Hardware Cache Evolution to Support Sparse Matrix Vector Multiplications

Frédéric Rousseau
Univ. Grenoble Alpes, CNRS, Grenoble INP - UGA, TIMA, France
Valentin Isaac-Chassande, Adrian Evans, Yves Durand – CEA Grenoble

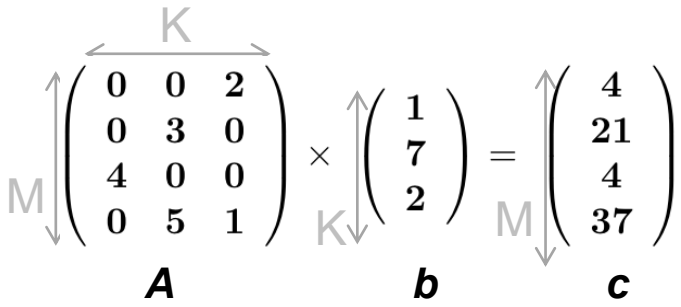
MPSoC'26

SpMV Multiplication Algorithms

- Sparse Matrix Vector Multiplication: 2 algorithms
  - *A(M, K) x b(K) = c(M)*

Algorithm 1:
MV *Inner-Product* Algorithm

```
for m ∈ [0, M[ do
    for k ∈ [0, K[ do
        c_m = c_m + A_{m,k} × b_k;
    end
end
```



Algorithm 2:
MV *Outer-Product* Algorithm

```
for k ∈ [0, K[ do
    for m ∈ [0, M[ do
        c_m = c_m + A_{m,k} × b_k;
    end
end
```

- A is read sequencially by line
- The read of b depends on NZ elements of A
- c is updated sequentially

NZ: Non Zero element

- A is read sequencially by column
- b is read sequentially
- Need Partial Outputs (PO) for c
  - Updates of PO elements depends on NZ elements of both inputs.
  - Updates of PO are not sequential

Intersection search problem (gather)

Reduction problem (scatter)

Survey of Dedicated HW Accelerators

Name	Date	Autonomy		Position in Memory Hierarchy			Most Adapted Algorithm			Hardware Support for		Sparse Kernels Addressed	
		Standalone	*Processor Assist*	*Near Processors*	*Near Cache*	*Near Main Memory*	*Inner-product*	*Outer-product*	*Gustavson*	*Gather*	*Scatter*		
Coup	2015		✓		✓			✓			✓	✓	~
EIE	2016	✓						✓		✓	✓	✓	
OuterSPACE	2018	✓						✓		✓	✓	✓	✓
ExTensor	2019	✓					✓	~		✓	~	~	✓
SPiDRE	2019		✓			✓	✓			✓		✓	~
ITS	2019	✓						✓		✓	✓	✓	
MatRaptor	2020	✓							✓	✓	✓		✓
SpArch	2020	✓						✓		✓	✓		✓
Gamma	2021	✓							✓	✓	✓		✓
SPAGHETTI	2021	✓						✓		✓	✓		✓
InnerSP	2021	✓							✓	✓	✓		✓
Sextans	2022	✓						✓		✓	✓		✓
ASA	2022		✓	✓				~	✓		✓	~	✓
Flexagon	2023	✓					✓	✓	✓	✓	✓		✓
MOSCON	2023	✓						✓		✓	✓		✓
GSCSp	2023	✓							✓	✓	✓		✓
GAS	2024	✓				✓		✓			✓	✓	✓
ACES	2024	✓							✓	✓	✓		✓
Sparm	2024	✓					✓	✓	✓	✓	✓		✓
SPADES	2024	✓					~	~	~	✓		✓	~
Vesper	2024	✓					✓	✓	✓	✓	✓	✓	✓
IOPS	2025	✓					✓	✓		✓	✓		✓
C^{3}ache	2025		✓		✓			~	~	✓	✓		✓
ScanNow	2025	✓							~	✓	✓	MV	✓

Hypothesis:

- kernel: SpMV
- Algorithm: Outer-Product
- Scatter (reduction problem)
- Processor assist near cache

Large SpM and CSC Representation

- Large matrices

SparseSuite Matrix Collection
62 square matrices

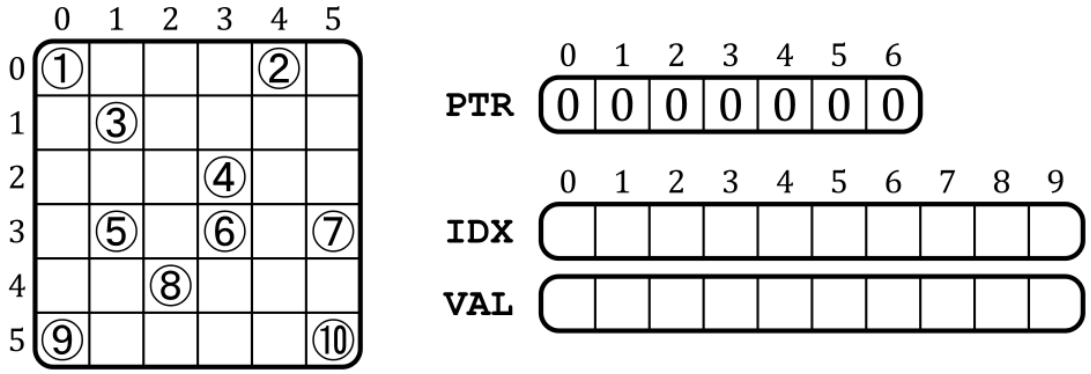
$M \in [124, 1\,971\,281]$

$$\text{Density} = \frac{\text{NZ elements}}{M \times M}$$

SparseSuite Matrix Collection
$d \in [10^{-6}, 10^{-2}]$

- CSC → simplest, mostly used

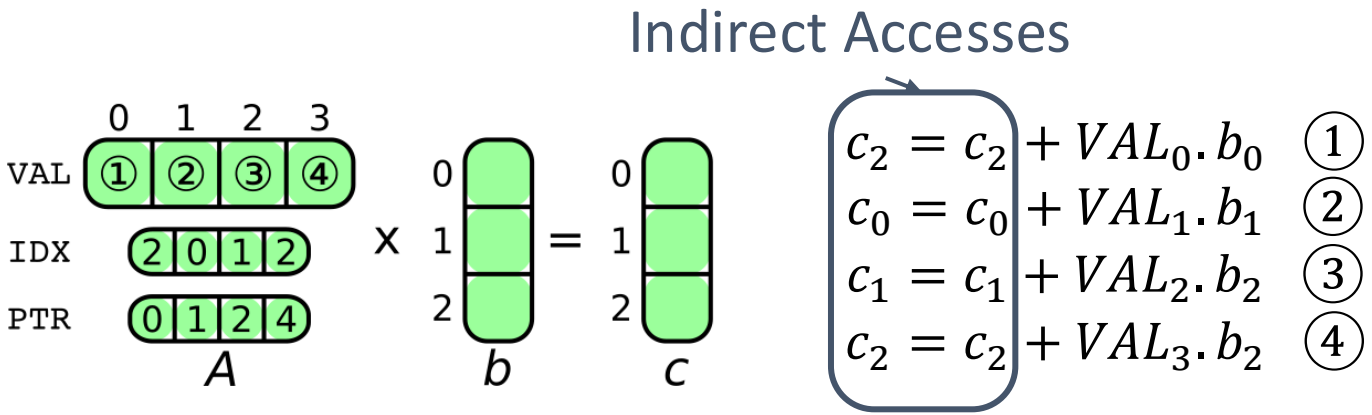
CSC (Compressed Sparse Column)



[3] Reginald P. Tewarson. 1973. Sparse matrices. Vol. 69. Academic press New York, State University of New York, Stony Brook, NY.
[4] Yousef Saad. 2003. Iterative Methods for Sparse Linear Systems (second ed.). Society for Industrial and Applied Mathematics.

SpMV using CSC for Outer-Product (OP)

- **SpMV**:
  - A is stored with a sparse format such as CSC
  - With the **Outer-Product SpMV**, **"random" accesses** occurs on the **output vector c**, when accumulating the **partial sum**



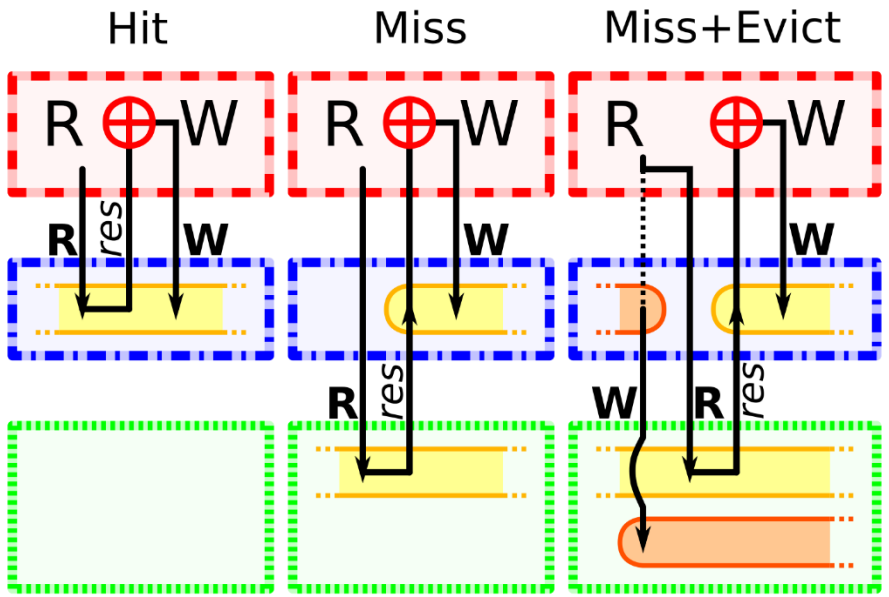
Why Generic Cache (GC) failed ?

- A **Reduction** on a vector v: $\boldsymbol{v_{key} = v_{key} + value}$
  1. A **READ** from memory to get v_{key}
  2. An **accumulation** to update v_{key} with $value$
  3. A **WRITE** to memory to store the updated v_{key}
  - We note this operation **RED**: $\boldsymbol{RED(key, value)}$

Processor

Cache

Main memory



R: Read / res: Read Response / W: Write / ⊕: Accumulation

Processor / Cache / Main Memory

Hit: no memory access

Miss: Read a cache line from mem.,

Miss+Evic: Written back cache line, then read cache line containing v_{key}

For large matrix, a small part of vector c can reside in cache, so Miss+Evict arises frequently

Is a Reduction Cache (RedC) a Solution ?

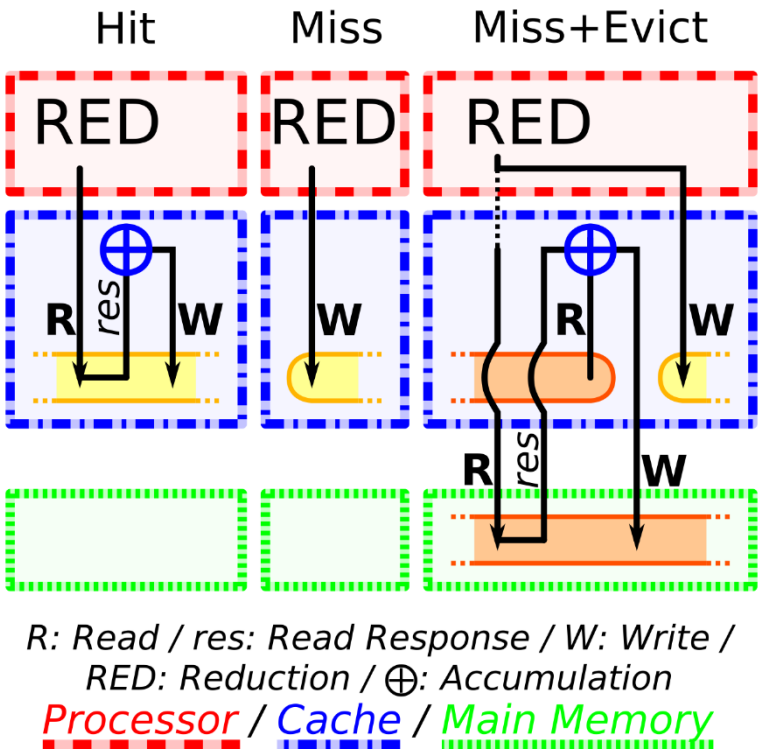
- A reduction cache is a data-cache which supports reduction instructions (RED) from the processor and perfoms accumulations locally (near-memory)

RED: $\boldsymbol{RED(key, value)}$

Processor

Cache

Main memory



R: Read / res: Read Response / W: Write / RED: Reduction / ⊕: Accumulation

Processor / Cache / Main Memory

Hit: no memory access

Miss: The cache allocates a new line with zeros, and insert the value (inconsistency with main memory).

Miss+Evic: A Read is performed, all values of the line are accumulated in the cache, and result line is written back to the memory.
The new value is written in the cache line

S. Srikanth, A. Jain, T. M. Conte, E. P. Debenedictis, and J. Cook. 2021. SortCache: Intelligent Cache Management for Accelerating Sparse Data Workloads. *ACM Trans. Archit. Code Optim.* 18, 4, Article 56 (sep 2021)

What are still missing to be efficient ?

- Reduction cache is efficient for reduction
  - Reductions are of type « fire-and-forget »
    - The CPU does not wait (even when eviction occurs)
  - No data transfer in case of miss
    - Generic cache: 0 + 1 + 2 = **3**
    - Reduction cache: 0 + 0 + 2 = **2**

- Isolated elements that disturb the efficiency of the cache !
  - Only one element in a cache line
  - Cache line eviction for one element to be written

- Reduction cache works well for reductions, not for random accesses

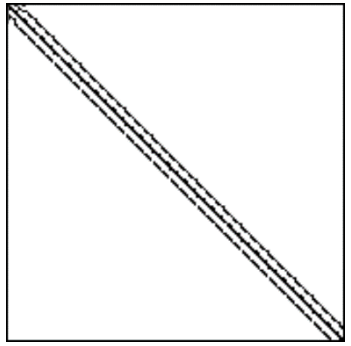
Would it be possible to use the matrix structure ?

Banded Matrices

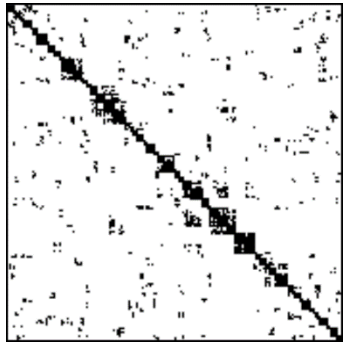
- We are dealing with banded matrices
  - Dense region around its main diagonal
  - Sparse region away from the diagonal

Applications from linear algebra systems

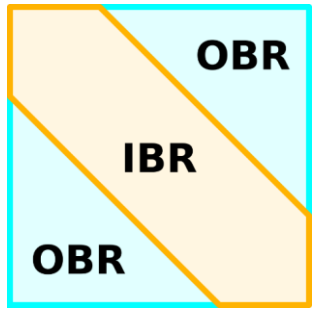
Perfect band
All elements in a small region
→ locally denser



Band + "Random"
Also elements elsewhere with low density



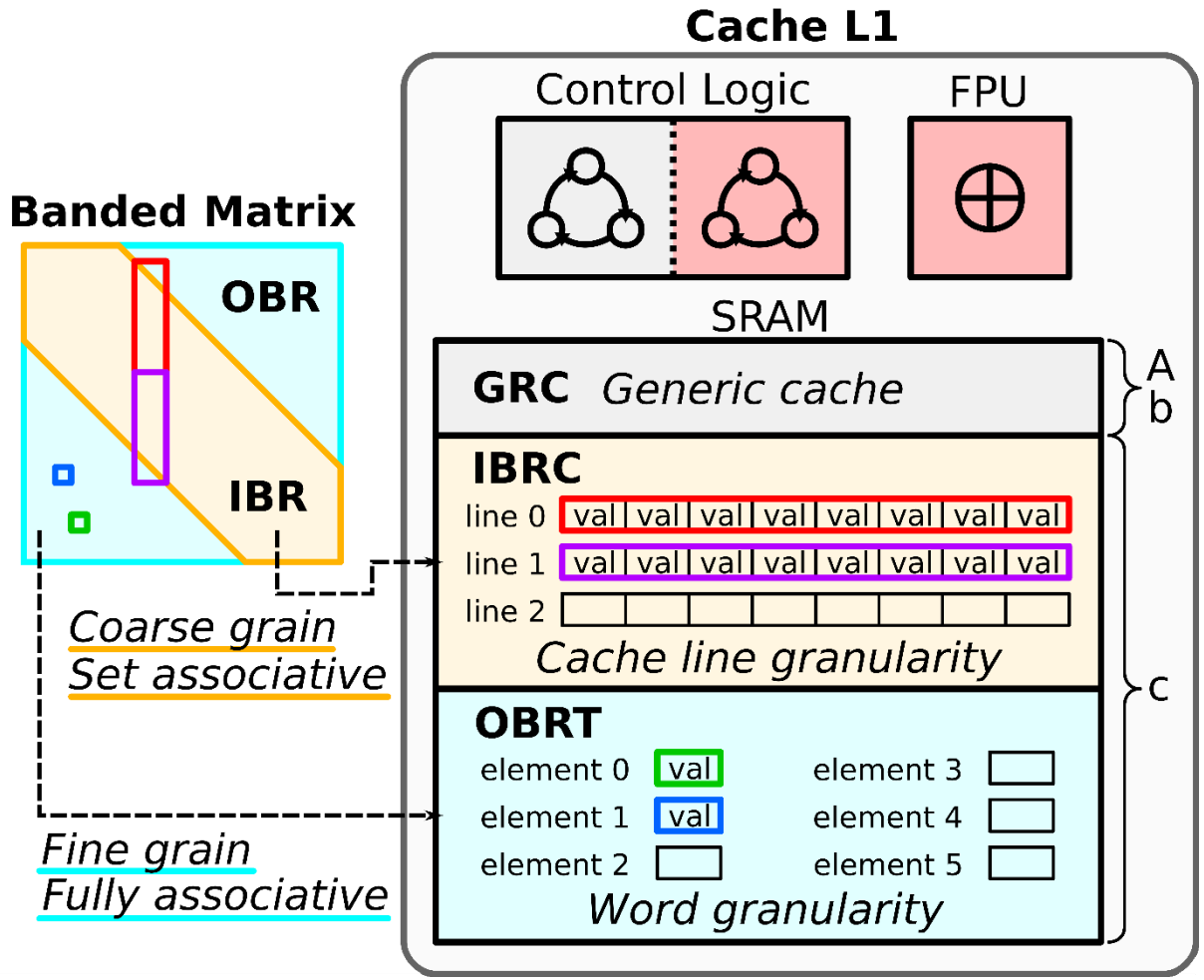
OBR: Out-of-Band Region (sparse)
IBR: In-Band Region (dense)



In SparseSuite, 48 (of 62) matrices are banded

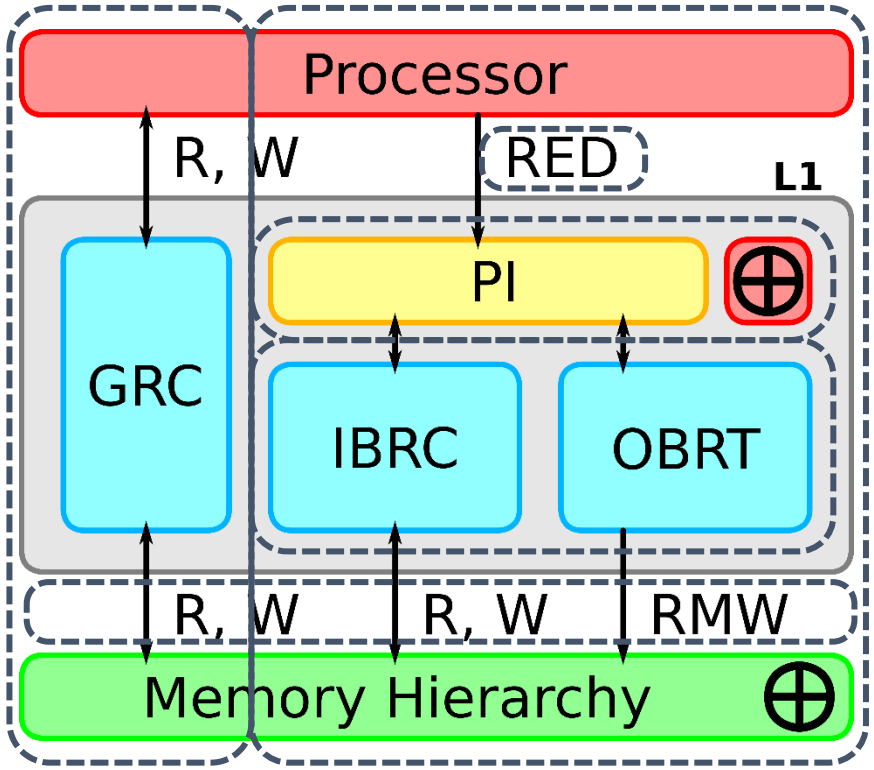
Concept of SpDCache (SparseDataCache)

- Add new mode and **control logic**
- Add a **FPU** (Floating Point Unit)
- Split the cache memory into **three regions:**
  - The **IBRC** (In-Band Region Cache) for the IBR
    - Reductions only
    - Granularity of a cache line
  - The **OBRT** (Out-of-Band Region Table) for the OBR
    - Reductions only
    - Granularity of a single word
  - The **GRC** (Generic Region Cache)
    - Generic memory transactions



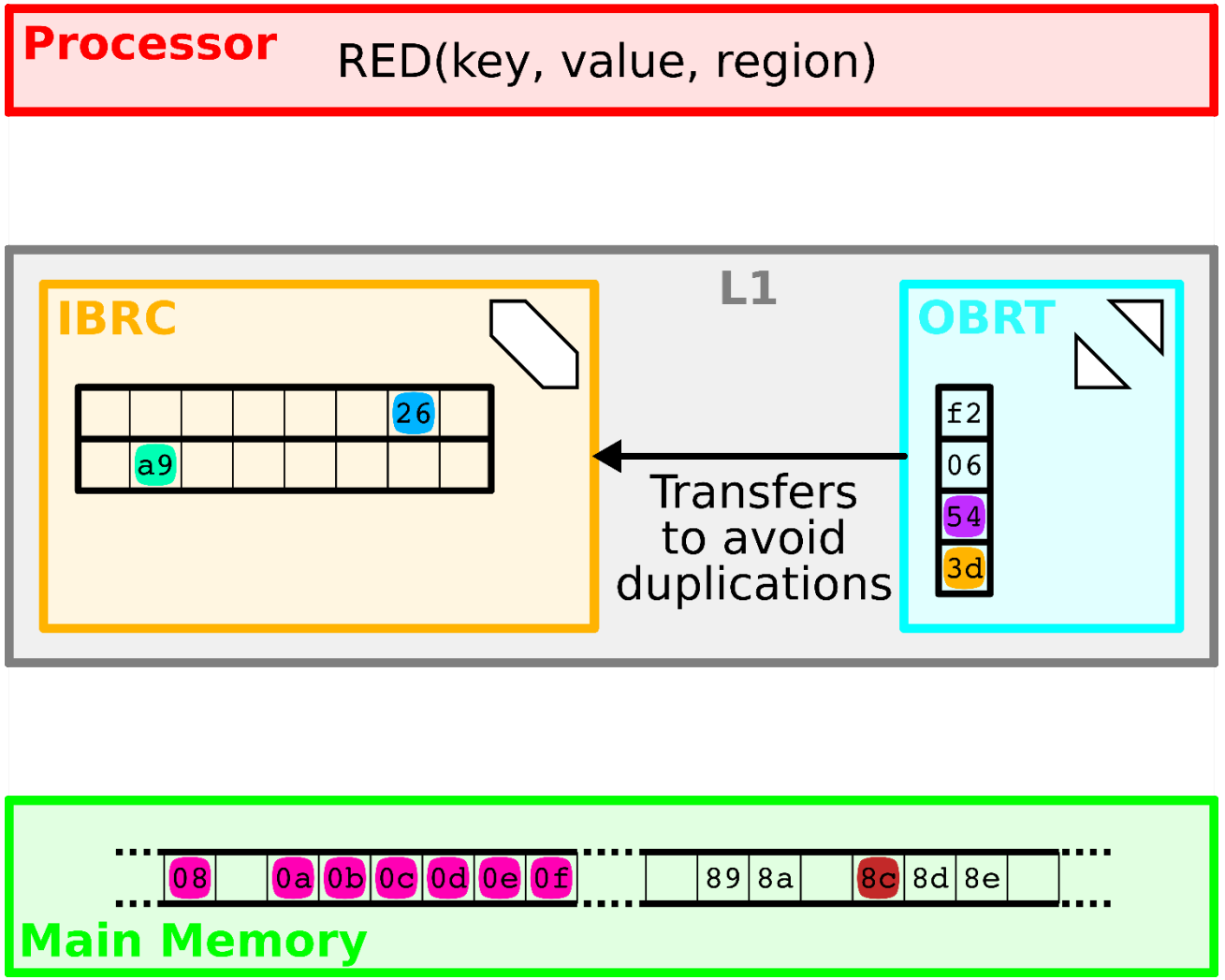
SpDCache: Architecture Overview

- In the **L1 cache**:
  - The **GRC** (N-Way Set Associative / Cache Lines)
  - The **IBRC** (N-Way Set Associative / Cache Lines)
  - The **OBRT** (Fully Associative / Single Entries)
    - Uses **atomic Read-Modify-Write** (RMW)
  - The **PI** (Processor Interface)
    - Process the RED instructions
  - An **arithmetic unit** (accumulation or FPU)
    - To process accumulations within the cache
- In the **memory**:
  - An accumulation unit



SpDCache: Reduction step by step

- **RED** transaction → **Fire-and-forget**
- **Hit / Miss**
  - **No** memory transaction
  - Insertion or accumulation **in cache**
- **Miss+Eviction from IBRC**
  - **2** memory transactions
  - Accumulation **in cache**
- **Miss+Eviction from OBRT**
  - **1** memory transaction
  - **Fire-and-forget**
  - Accumulation **in main memory**



SpDCache: Results and performances

TABLE I: Decrease in Memory Traffic and Improvement in Cache Utilization from Python Simulation

Matrix	NNZ / M	Output Memory Traffic			Evicted Cache Line Utilization			Cache Memory Utilization (%)		
		RedC	SpDC		GC	RedC	SpDC	GC	RedC	SpDC
		%/GC	%/GC	% ATO	Avg	Avg	Avg	Avg	Avg	Avg
consph	6.0m / 83k	23.5	14.6	27.9	6.17	6.60	7.87	48.5	89.7	96.8
roadNet-CA	5.5m / 2.0m	61.9	25.3	14.0	4.31	3.75	7.64	39.6	50.1	83.0
pdb1HYS	4.3m / 36k	11.9	8.1	22.5	5.89	6.97	7.89	48.7	86.8	95.3
offshore	4.2m / 0.3m	77.9	8.4	43.4	1.96	1.87	7.23	31.7	27.1	80.1
webbase-1M	3.1m / 1.0m	71.5	34.9	37.0	5.77	5.40	8.00	42.4	65.7	90.4
ramage02	2.9m / 17k	9.5	4.6	44.1	5.21	6.42	7.85	47.7	81.9	94.1
filter3D	2.7m / 0.1m	50.9	10.6	35.3	2.72	3.28	7.14	35.9	50.4	86.0
2cubes_sphere	1.6m / 0.1m	58.0	8.7	33.5	2.29	2.24	7.49	36.5	31.2	79.7
mac_econ_fwd500	1.3m / 0.2m	28.3	22.9	4.6	4.40	6.04	7.62	42.4	70.7	73.6
scircuit	1.0m / 0.2m	71.0	22.6	35.4	3.95	3.60	7.94	40.2	46.7	91.0
wiki-Vote	0.1m / 8.3k	62.1	5.1	29.5	1.74	1.81	5.68	29.4	25.0	62.2
Geometric Mean		**39.2**	**12.3** [a]	**25.9**	**3.69**	**3.91**	**7.46** [b]	**39.7**	**51.9**	**84.1** [c]
raefsky3	1.5m / 21k	19.7	26.3	0.0	7.98	8.00	8.00	49.5	95.4	65.0
msc10848	1.2m / 11k	35.9	14.4	70.9	4.96	5.73	7.86	47.5	72.0	92.6

[a]Decrease by 8.1× compared to GC. [b]Increase by 2.0× compared to GC. [c]Increase by 2.1× compared to GC.

Output Memory Traffic: *Decrease by **8.1** compared to GC, and **3.2** compared to RedC*
Evicted Cache Line utilization: *Increase by **2.0** compared to GC, and **2.0** compared to RedC*
Cache Memory Utilization: *Increase by **2.1** compared to GC, and **1.6** compared to RedC*

Speedup*: **10%** compared to RedC*

Status of SpDCache

- RTL implementation in progress based on HpDCache (OpenHW group)
  - Under test in a CVA6 RISCV architecture
  - Only integer unit (not yet FPU) for accumulation

- The size of the 3 regions, GCR, IBRC and OBRT is statically defined
  - GCR 25%, IBRC 50%, OBRT 25%
  - Dynamic configuration would be great !

- Matrices are stored in CSC sparse format
  - How to determine the target region: RED(key, value, region)
    - 2 solutions:
      - Computation on-the-fly (but additional instructions in critical loop)
      - Pre-computation of the region
        - Looking for new CSC format integrating the region

Hardware Cache Evolution to Support Sparse Matrix Vector Multiplications

Frédéric Rousseau
Univ. Grenoble Alpes, CNRS, Grenoble INP - UGA, TIMA, France
Valentin Isaac-Chassande, Adrian Evans, Yves Durand – CEA Grenoble

MPSoC'26

Survey of Dedicated HW Accelerators

- **Several characteristics**
  - **Kernel:** Matrix-Matrix | Matrix-Vector | Generic
  - **Algorithm:** Inner-Product | Outer-Product | Gustavson | Custom Tiling Method
  - **Problems addressed:** Gather (Intersections) | Scatter (Reductions)
  - **Accelerator type:** Standalone | Processor Assist
  - For processor assisted accelerators, **position in the memory hierarchy:**
    Near Processor | Near Caches | Near Main Memory