A systolic architecture for supporting Wiedemann's algorithm

Rainer Steinwandt[1]

(joint work with Willi Geiselmann[1],

Adi Shamir[2] and Eran Tromer[2])

[1] Universität Karlsruhe, Germany

[2] Weizmann Institute of Science, Israel

NFS & (block) Wiedemann

NFS: relation collection + linear algebra (LA) step

dominating for total running time

(Block) Wiedemann for GF(2):

reduces LA step to iterated matrix-by-vector multiplications

$$Av,\ A^2v,\ A^3v,\ \ldots,\ A^kv$$

with **sparse** (... **but potentially large**) matrix A

1024 bit: $A \in GF(2)^{10^{10} \times 10^{10}}$

... doing this fast could be nice for GF(p), too (→[Frey04])

LA hardware: basic approach

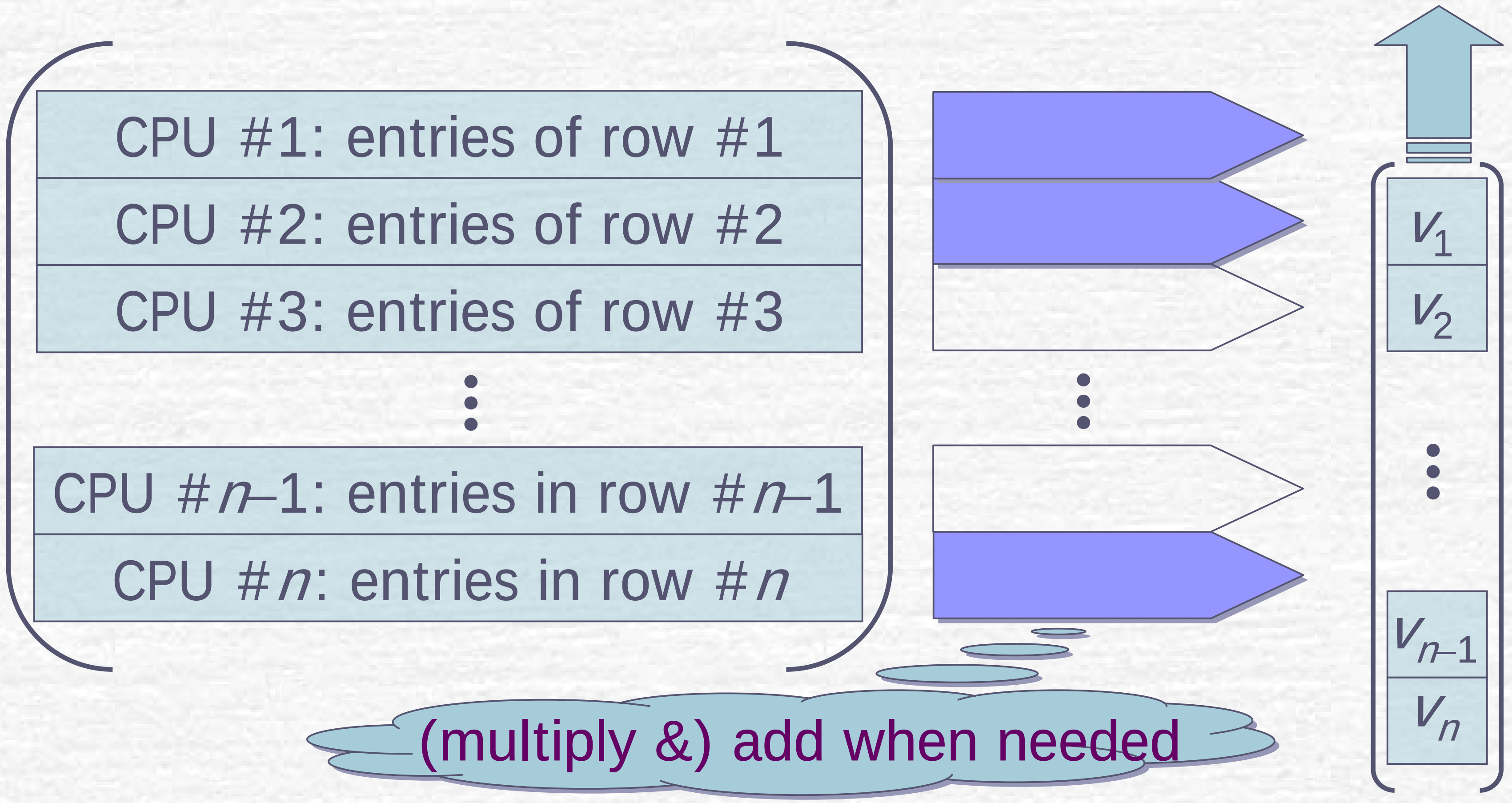
Currently most promising hardware devices for LA step:

- offer methods for efficiently computing the vector chains $Av, A^2v, A^3v, \ldots, A^kv$ using a **2-D mesh architecture**:
    - 2-D **sorting** ($\rightarrow$[Bernstein '01])
    - 2-D **routing** ($\rightarrow$[Lenstra et al. '02])
- impose **another 2-D splitting for** doing with **small chips** ($\rightarrow$[Geiselmann, S. '03])

..., cheap

... not utopian, but not as simple & efficient as desirable

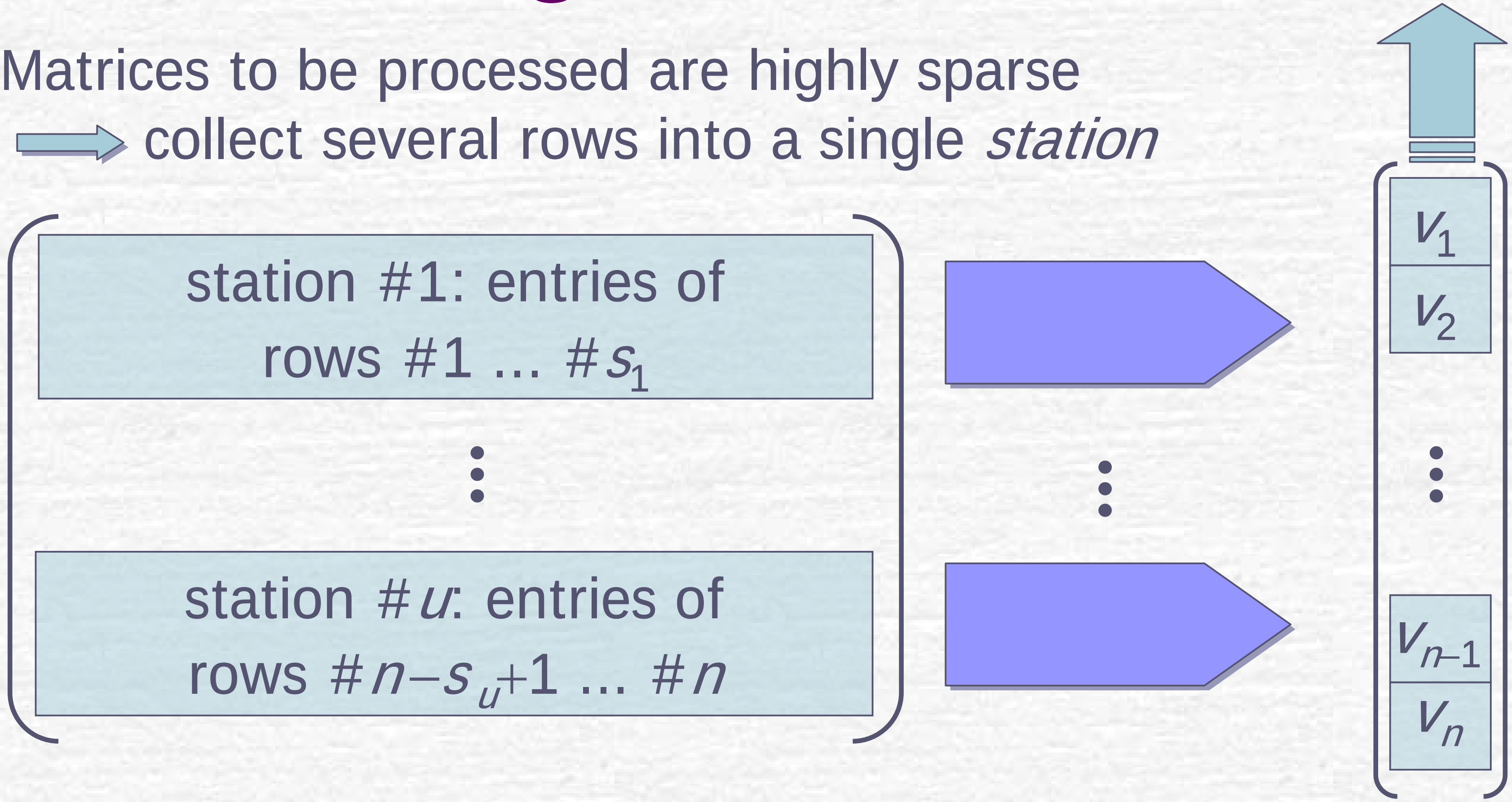
Multiplying with $v \in \mathrm{GF}(p)^n$


CPU #1: entries of row #1
CPU #2: entries of row #2
CPU #3: entries of row #3
CPU #n–1: entries in row #n–1
CPU #n: entries in row #n
v_1
v_2
v_{n-1}
v_n
(multiply &) add when needed

Collecting rows in stations

Matrices to be processed are highly sparse

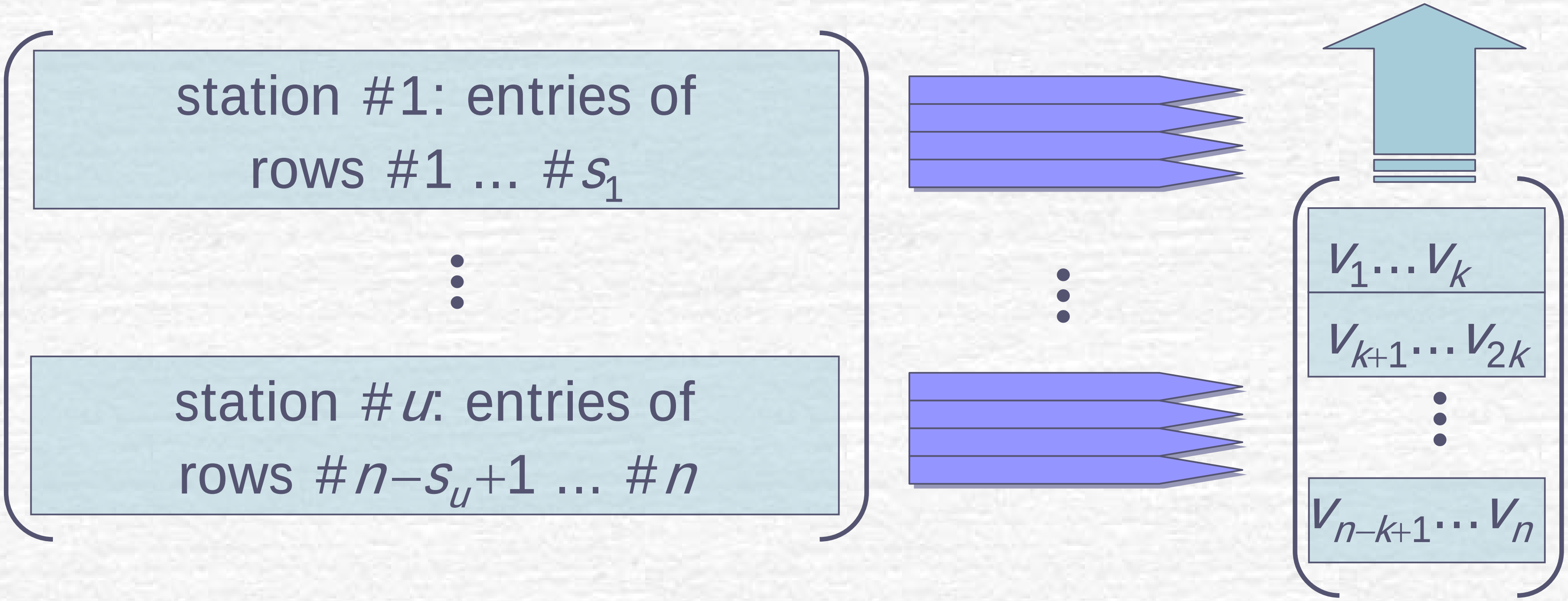
$\Rightarrow$ collect several rows into a single *station*


station #1: entries of rows #1 ... #s_1
⋮
station #u: entries of rows #$n-s_u+1$ ... #n
⋮
v_1
v_2
⋮
v_{n-1}
v_n

Additional parallelization

Needed arithmetics is not space-consuming

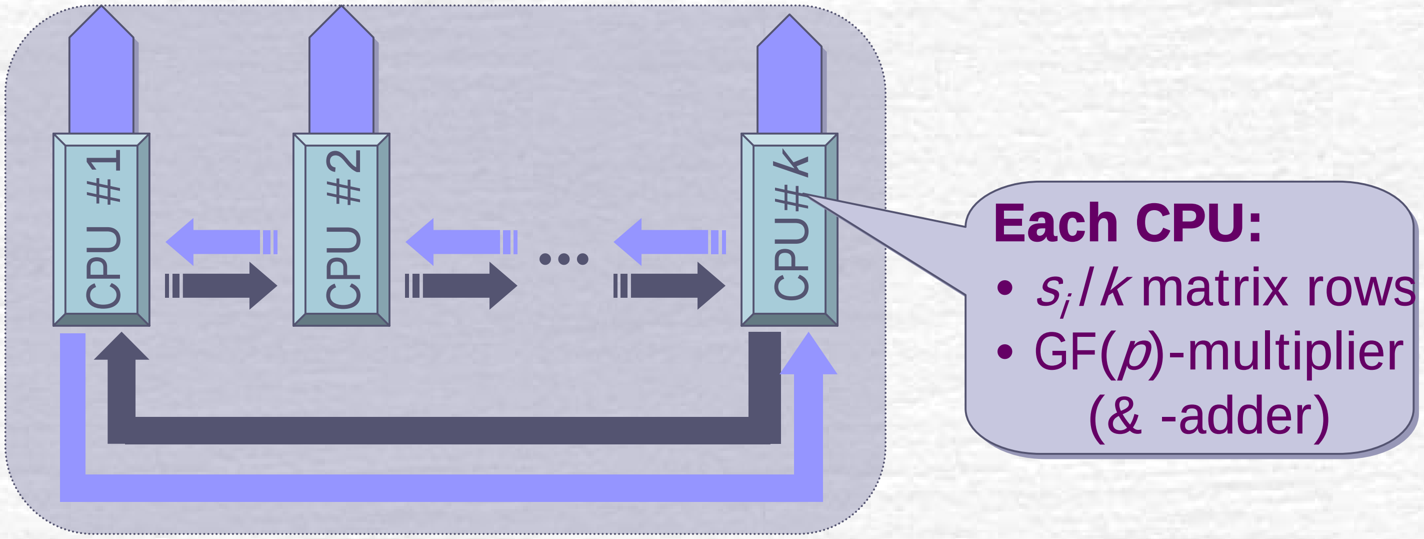
$\Rightarrow$ process $k>1$ vector components in parallel



station #1: entries of rows #1 ... #s_1
station #u: entries of rows #$n-s_u+1$ ... #n
$v_1 \ldots v_k$
$v_{k+1} \ldots v_{2k}$
$v_{n-k+1} \ldots v_n$

… using intra-station buses

Handling *k* vector components in parallel in each station:



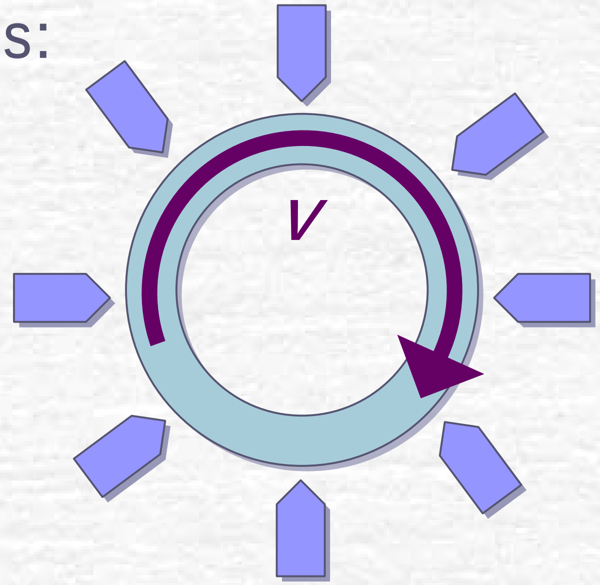
Circular buses for intra-station transport of *v*-entries.

Multiplying with *A* again

Actually needed: $A \cdot v$, $A \cdot Av$, $A \cdot A^2 v$, …

⇨ result of multiplication must go back into vector pipeline

⇨ rearrange stations:



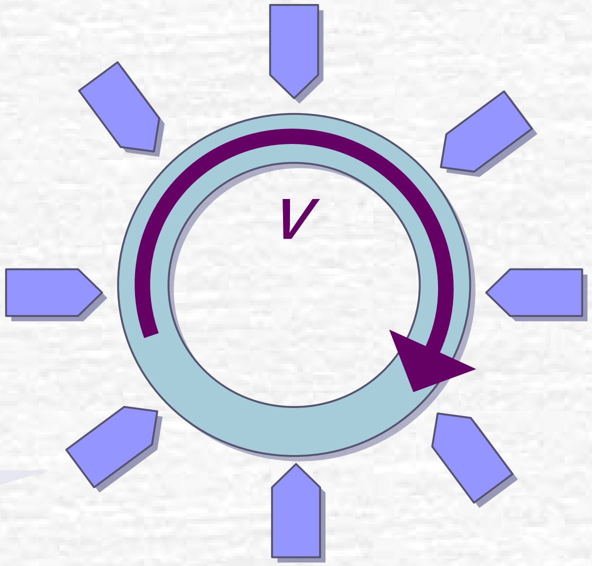
… have each station scan *v* in a different cyclic order

Doing another multiplication

GF(*p*)-addition commutative

1 complete cycle yields $A \cdot v$

stations switch to 2nd mem. bank holding $A \cdot v$



Device is immmediately prepared for next multiplication.

Critical parameters

- **I/O Bandwith, number of pins:**
  limits the speed at which v can be fed into the stations & therewith overall LA time
- **Memory:**
  representing the non-zero entries of A &
  storing the vector(s) v requires large amount of (D)RAM
- **Clock rate:**
  simple logic allowing high clocking rate vs.
  (slow) space-optimized memory

Techn(olog)ical limitations

- #pins limited through chip size (>2^{12} pins means large chips)
- logic for systolic design simpler than for mesh-based designs

$\Rightarrow$ increasing clocking rate to 1 GHz seems doable

What about the memory?

vector *v*: dense, 2×(D)RAM for m (=10^{10}) GF(p)-entries

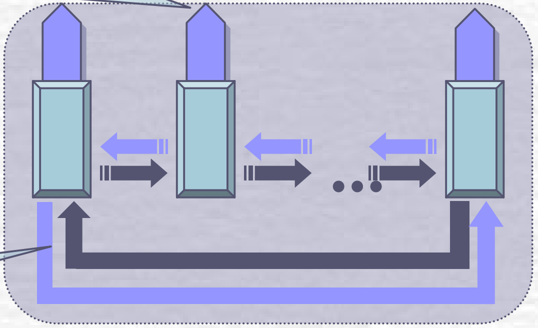
matrix *A*: GF(p)-entry, row coord. within CPU, auxiliary flags

no need for random access, DRAM sufficient

Matrix handling

"External table" for reading *v*-entries:

#wait cycles	"read it" flag	bus no. to write on



"Internal table" for storing the matrix:

#wait cycles	"read it" flag	bus no. to read from
GF(p)$^\times$-entry	row coord.	"delete it" flag

Distributing the matrix

As with mesh based designs, we can **split *A* into submatrices** ($\rightarrow$[Geiselmann, S. '03]):

$$A=\left(\begin{array}{c|c|c|c} A_{1,1} & A_{1,2} & \cdots & A_{1,r} \\ \hline A_{2,1} & A_{2,2} & \cdots & A_{2,r} \\ \hline \multicolumn{4}{c}{\vdots} \\ \hline A_{r,1} & A_{r,2} & \cdots & A_{r,r} \end{array}\right),\quad v=\left(\begin{array}{c} v_{1,1} \\ \hline \vdots \\ \hline v_{1,s} \end{array}\right) \cdots = \left(\begin{array}{c} v_{s,1} \\ \hline \vdots \\ \hline v_{r,r} \end{array}\right),\quad A\cdot v=\left(\begin{array}{c} \sum A_{1,j}\cdot v_{1,j} \\ \hline \vdots \\ \hline \sum A_{r,j}\cdot v_{r,j} \end{array}\right)$$

store submatrix coordinates only

Block matrix multiplication

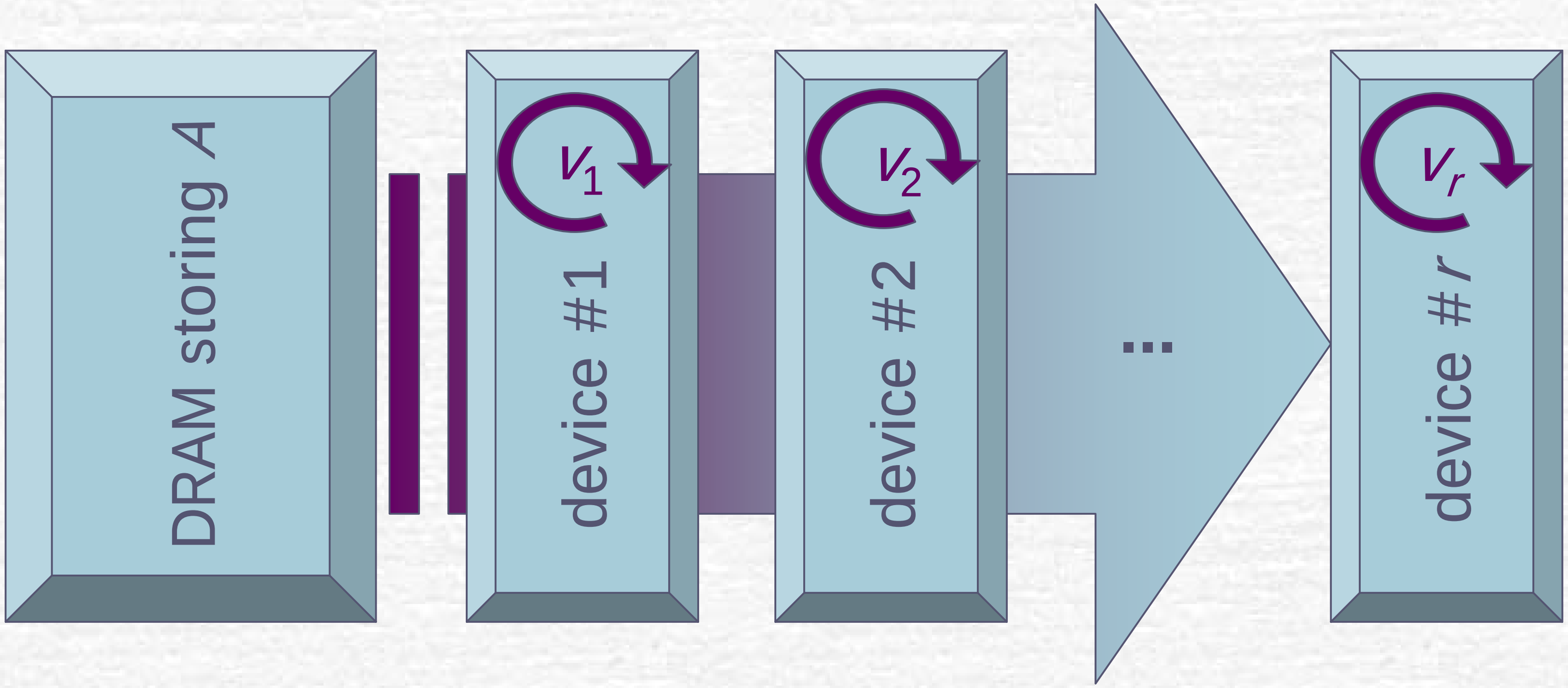
- assign a **multiplication circuit** to **each submatrix** $A_{i,j}$
- distribute/**load** appropriate ***v*-parts** into each circuit
- compute **all** $\boldsymbol{A_{i,j} \cdot v_{i,j}}$ **–values**
- **output** all **subproducts & add them** in a pipeline

result must be split &
loaded into the device

Limiting factor for run time: I/O bandwidth/#pins

Systolic parallelization

Increased blocking factor without repeatedly storing *A*:



... combining it all

splitting of *A* into submatrices can be **combined with systolic parallelization**

↓

short vectors + small matrices + simple logic

↓

small interconnected chips

↓

... may be fast, but not that trivial to implement

↓

2D-systolic looks preferable

Systolic vs. mesh based design

Features of (pure) systolic approach:

- **simple logic**
- use of **small chip sizes** seems doable
- **integration of error handling** looks doable
- **no** need for **heuristic complexity** bounds
- **simulation** in software **is possible**

... how fast can we go?

- using **similar chip area** as currently fastest mesh-based proposals (→[Geiselmann et al. '05]), a **significant speed-up**, **say factor 2**, seems realistic
- various possibilites for optimization: area × time, cost, ...

Simpler and more efficient than existing proposals:

LA step for 1024 bit looks (even more 🙂) doable.

Conclusion

- systolic design looks **preferable to mesh**-based approach: seems to be simpler, faster and require smaller chips
- topic of **"optimal" parameter choice** (purely systolic, matrix splitting, ...) not fully explored yet

... coping with LA step for 1024 bit is not utopian 🙂