Improving GPU's energy efficiency by using data regularity

Fabrice Mouhartem

With the team ALF

Outline

Outline

Presentation

What is a GPU ?

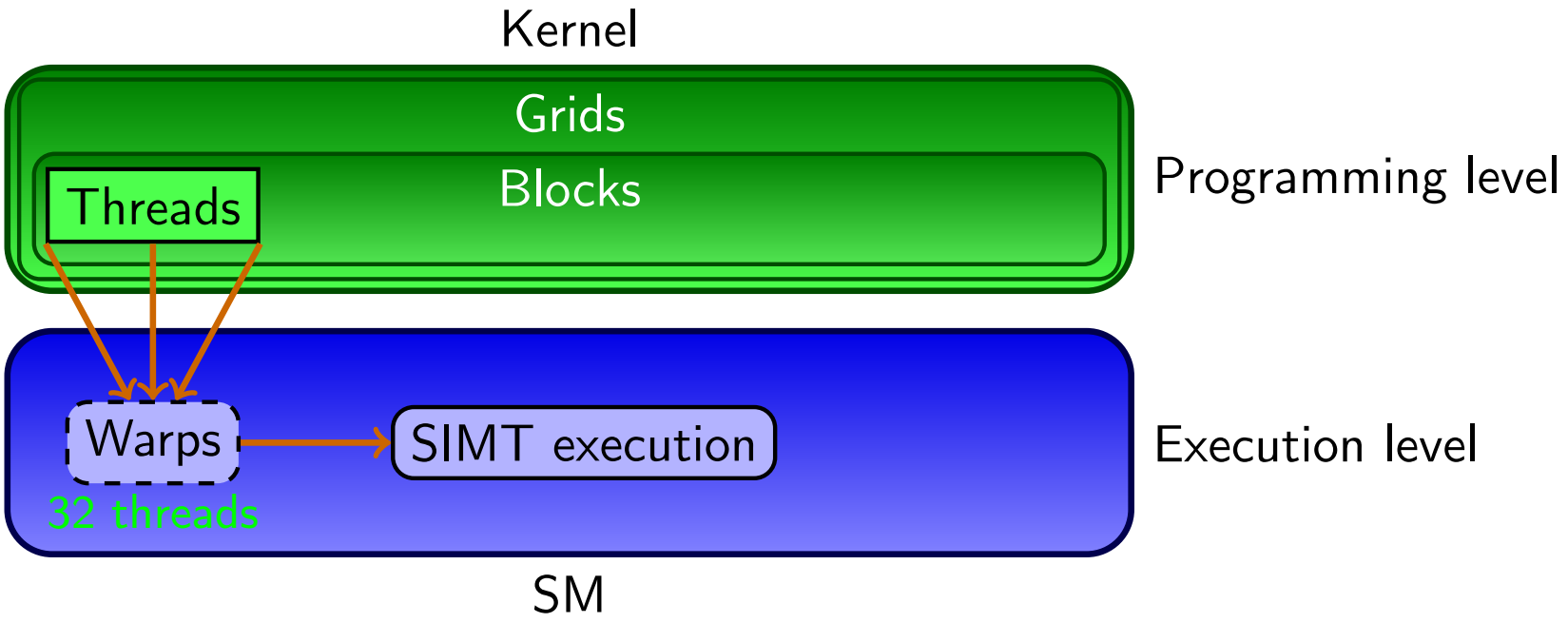
A GPU is a unit specialized in graphical operations which is nowadays used for general purpose computing.

Presentation

What is a GPU ?

A GPU is a unit specialized in graphical operations which is nowadays used for general purpose computing.

GPU programming overview.



The problem

Observation

GPU performance is limited by the energy consumption of the chip.

The problem

Observation

GPU performance is limited by the energy consumption of the chip.

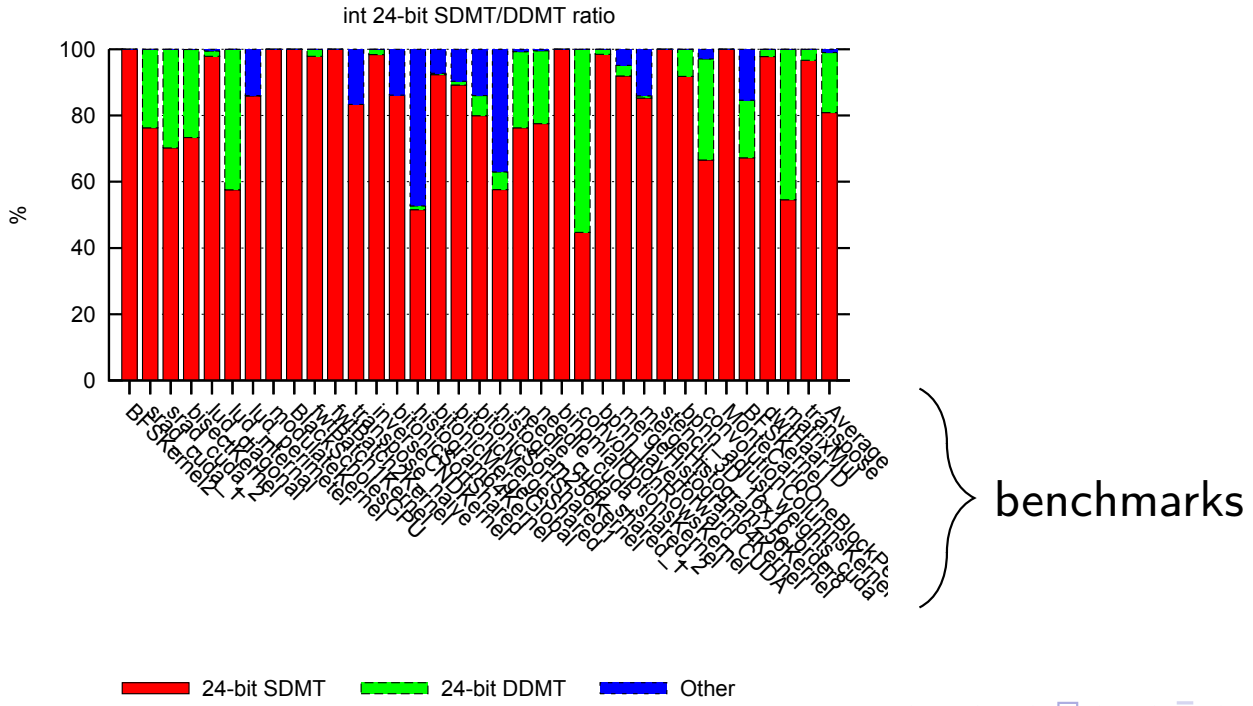
Goal

Improve energy efficiency : lower power for same performances or higher performances.

Observations

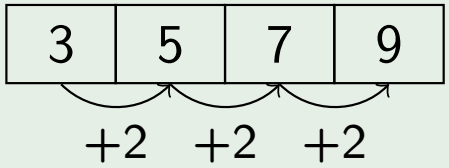
Registers are very similar

SDMT	12	x	12	y	12	z	12	w	81% of integer registers.
	24 bits	8 bits							
DDMT	12	x	10	y	12	z	12	w	97% of integer registers.

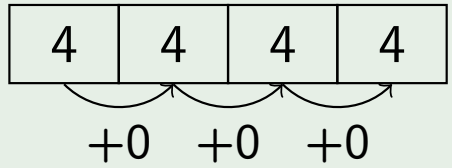


Observations

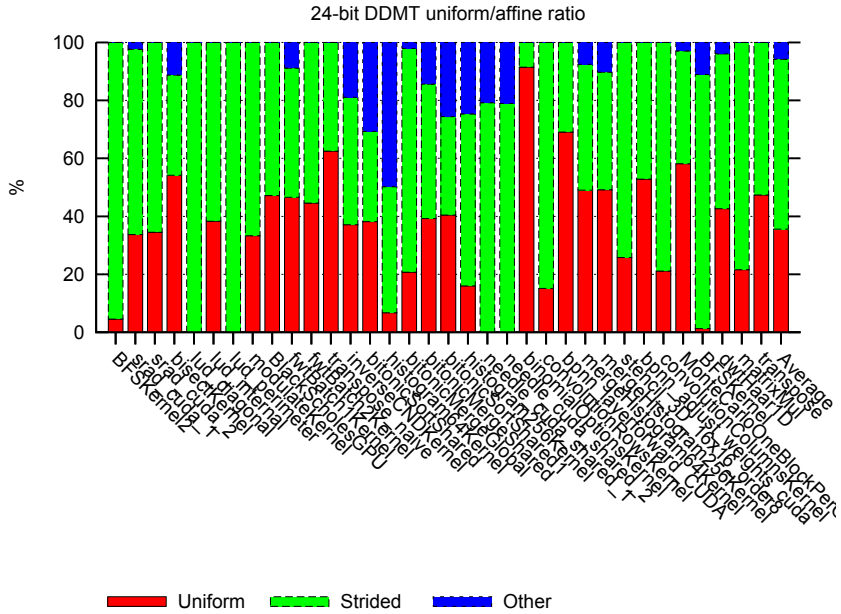
And the similar registers are mostly affine.



96% of DDMT registers are affine.



35% of DDMT registers are uniform.



Ideas

Two approaches :

- Compiler level
- Hardware level

Ideas

Two approaches :

- Compiler level
- Hardware level

Ideas

Two approaches :

- Compiler level
- Hardware level

Two solutions :

- Similar registers optimization.
- Affine vector optimization.

Ideas

Two approaches :

- Compiler level
- Hardware level

Two solutions :

- Similar registers optimization.
- Affine vector optimization.

Outline

Definitions

Affine vector

An affine vector is a vector register such that :

$$v[i] = b + s * i$$

Definitions

Affine vector

An affine vector is a vector register such that :

$$v[i] = b + s * i$$

Property

An affine vector is totally defined by its base and its stride.

Definitions

Affine vector

An affine vector is a vector register such that :

$$v[i] = b + s * i$$

Property

An affine vector is totally defined by its base and its stride.

Example

b = 7, s = 4

7	11	15	19	23	27	31	35

State of the art

Remark

It is possible to predict the nature of the result of an affine operation from nature of the operands.

State of the art

Remark

It is possible to predict the nature of the result of an affine operation from nature of the operands.

$+$	U	A	G
U	U	A	G
A	A	A	G
G	G	G	G

$\times$	U	A	G
U	U	A	G
A	A	G	G
G	G	G	G

$<<$	U	A	G
U	U	A	G
A	G	G	G
G	G	G	G

$$(b + s \cdot i) + (b' + s' \cdot i) = (b + b') + (s + s') \cdot i$$

$$(b + s \cdot i) \times (b' + 0 \cdot i) = (b \cdot b') + (s \cdot b') \cdot i$$

$$(b + s \cdot i) \times (b' + s' \cdot i) = b \cdot b' + (s \cdot b' + s' \cdot b) \cdot i + s \cdot s' \cdot i^2$$

State of the art

The situation

We can add an affine tag to vectors [CDZ10].

VRF								b,s		tag
1	5	2	7	3	9	2	9	0	0	0
1	2	8	3	4	5	7	6	0	0	0
0	0	0	0	0	0	0	0	5	7	1
1	1	1	0	3	8	4	8	0	0	0

State of the art

This method does not handle divergences on vector operations.

State of the art

This method does not handle divergences on vector operations.

Divergence

A divergence occurs when an operator does not apply on the entire vector.

State of the art

This method does not handle divergences on vector operations.

Divergence

A divergence occurs when an operator does not apply on the entire vector.

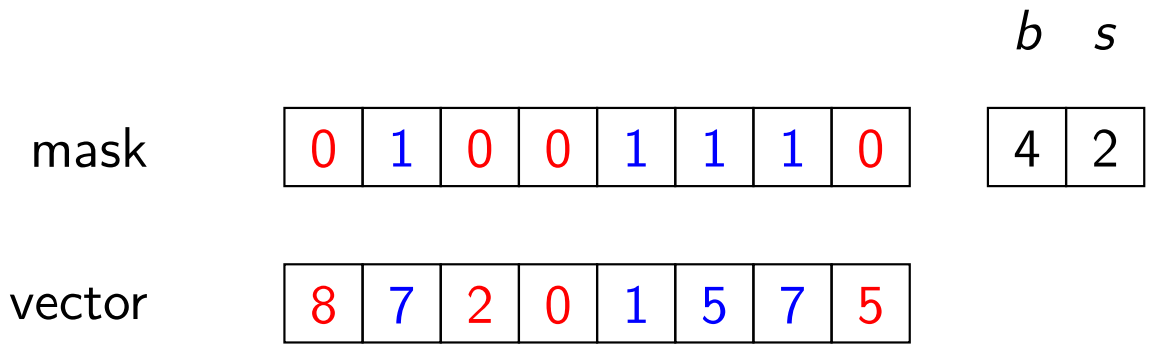
Example of divergence

mask = | 1 | 1 | 1 | 0 |

add | 3 | 5 | 7 | 9 | , | 5 | 4 | 3 | 2 | → | 8 | 9 | 10 | 9 |

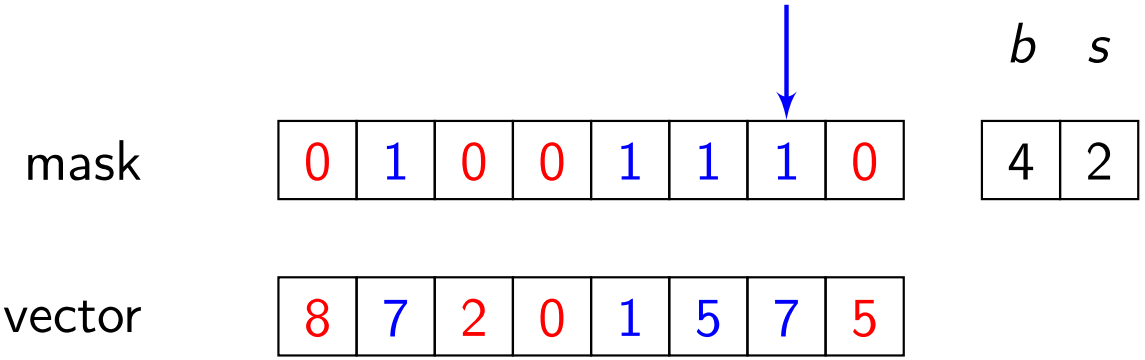
Our proposition

- We will lower the granularity of the detection.



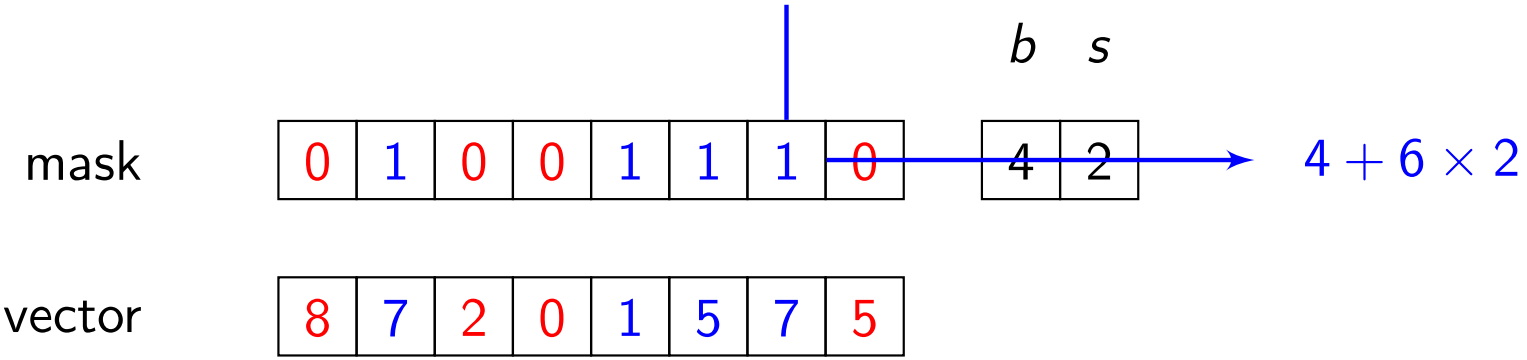
Our proposition

- We will lower the granularity of the detection.



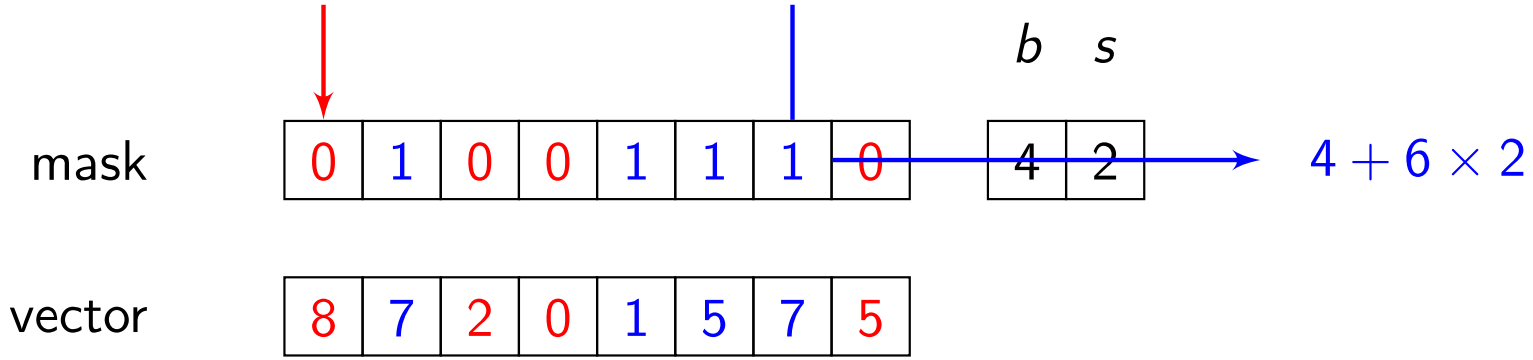
Our proposition

- We will lower the granularity of the detection.



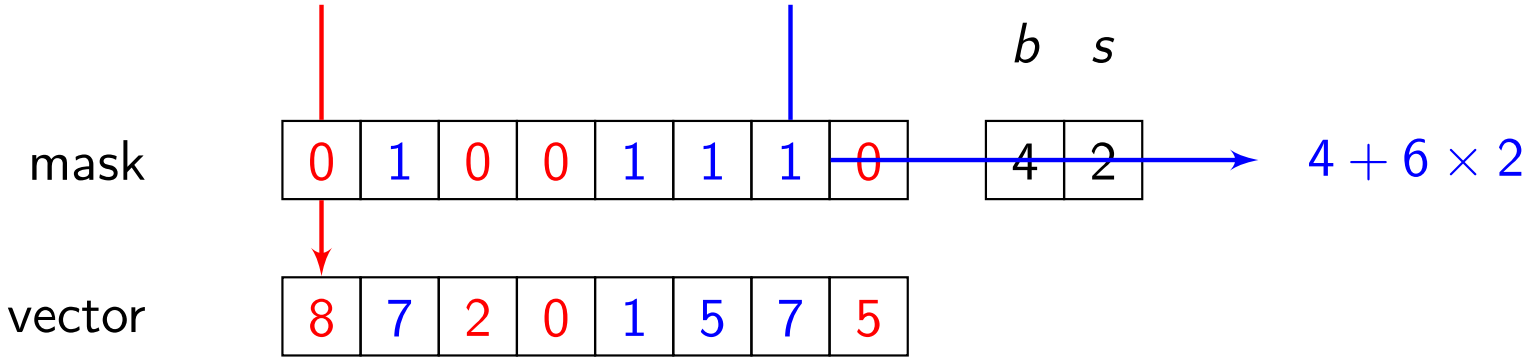
Our proposition

- We will lower the granularity of the detection.



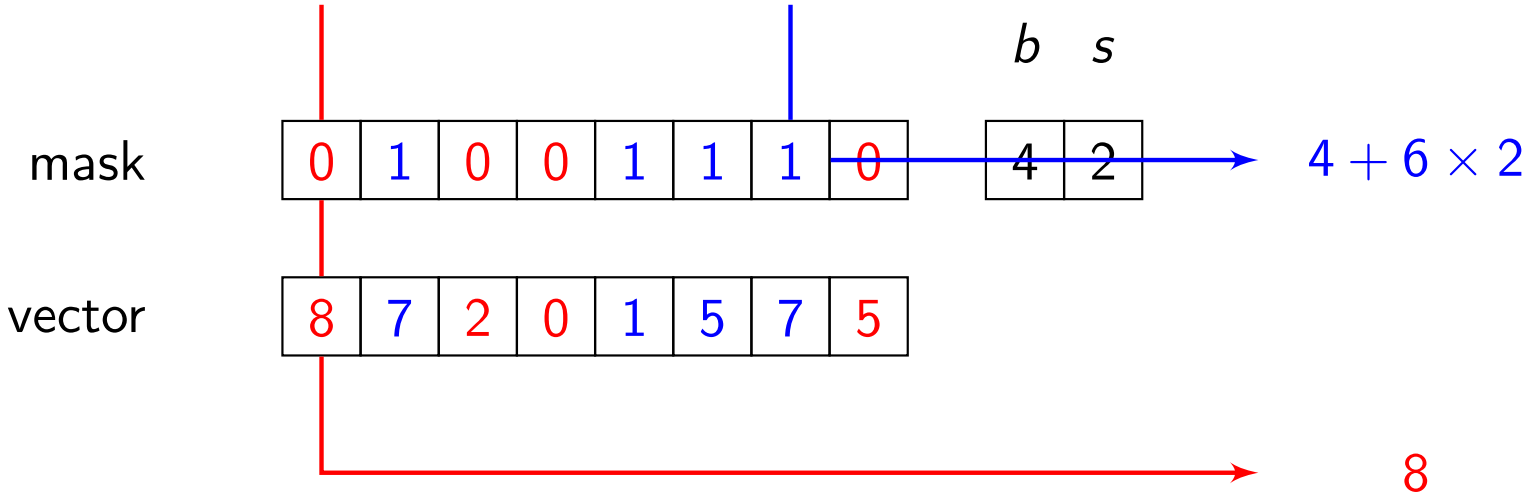
Our proposition

- We will lower the granularity of the detection.



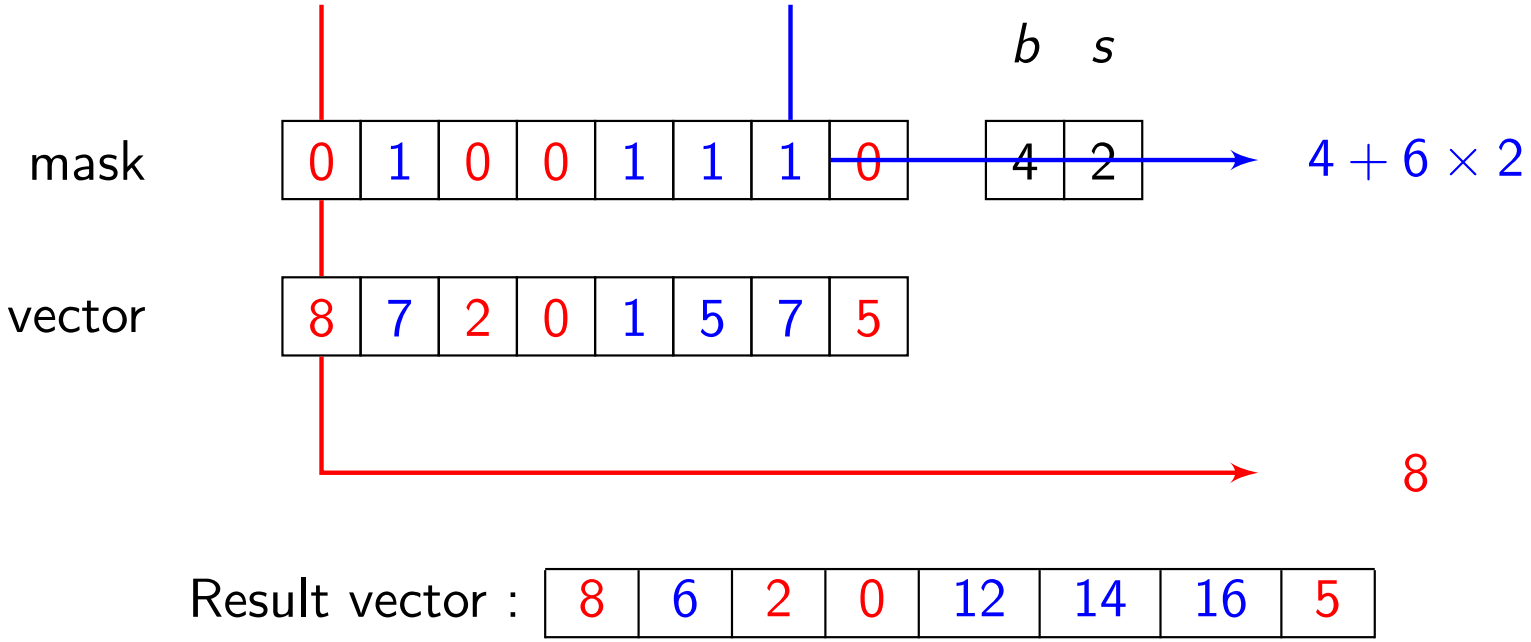
Our proposition

- We will lower the granularity of the detection.



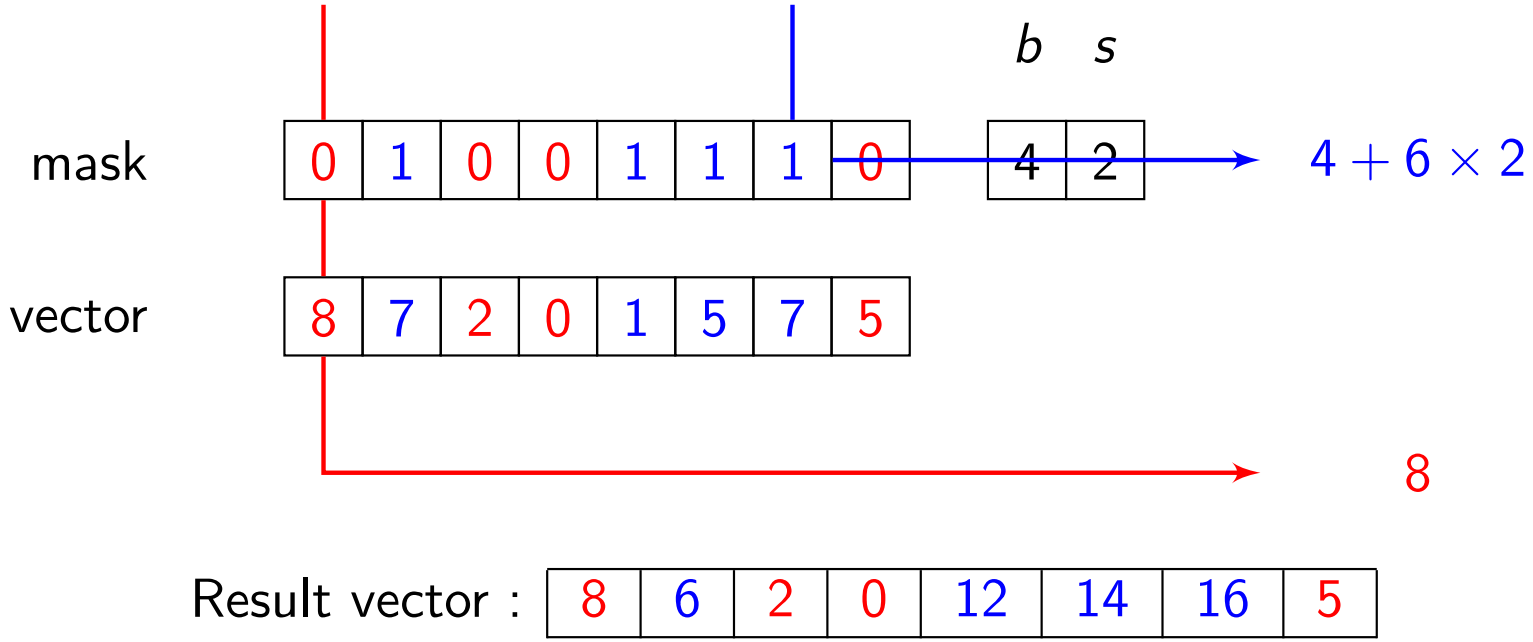
Our proposition

- We will lower the granularity of the detection.



Our proposition

- We will lower the granularity of the detection.

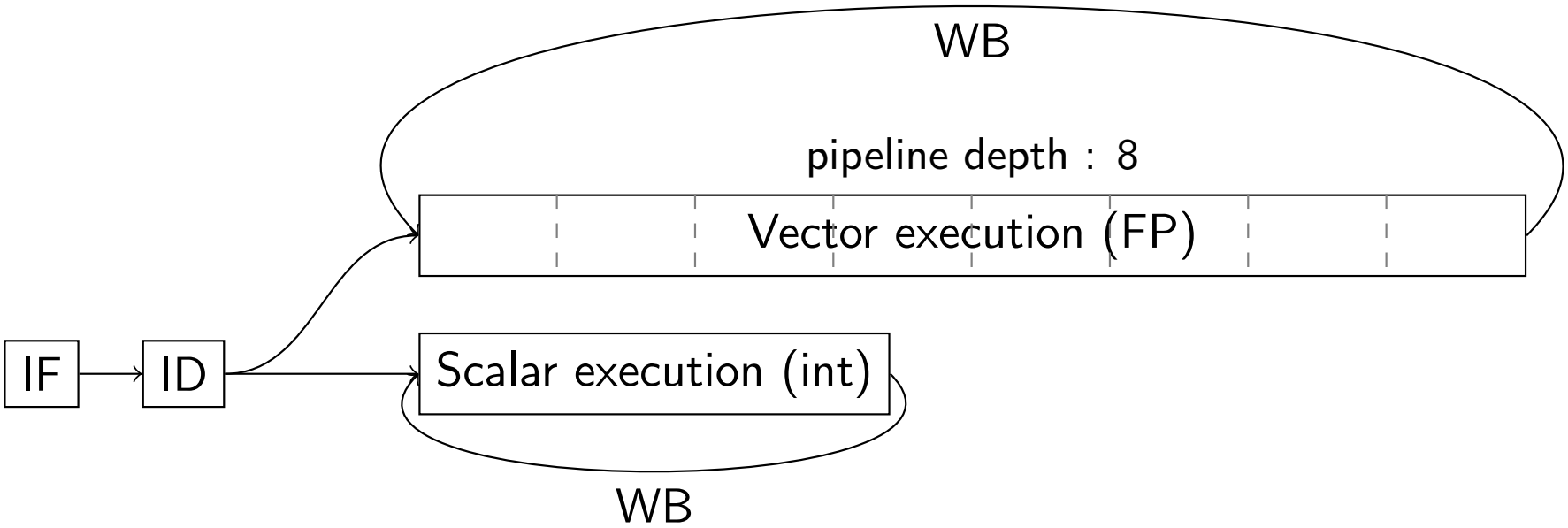


Implementation cost

32 + 32 + 8 bit per vector (initially $32 \cdot 32 = 1024$ bits) = 7%

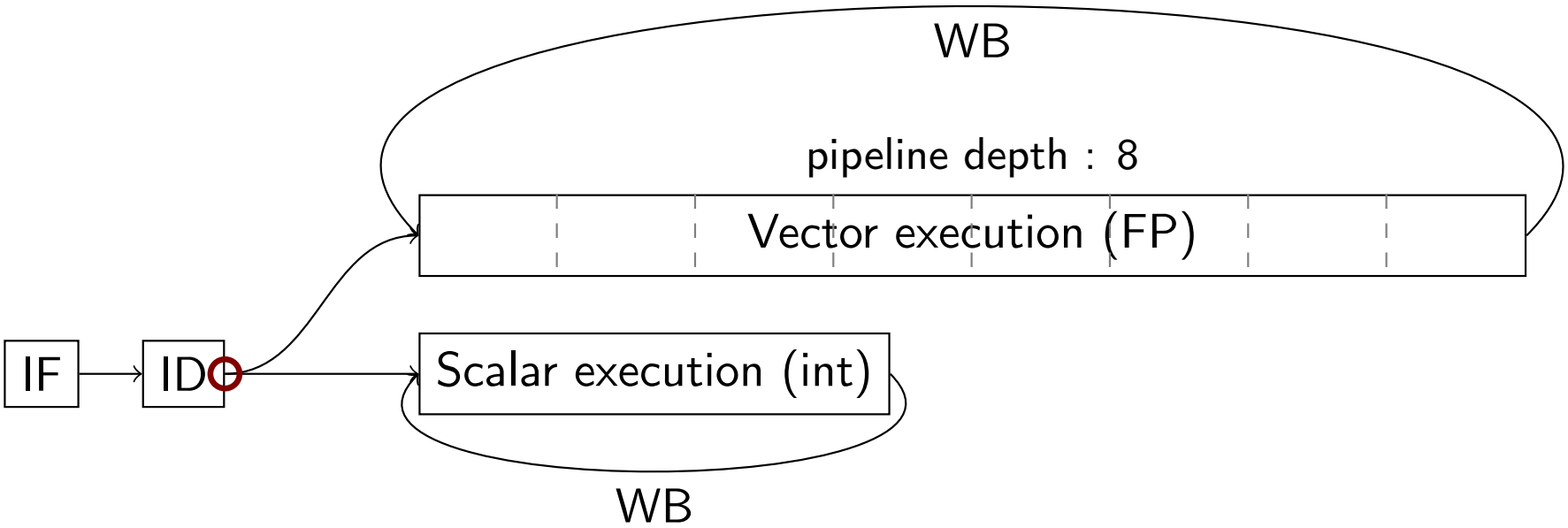
Our proposition

- Execution pipeline :



Our proposition

- Execution pipeline :



Our proposition

Problem

How can we handle hybrid operations ?

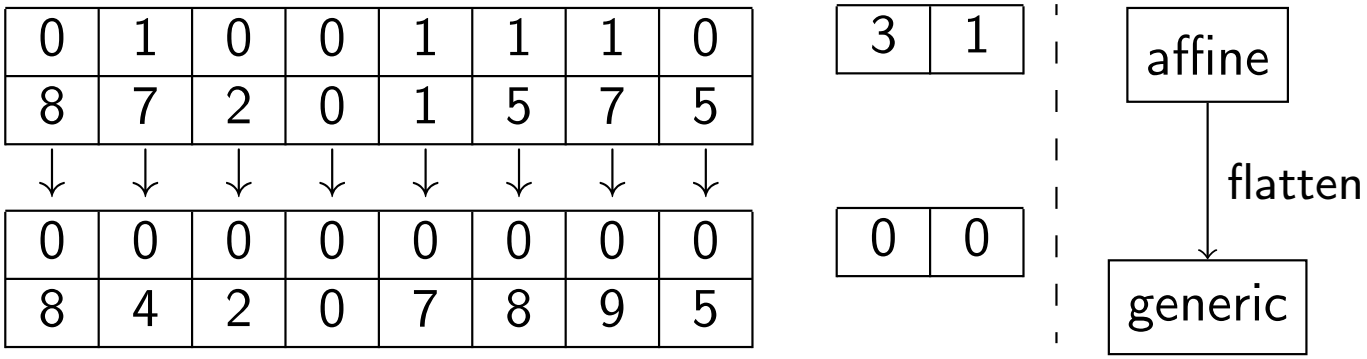
Our proposition

Problem

How can we handle hybrid operations ?

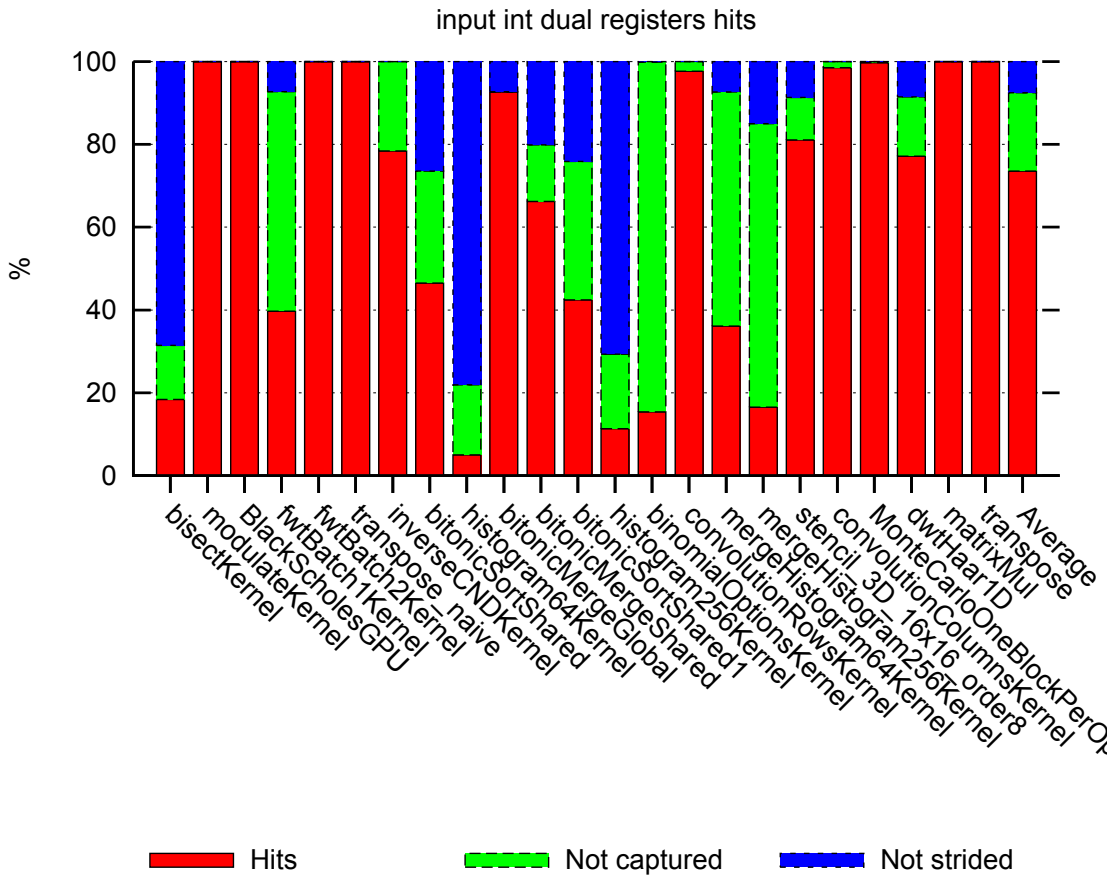
Flatten

We add a flatten operation to turn partial affine vectors into generic vectors.



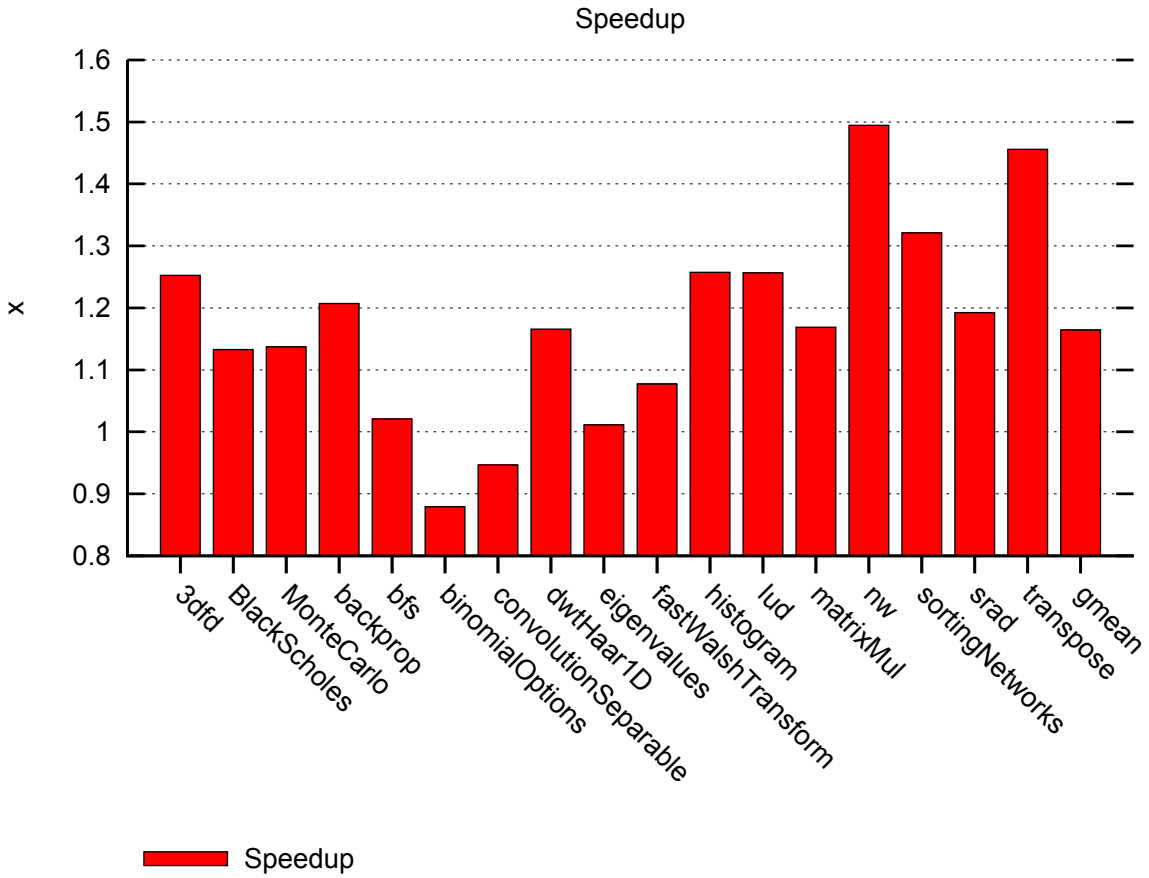
Results

- 82% average hits.



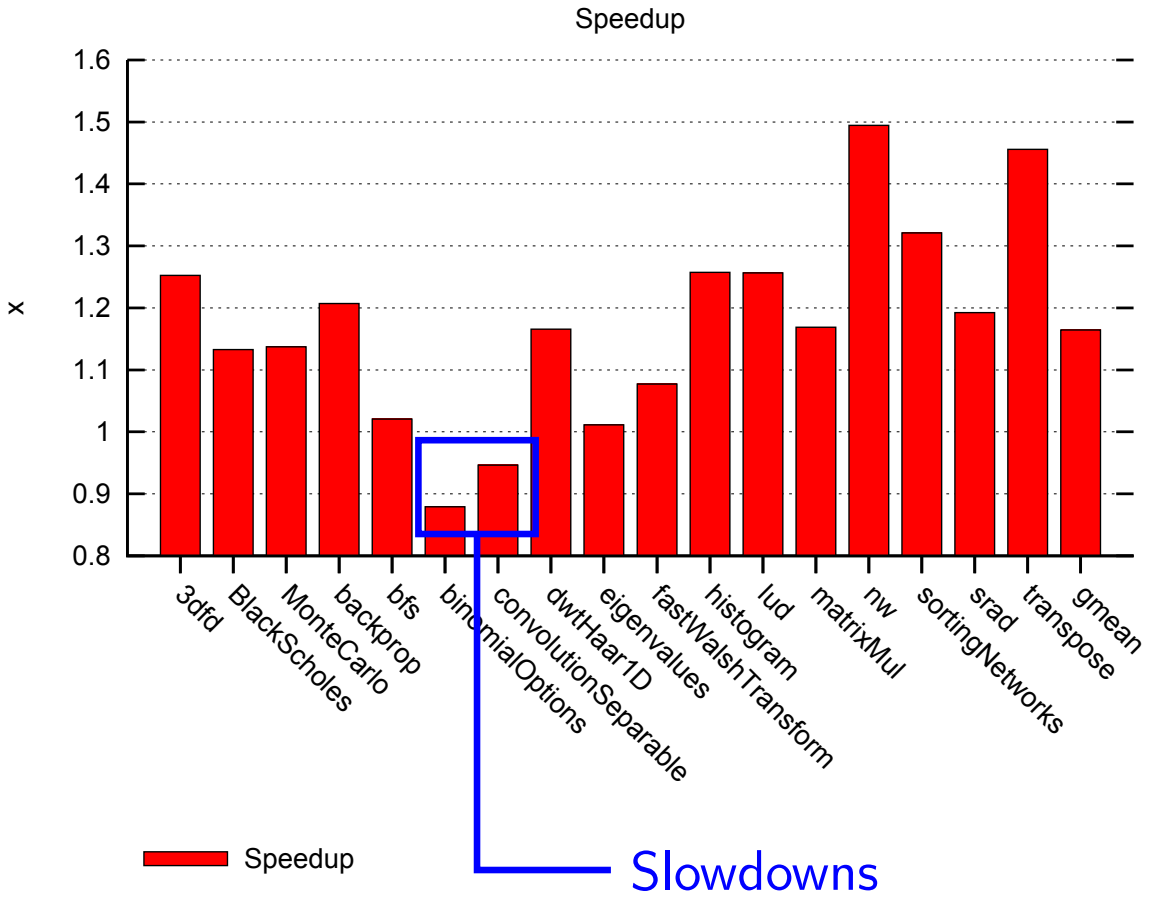
Results

- Speedup : 16%



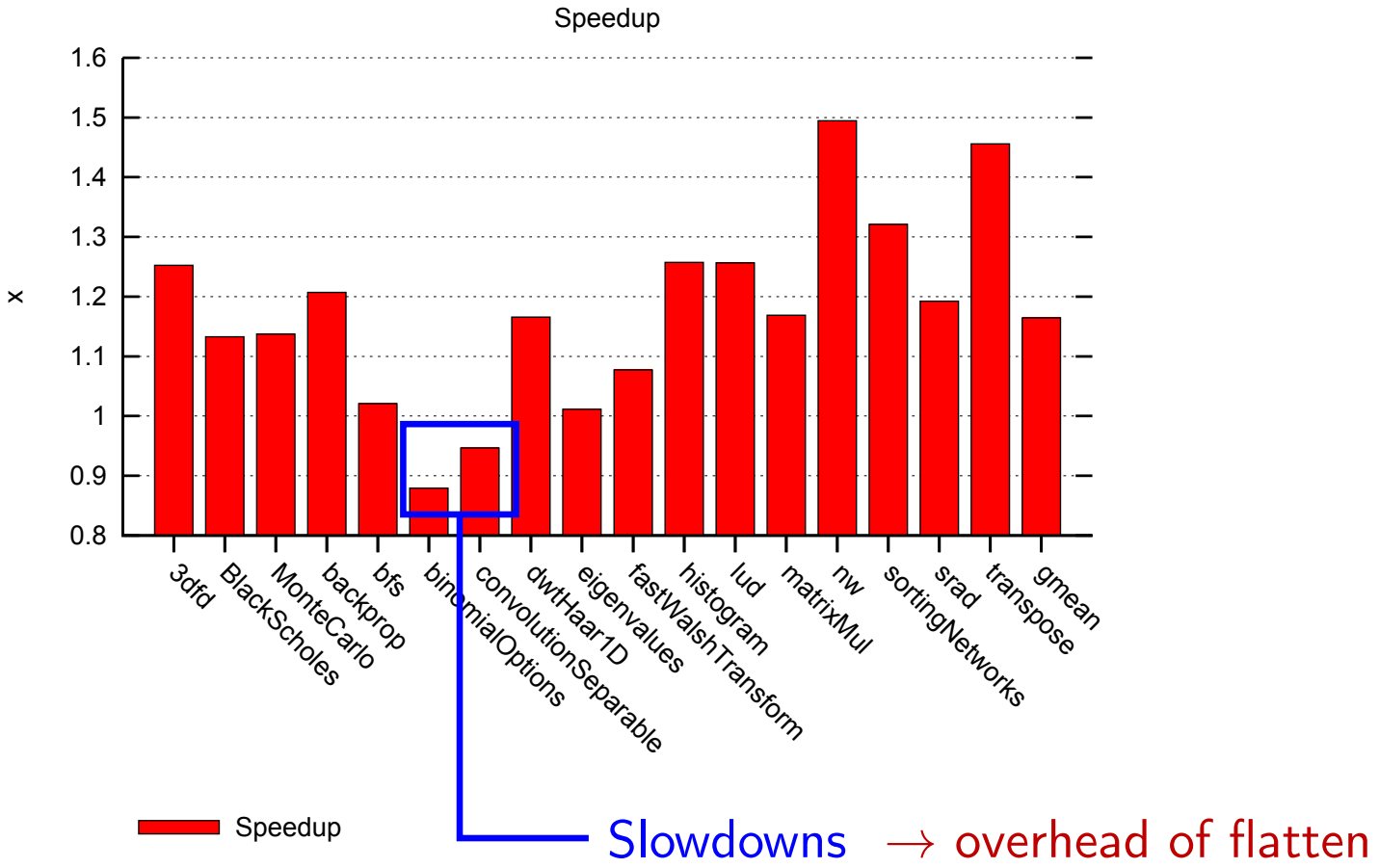
Results

- Speedup : 16%



Results

- Speedup : 16%



Results

- General speedup.

Results

- General speedup.
- Simple to implement.
  - Add of the scalar register file.
  - Implementation of the flatten operation.

Results

- General speedup.
- Simple to implement.
  - Add of the scalar register file.
  - Implementation of the flatten operation.
- Use of a scalar functional unit : lower energy use.

Results

- General speedup.
- Simple to implement.
  - Add of the scalar register file.
  - Implementation of the flatten operation.
- Use of a scalar functional unit : lower energy use.
- Slowdowns can happen.

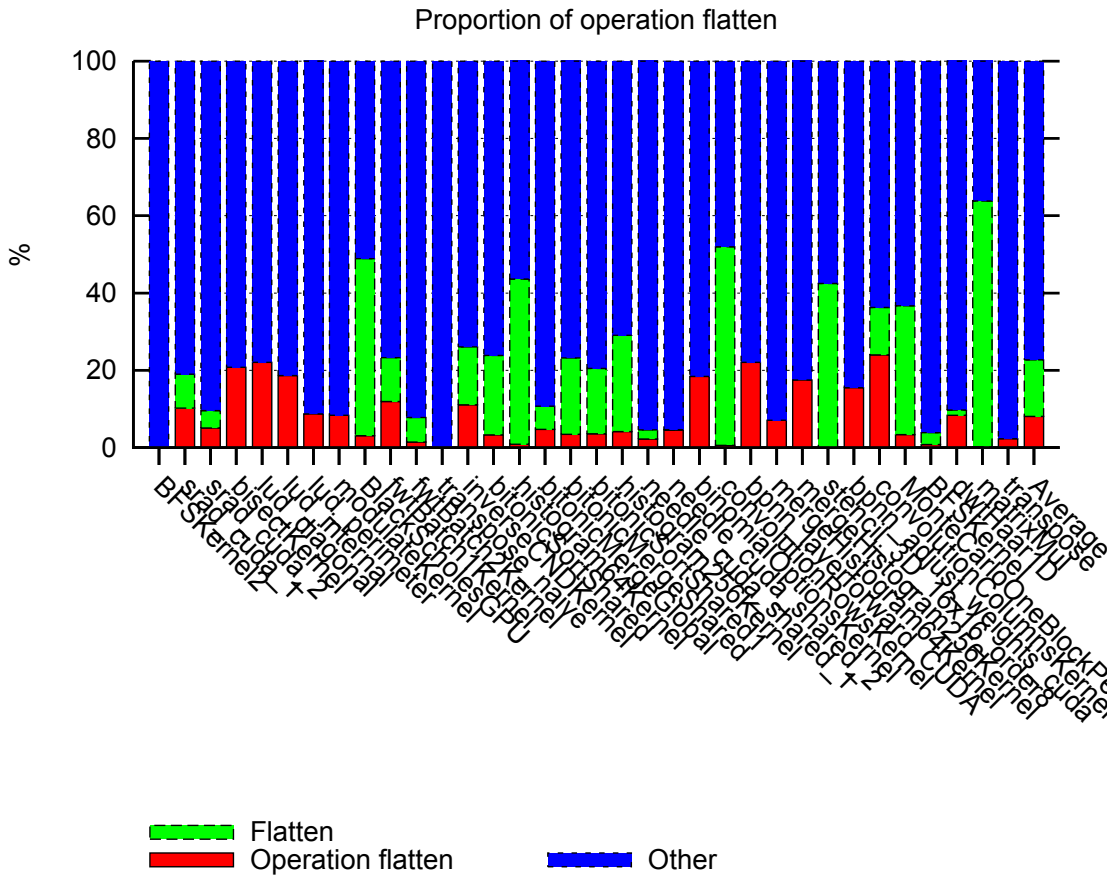
Results

- General speedup.
- Simple to implement.
  - Add of the scalar register file.
  - Implementation of the flatten operation.
- Use of a scalar functional unit : lower energy use.
- Slowdowns can happen.
  - Reason : flatten at the beginning.
  - We will try to avoid it.

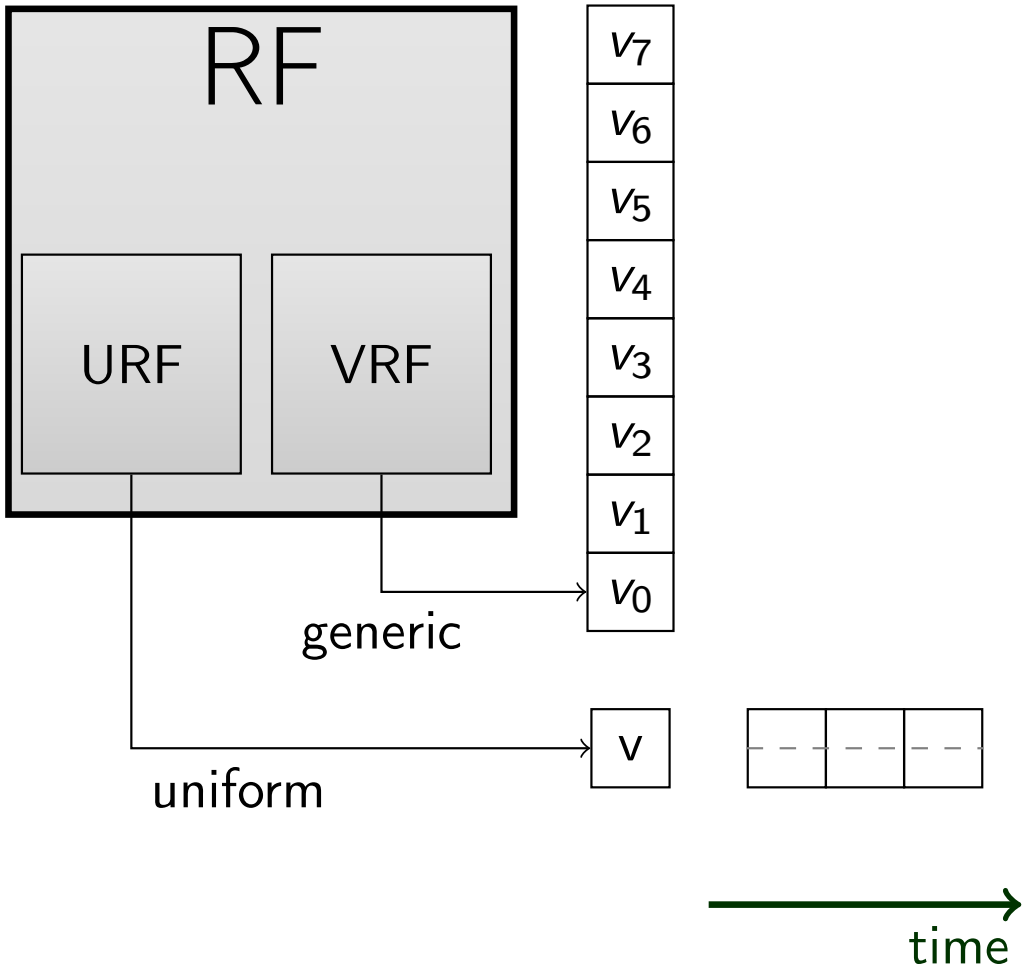
Outline

Observation

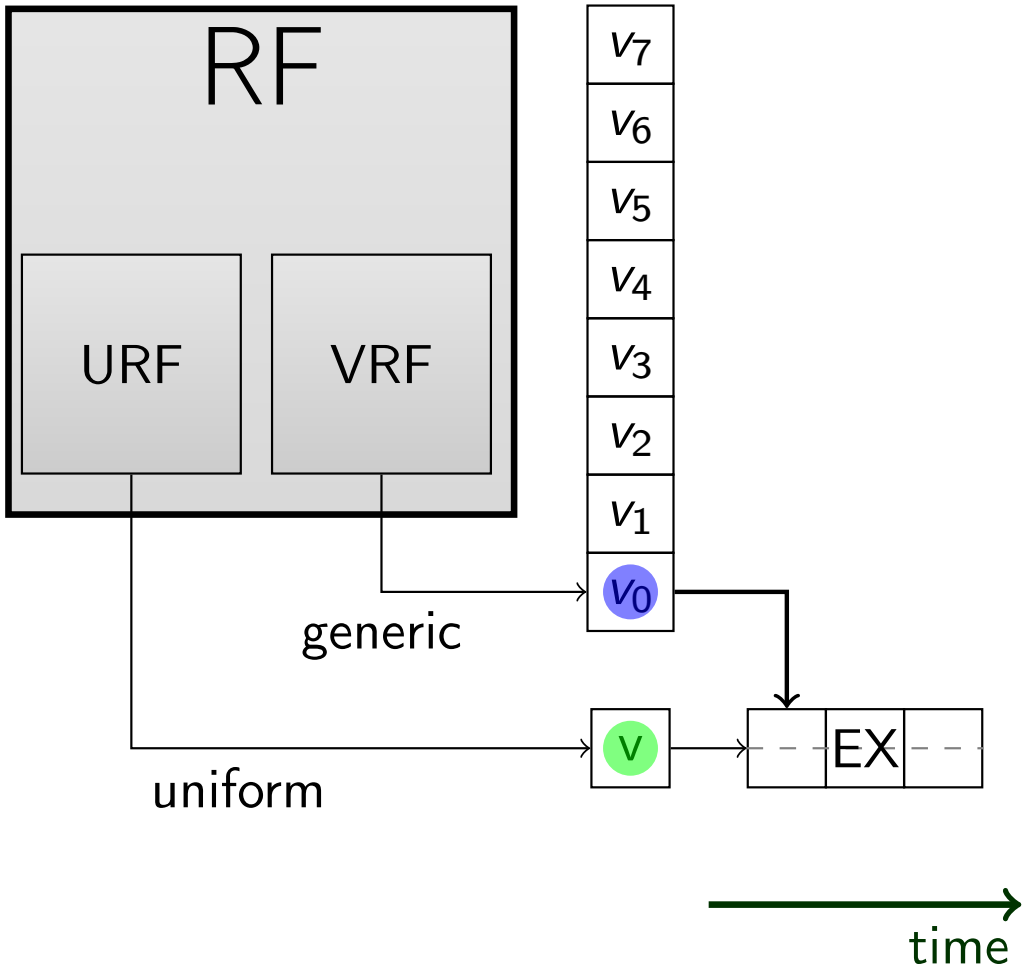
35% of flattens come from generic operations.



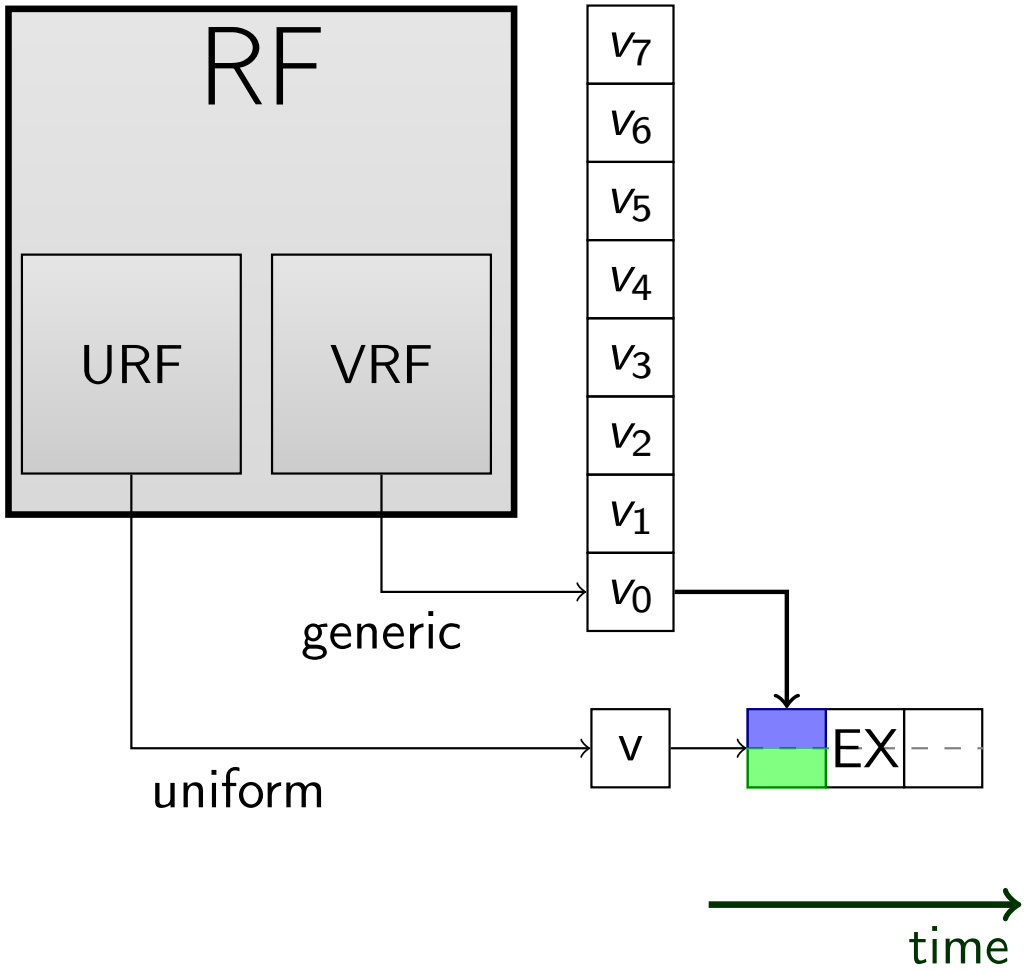
Vertical SIMT : State of the art [Kra13]



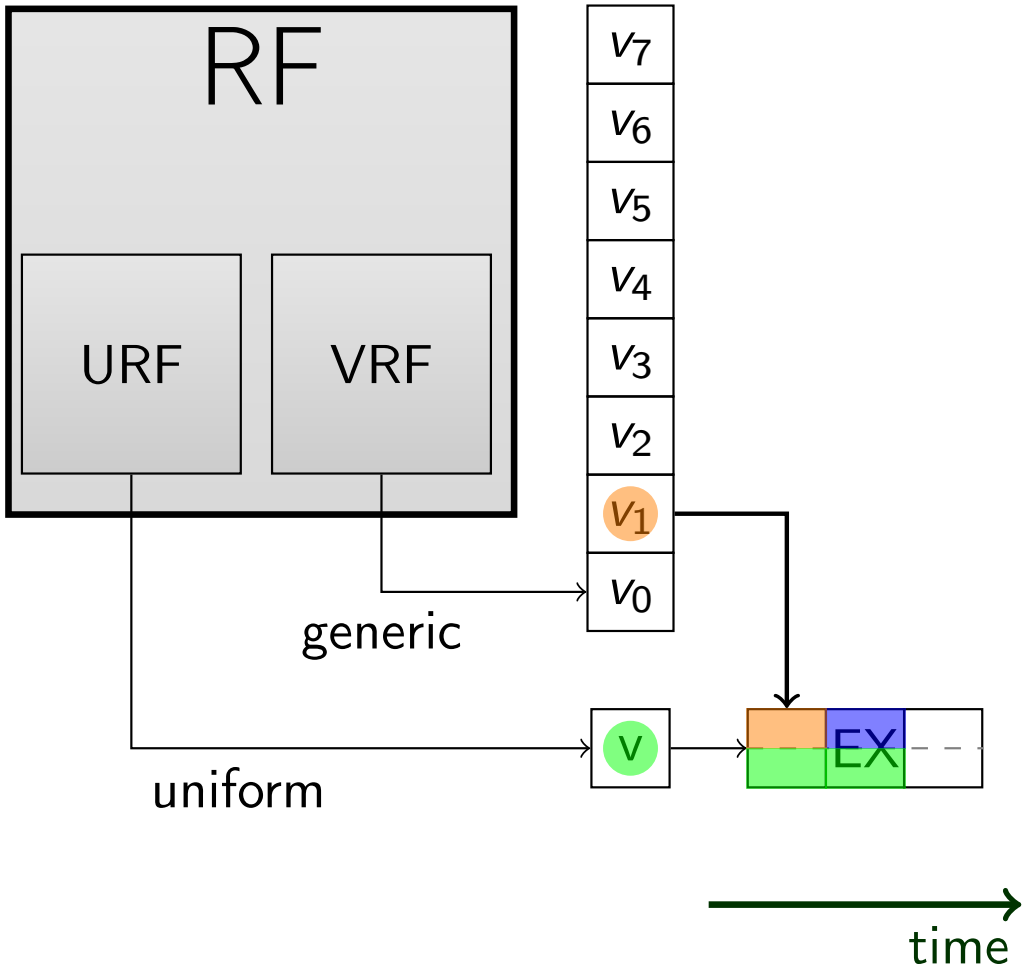
Vertical SIMT : State of the art [Kra13]



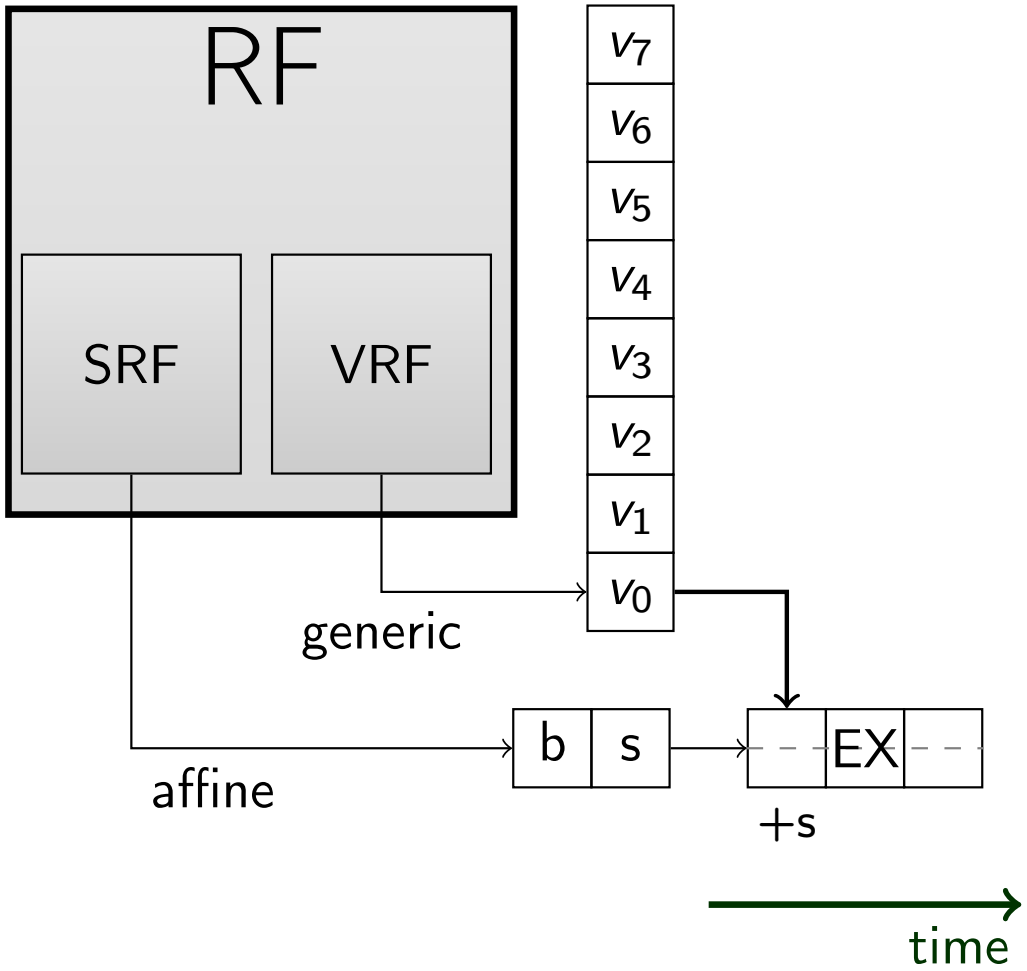
Vertical SIMT : State of the art [Kra13]



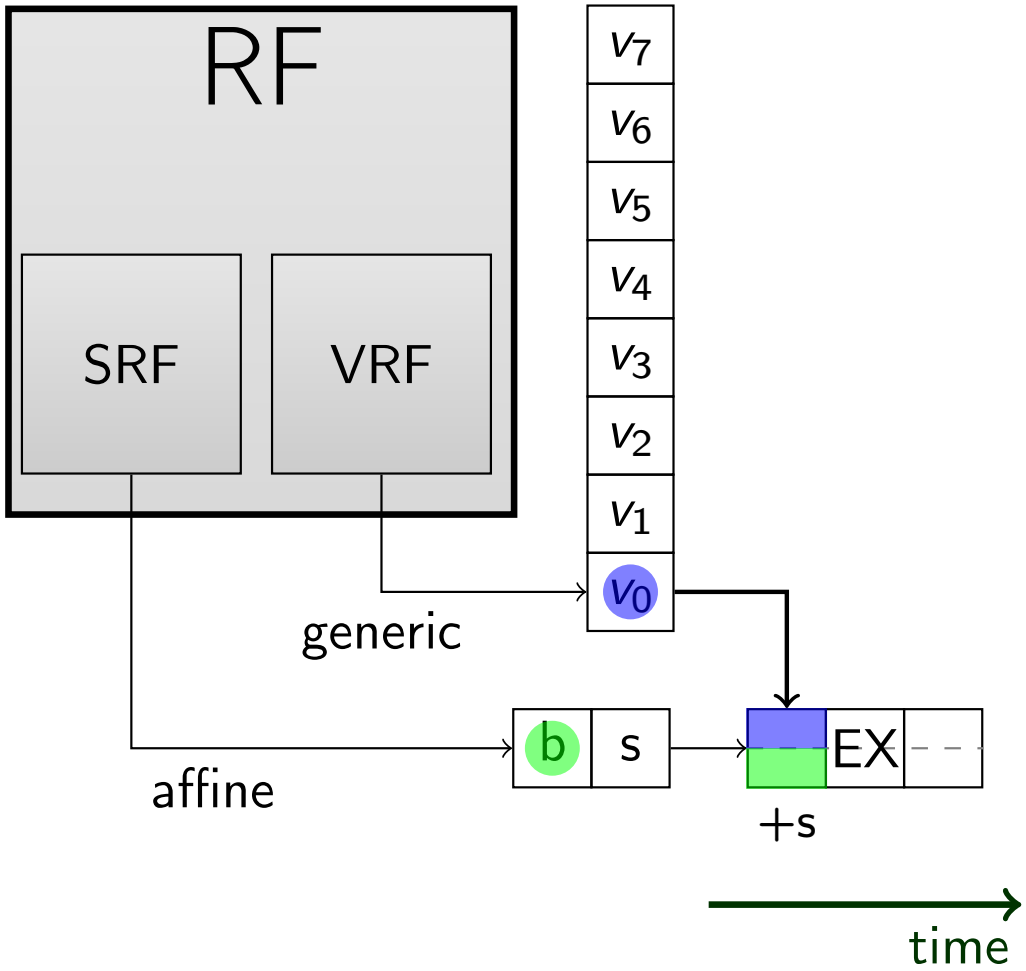
Vertical SIMT : State of the art [Kra13]



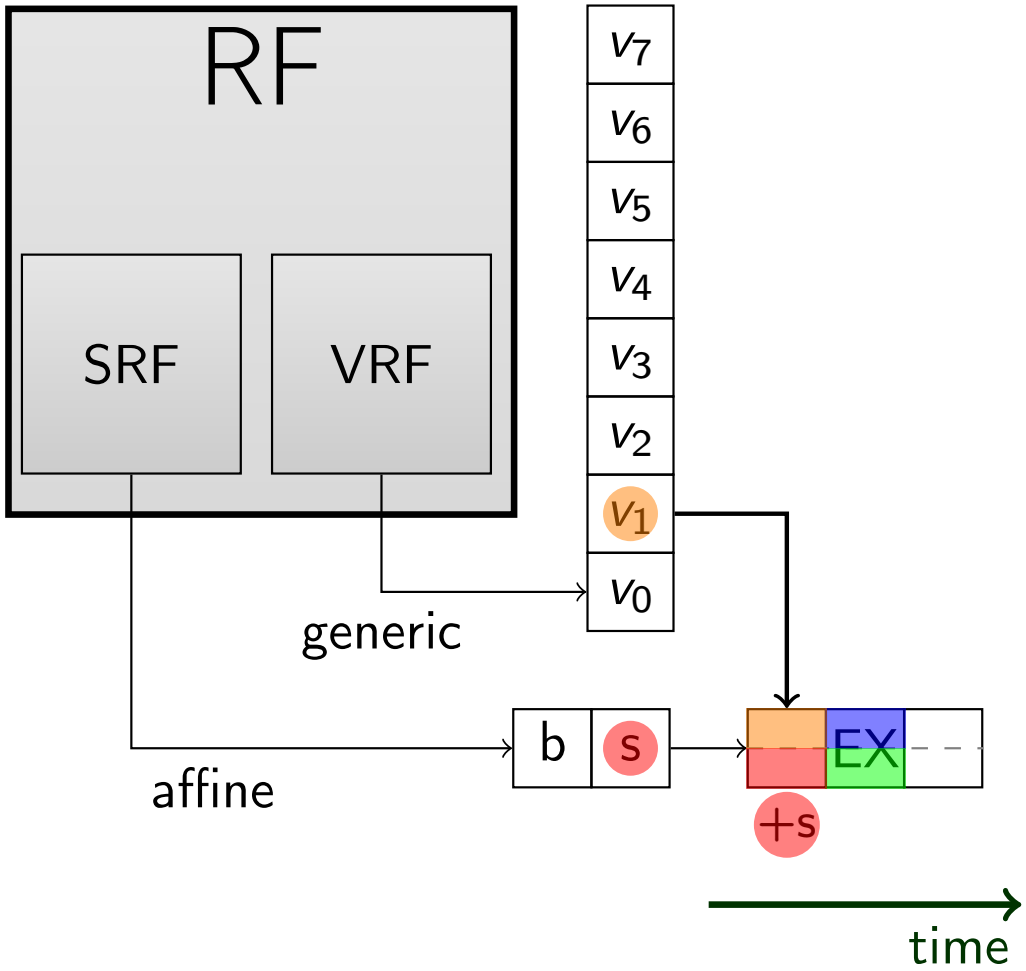
Vertical SIMT : our proposal



Vertical SIMT : our proposal



Vertical SIMT : our proposal



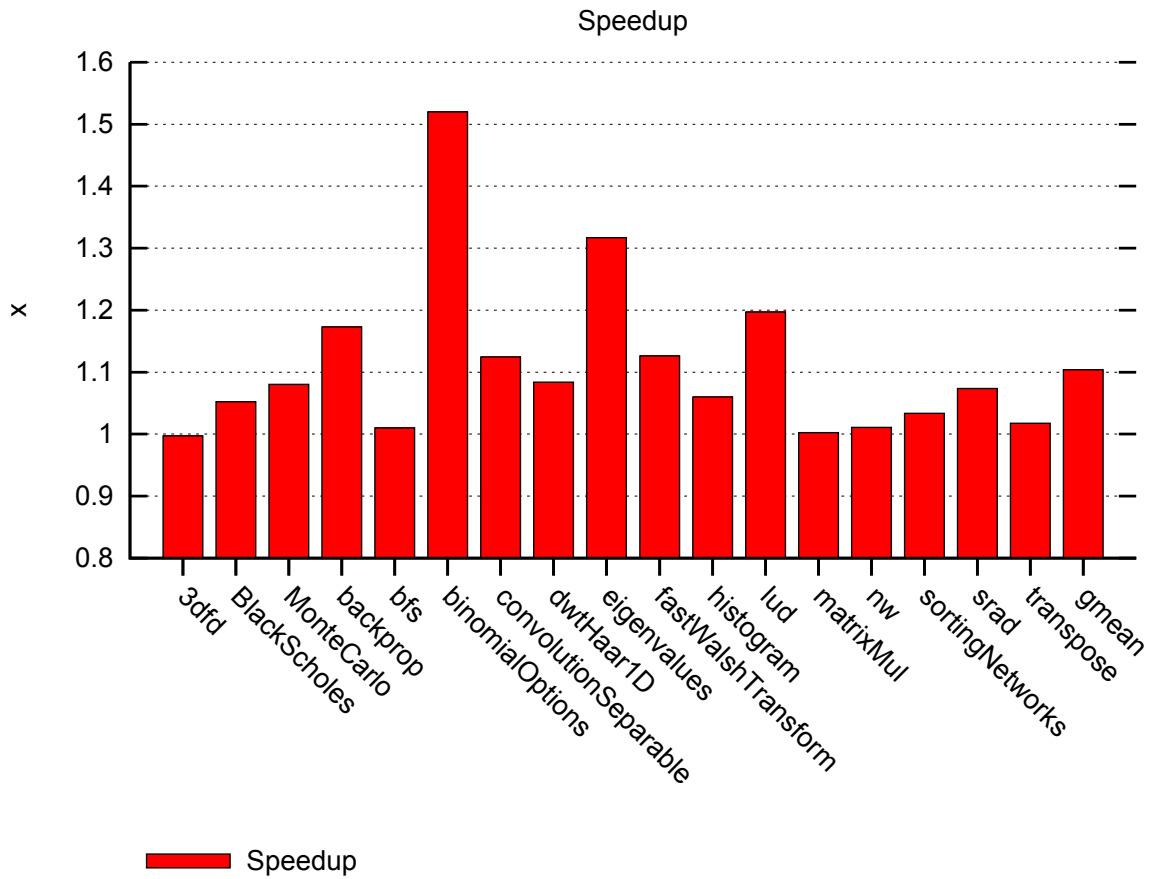
Vertical SIMT

Results :

- No affine information loss.
- No flatten execution.

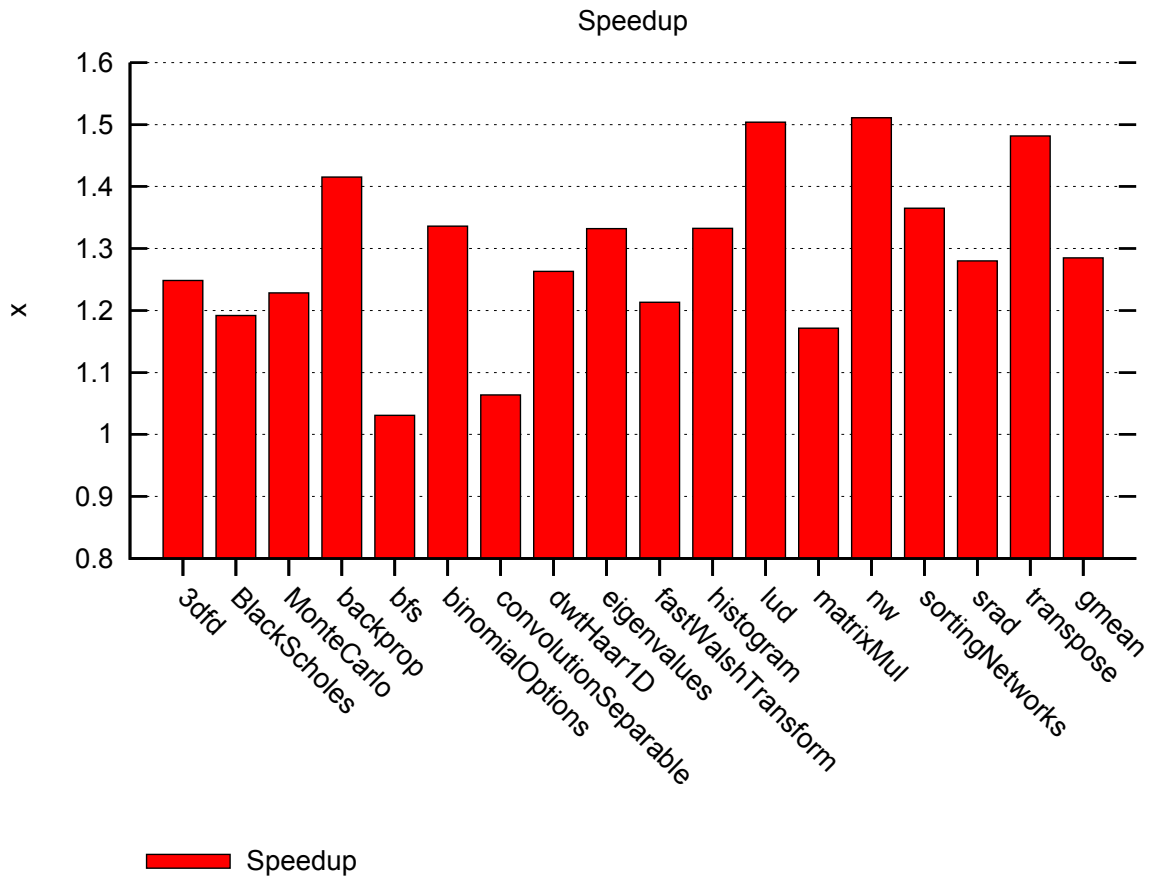
Vertical SIMT

- 10% more speedup compared to our previous solution.



Vertical SIMT

- 29% baseline speedup.



Results

- No more slowdowns.
- Higher speedup.

Results

- No more slowdowns.
- Higher speedup.
- We can use a separate register file : no more useless vector read.

Conclusion

- We proposed an architecture which decreases the energy consumption of GPU by taking advantage of inter-thread value correlation.
- Other possible ways : taking advantage of the SDMT structure to compress data.
- We could also use hints from the compiler to add an instruction word to allow software optimizations.
- Current work : combine with affine vector cache [CK+11].

Thank you very much for your attention.

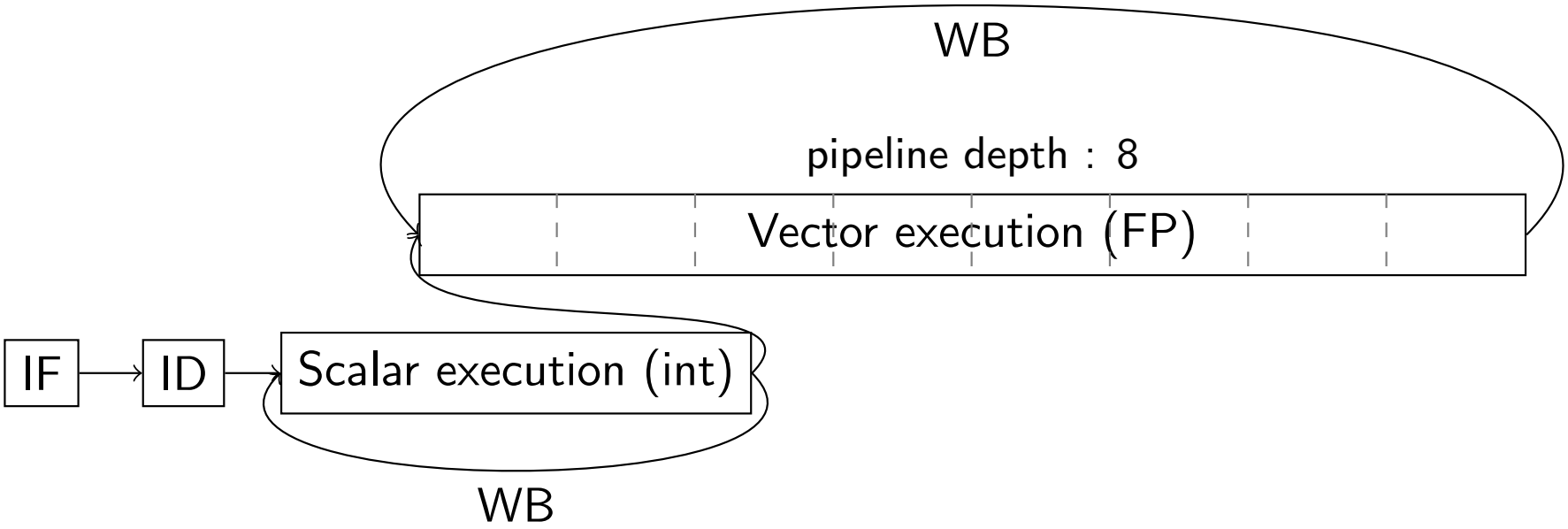
References


Sylvain Collange, David Defour, and Yao Zhang.
Dynamic detection of uniform and affine vectors in GPGPU computations.
In *Euro-Par 2009–Parallel Processing Workshops*, pages 46–55. Springer, 2010.


Sylvain Collange, Alexandre Kouyoumdjian, et al.
Affine vector cache for memory bandwidth savings.
2011.


Ronny M. Krashinsky.
Temporal SIMT execution optimization.
http://www.freepatentsonline.com/y2013/0042090.html, February 2013.

More implementable pipeline



The replay problem

Situation

Horizontal SIMT, we have a flatten operation to handle conflicts.

Problem

If there are affine operands in a generic operation, the affine vectors has to be flattened, and then the operation replayed in a generic fashion.

- Origins :
  - Hybrid operations.
  - Overflow from an affine operation.
- Costs :
  - The operation is twice longer in the pipeline (flatten + operation).
  - We loose the affine information in a read.