters

T_{CLK_mult} = multiplication clock period

T_{CLK_IO} = IO clock period

x = bits to exchange between FPGAs / (bus size between FPGAs)

$= 2 \cdot (1 + 2 \cdot k^{(f)} + k_p + K) \cdot (m^{(1)})^2) / 277$

T_{step} = Total time needed for transfer of packet between cells across FPGAs

$= 4 \cdot T_{CLK\_IO} + (x-1)\, T_{CLK\_IO} + T_{CLK\_mult}$

h_c = slowdown factor due to limited inter-FPGA IO connections

$= T_{step} / T_{CLK\_mult}$

T_{routne} = routing time for sub-multiplication in the mesh

$= \#\text{entries per cell} \cdot \#\text{steps} \cdot T_{CLK\_mult} \cdot h_c$

$= p \cdot d^{(f)} \cdot 4 \cdot m^{(f)} \cdot T_{CLK\_mult} \cdot h_c$

Loading Unloading Parameters

$b^{(1)}$ = #pins for data transfer for 1 FPGAs = #maximum FPGA IO/ 2

$b^{(f)}$ = #pins for data transfer for f FPGAs

= (#maximum FPGA IO /4) $\cdot \sqrt{f}$

s_{IO} = clock stages between two FPGA connections

T_{CLK_load} = loading clock period

T_{load} = time for loading and unloading for a sub-multiplication

= [((#matrix entries bits) + (#vector bits to load) +

(#vector bits to unload))/ $b^{(f)}$ +

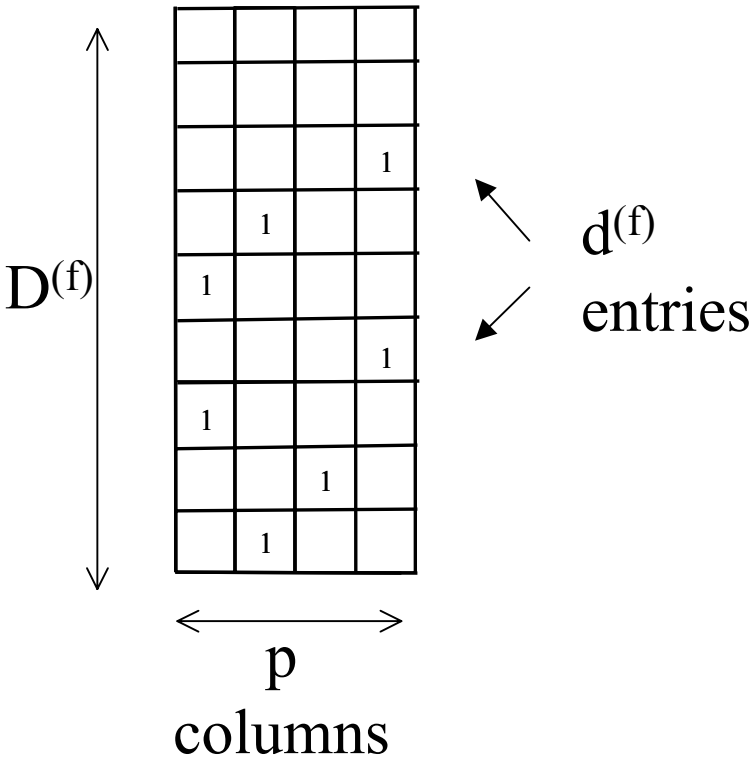
$s_{IO} \cdot (m^{(f)}-1)$ $m^{(f)}$* loading packet bits/$b^{(f)}$] $\cdot T_{CLK\_load}$ =

$$\left(\frac{((1+4\cdot \boldsymbol{k}^{(f)}+2\cdot \boldsymbol{k_p})\cdot \boldsymbol{d}\cdot \boldsymbol{D}^{(f)})+(\boldsymbol{K}\cdot \boldsymbol{D}^{(f)})+\boldsymbol{K}\cdot \boldsymbol{D}^{(f)}\cdot \boldsymbol{D}^{(f)}/\boldsymbol{D}}{\boldsymbol{b}^{(f)}} + s_{IO}\cdot (m^{(f)}-1)m^{(f)}\cdot (1+4\cdot \boldsymbol{k}^{(f)}+2\cdot \boldsymbol{k_p}+\boldsymbol{K})/b^{(f)} \right) \cdot T_{CLK\_load}$$

Parameters

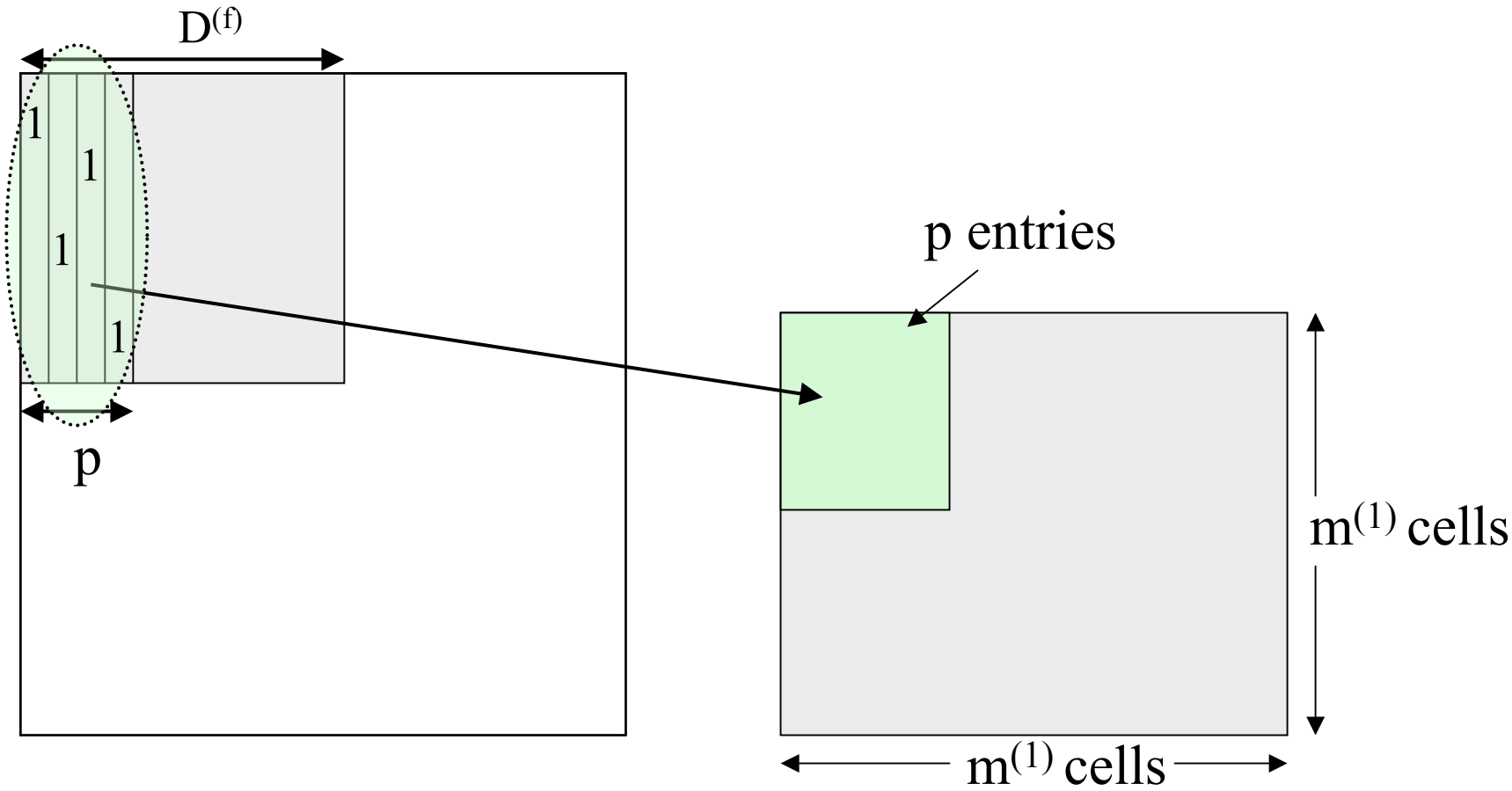
n = number of times to repeat sub-multiplications
$= D^2/(D^{(f)})^2 = D^2/((m^{(f)})^2\, p)^2$

T_{Total} = total time for a Matrix step = $3 \cdot D/K \cdot n \cdot (T_{route} + T_{load})$



total of $d^{(f)} \cdot p$ entries

$$d^{(f)} \cdot p = \lceil d \cdot p \cdot f \cdot (m^{(1)})^2 / D \rceil \cdot p$$
$$\leq \text{ threshold area on FPGA}$$


D(f)
1
1
1
1
p
p entries
m(1) cells
m(1) cells

Total time of routing

- **Total routing takes maximum d· 4·m· 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