# Modèle théorique du produit matrice vecteur sur corps finis.

Bastien Vialla

sous la direction de :
Pascal Girogi, Laurent Imbert

LIRMM CNRS UM2



# Motivations

- La résolution de systèmes linéaires creux intervient dans beaucoup de problèmes de cryptanalyse.
- Les matrices sont creuses et de grandes dimensions

# Motivations

- La résolution de systèmes linéaires creux intervient dans beaucoup de problèmes de cryptanalyse.
- Les matrices sont creuses et de grandes dimensions

**Définition**

Une matrice de dimensions $n$ est dite creuse si elle possède $O(n \log^2(n))$ éléments non nuls.

# Motivations

- La résolution de systèmes linéaires creux intervient dans beaucoup de problèmes de cryptanalyse.
- Les matrices sont creuses et de grandes dimensions

**Définition**

Une matrice de dimensions $n$ est dite creuse si elle possède $O(n \log^2(n))$ éléments non nuls.

- Impossible de stocker la matrice en format dense.
- Les algorithmes classiques (Gauss, ...) densifient la matrices.
- Utilisations d'algorithmes itératifs : Wiedemann, Lanczos, ...
  - Approche boîte noire.
  - Améliorent la complexité.

# Motivations

- Produit d'une matrice par un vecteur, spMv, est une opération clé.
- Les performances dépendent du format de stockage de la matrice.

# Motivations

- Produit d'une matrice par un vecteur, spMv, est une opération clé.
- Les performances dépendent du format de stockage de la matrice.
  - COO
  - CSR, CSR-OO, CSR-ZigZag, CSR-Delta, CSR-Delta-ZigZag,...
  - CSC, CSC-OO, CSC-ZigZag, CSC-Delta, ...
  - ELL, cELL, ELL-CSR, ELL-COO, ELL-COO-CSR, ...
  - BlockCSR, BlockCSC
  - Quadtree, Bitmap Recursive Format
  - JDS, SELL, SELL-$\sigma$-c
  - JAG, Diag, Skyline
  - CRX, ...

# Motivations

- Produit d'une matrice par un vecteur, spMv, est une opération clé.
- Les performances dépendent du format de stockage de la matrice.
  - COO
  - CSR, CSR-OO, CSR-ZigZag, CSR-Delta, CSR-Delta-ZigZag,...
  - CSC, CSC-OO, CSC-ZigZag, CSC-Delta, ...
  - ELL, cELL, ELL-CSR, ELL-COO, ELL-COO-CSR, ...
  - BlockCSR, BlockCSC
  - Quadtree, Bitmap Recursive Format
  - JDS, SELL, SELL-$\sigma$-c
  - JAG, Diag, Skyline
  - CRX, ...
- Pas de notion de convergence et de solution approchée.
- Arithmétique modulaire, et multiprécision.

# Motivations

- Produit d'une matrice par un vecteur, spMv, est une opération clé.
- Les performances dépendent du format de stockage de la matrice.
  - COO
  - CSR, CSR-OO, CSR-ZigZag, CSR-Delta, CSR-Delta-ZigZag,...
  - CSC, CSC-OO, CSC-ZigZag, CSC-Delta, ...
  - ELL, cELL, ELL-CSR, ELL-COO, ELL-COO-CSR, ...
  - BlockCSR, BlockCSC
  - Quadtree, Bitmap Recursive Format
  - JDS, SELL, SELL-$\sigma$-c
  - JAG, Diag, Skyline
  - CRX, ...
- Pas de notion de convergence et de solution approchée.
- Arithmétique modulaire, et multiprécision.

# Des chiffres

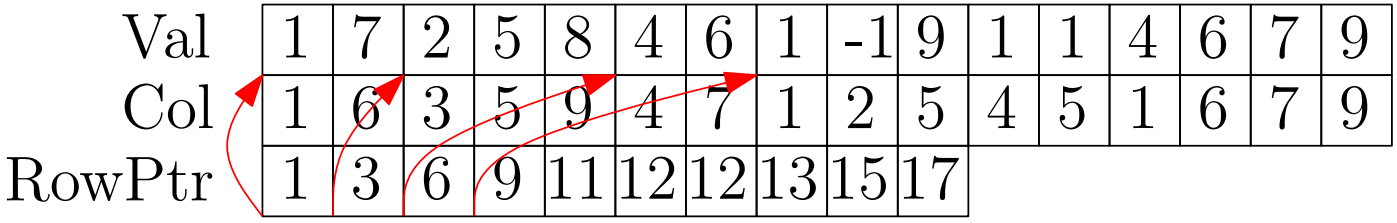
Matrice de FFS809

- Dimensions : $3600267 \times 3600267$
- $\approx 100$ éléments non nuls par lignes
- 90% de $+/-1$

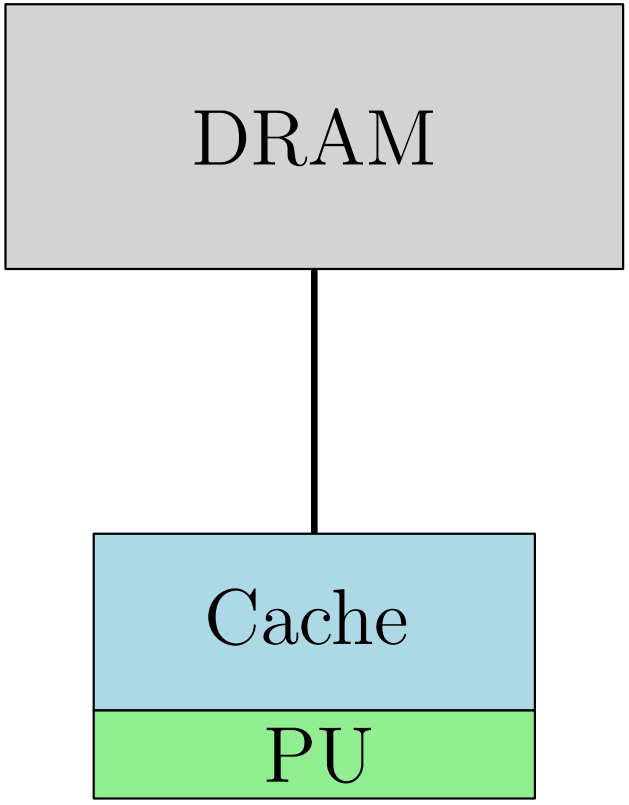
Lors d'un Wiedemann on fait plus de 11000000 spMv

# Format CSR (Compress Sparse Row)

$$A = \begin{bmatrix} 1 & & & & & 7 & & \\ & & 2 & & 5 & & & 8 \\ & & & 4 & & & 6 & \\ 1 & -1 & & & & & & \\ & & & & 9 & & & \\ & & & & & & & \\ & & & 1 & 1 & & & \\ 4 & & & & & 6 & & \\ & & & & & & 7 & 9 \end{bmatrix}$$

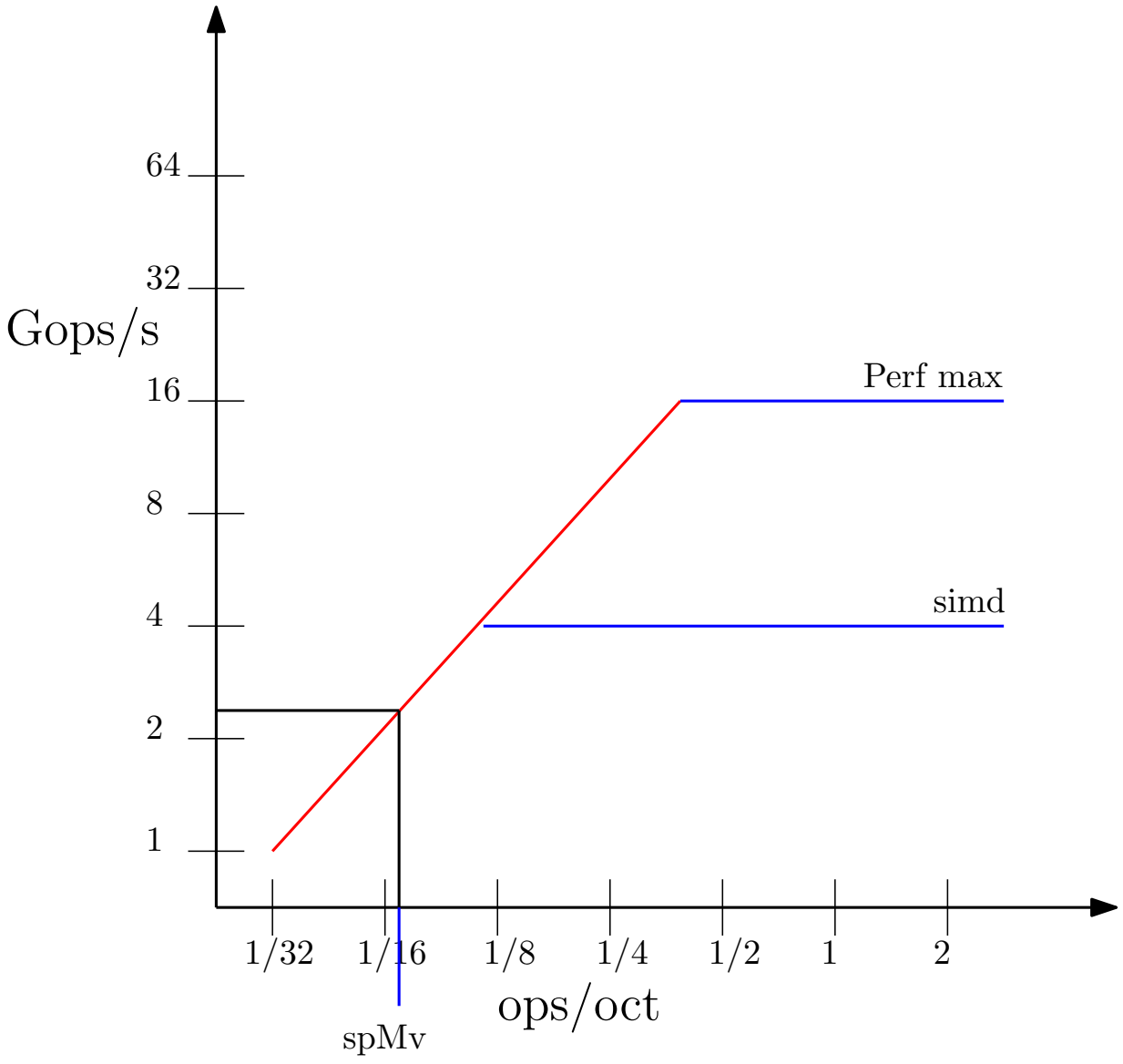


# Modèle de l'ordinateur




- On regarde les transferts depuis la DRAM
- On suppose l'accès au cache infiniment rapide
- On suppose que le cache marche par ligne
- On suppose une opération par cycle
- On regarde le ratio oct/ops

# Modèle pour un cpu intel xeon



# Modèle théorique pour le spMv

$$I_{spMv} = \frac{v_{mat} + v_{in}(\alpha) + v_{out}}{\#operations}$$
$$= \frac{v_{mat} + v_{in}(\alpha) + v_{out}}{1add + 1mul + \gamma mod} oct/ops$$

- $\alpha$ modélise les effets de caches
- $\gamma$ modélise le retardement du modulo

# Modèle théorique pour le spMv

$$I_{spMv} = \frac{v_{mat} + v_{in}(\alpha) + v_{out}}{\#operations}$$
$$= \frac{v_{mat} + v_{in}(\alpha) + v_{out}}{1add + 1mul + \gamma mod} oct/ops$$

- $\alpha$ modélise les effets de caches
- $\gamma$ modélise le retardement du modulo

$$P_{spMv} = \frac{MaxBandWidth}{I_{spMv}} ops/s$$

- $P_{spMv}$ est optimiste

# Modèle théorique pour le format CSR

Listing 1 : Code du spMv avec le format CSR

```
1   for(i = 1 ; i <= rowDim ; ++i)
2     for(j = rows_ptr[i] ; j < rows_ptr[i+1] ; ↩
          ++j)
3       out[i] += val[j]*in[idx[j]];
```

$$I_{CSR} = \frac{v_{mat} + v_{in}(\alpha) + v_{out}}{1add + 1mul + \gamma mod}$$
$$= \frac{1Val + 1Idx + Ptr/Nr + \alpha Elt + 2Elt/Nr}{1add + 1mul + \gamma mod}$$

# Améliorations

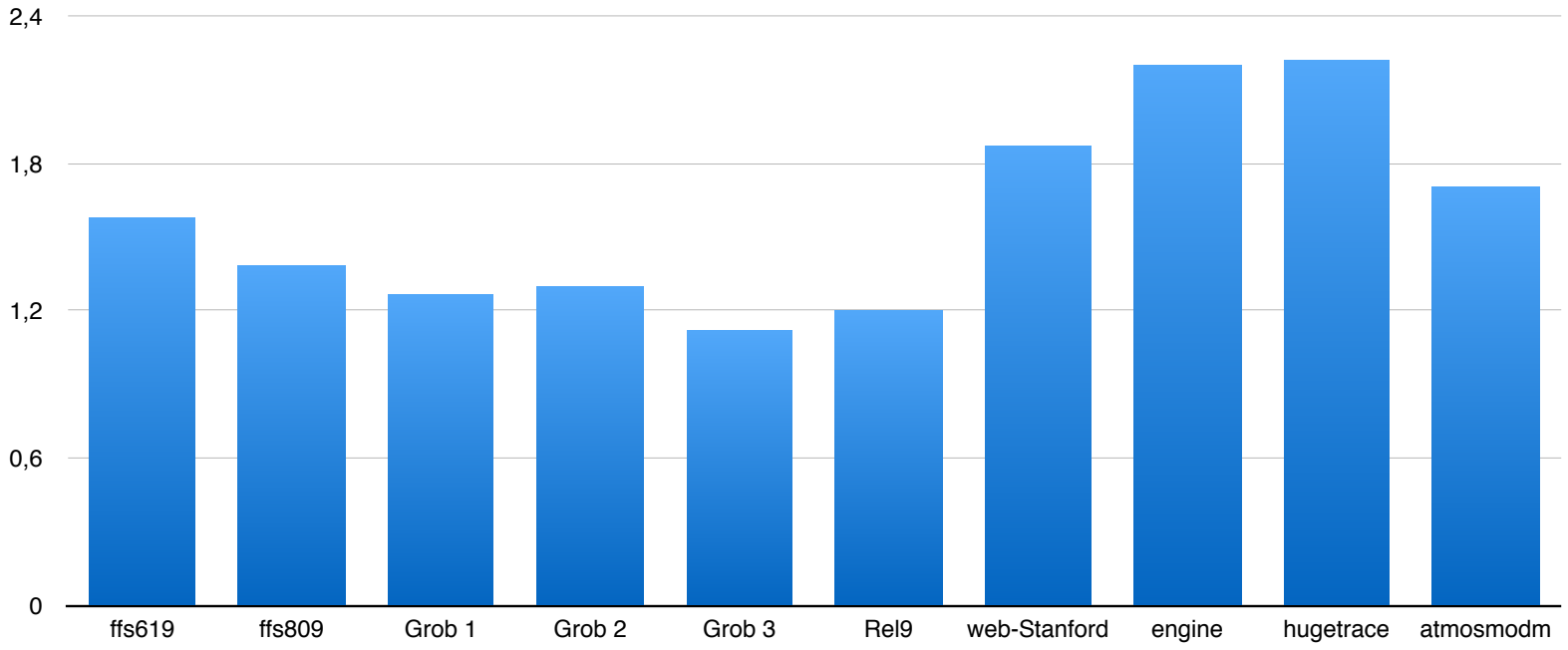
- Bon type de données
- Isoler les constantes
- Software prefetching

# Améliorations

- Bon type de données
- Isoler les constantes
- Software prefetching

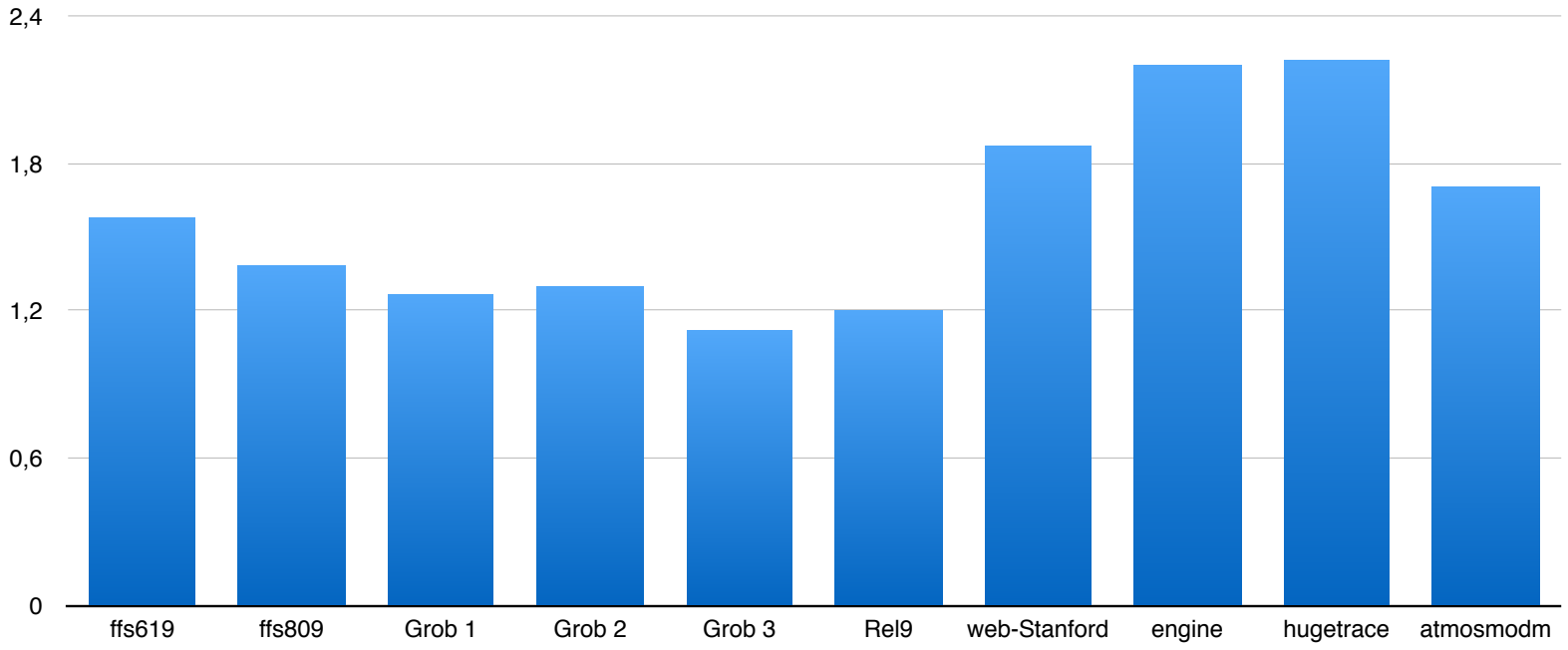
- Multiprécision, choix le plus adapté (gmp, mpfq, rns)
- Multivecteurs, choix du parcours
- Optimisation polyhédrale

# Benchs




Speedup par rapport à LinBox. (mod 2147483659)

# Benchs




Speedup par rapport à LinBox. (mod 2147483659)

FFS809 : $3,6\ s \rightarrow 2,6\ s$, $\approx 11000000\ s$

# Conclusions

- L'opération spMv est fortement ḿemory bound ;
- Il faut compresser l'information.
- Utilisation de GPU, Xeon Phi.
- L'amélioration de la localité spatiale des éléments est NP-Complet (partitionement d'hypergraphe).

Backup slides.

# Partionnement d'hypergraphe

- $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. $\forall n_j \in \mathcal{N}, n_j \subseteq \mathcal{V}$.

# Partionnement d'hypergraphe

- $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. $\forall n_j \in \mathcal{N}, n_j \subseteq \mathcal{V}$.
- Poids $w(v_i), v_i \in \mathcal{V}$. Coût $c(n_j), c_j \in \mathcal{N}$.

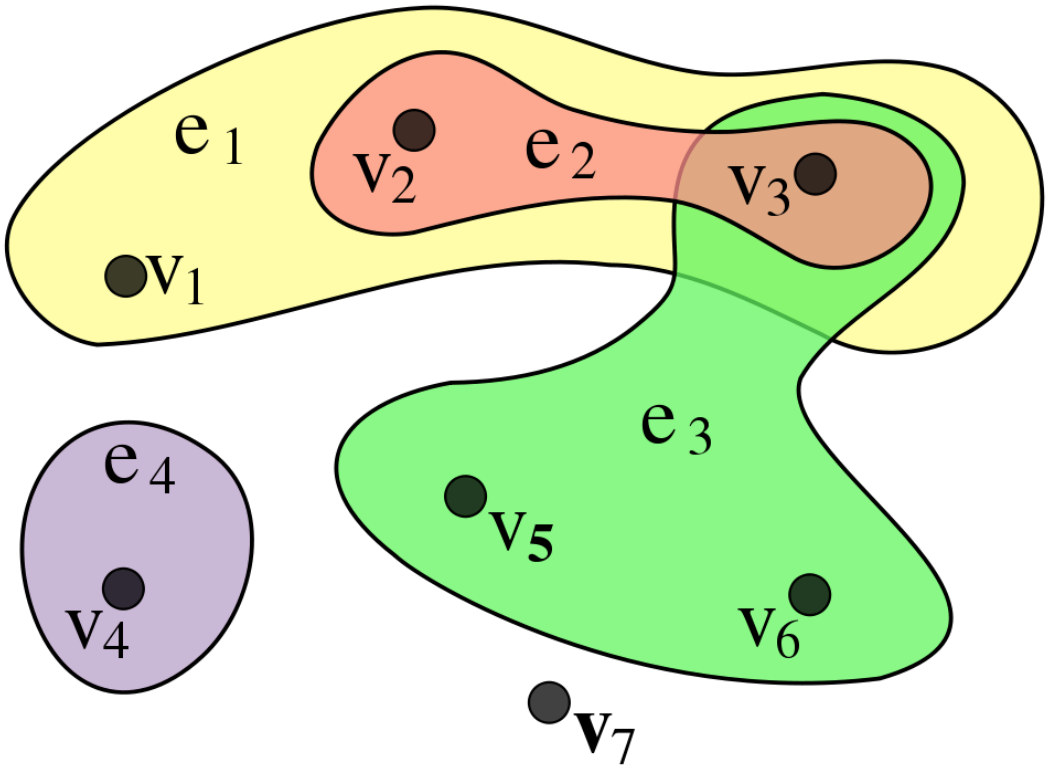
# Partionnement d'hypergraphe

- $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. $\forall n_j \in \mathcal{N}, n_j \subseteq \mathcal{V}$.
- Poids $w(v_i), v_i \in \mathcal{V}$. Coût $c(n_j), c_j \in \mathcal{N}$.
- On veut :
  - $\{\mathcal{V}_1, \cdots, \mathcal{V}_K\}$, tel que $w(\mathcal{V}_j) \leq W_{moy}(1 + \epsilon)$
  - Réduire la taille de la coupe. (Voir tableau)

# Partionnement d'hypergraphe

- $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. $\forall n_j \in \mathcal{N}, n_j \subseteq \mathcal{V}$.
- Poids $w(v_i), v_i \in \mathcal{V}$. Coût $c(n_j), c_j \in \mathcal{N}$.
- On veut :
  - $\{\mathcal{V}_1, \cdots, \mathcal{V}_K\}$, tel que $w(\mathcal{V}_j) \leq W_{moy}(1 + \epsilon)$
  - Réduire la taille de la coupe. (Voir tableau)
- NP-difficile, heuristique multilevel (hMeTiS, PaToH, Mondriaan)

# Exemple hypergraphe



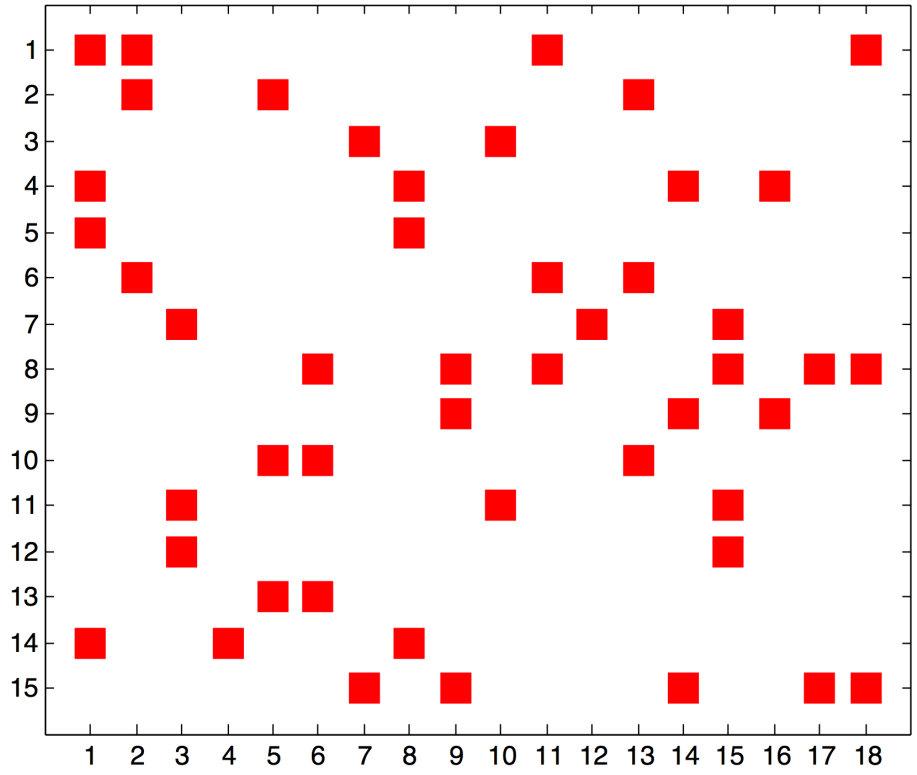
# Matrice creuse vers Hypergraphe

- row-net hypergraph :
  - Pour toutes colonnes $c_i$, il existe $v_i \in \mathcal{V}$.
  - Pour toutes lignes $r_j$, il exite $n_j \in \mathcal{N}$.
  - $\forall v_i \in \mathcal{V} w(v_i) = \#nnz \in c_i$.
- column-net hypergraph : :)
- row-column-net hypergraph :
  - $\forall a_{ij} \neq 0, \exists v_{ij} \in \mathcal{V}$.
  - il existe une hyperarête par colonnes et lignes de la matrice.

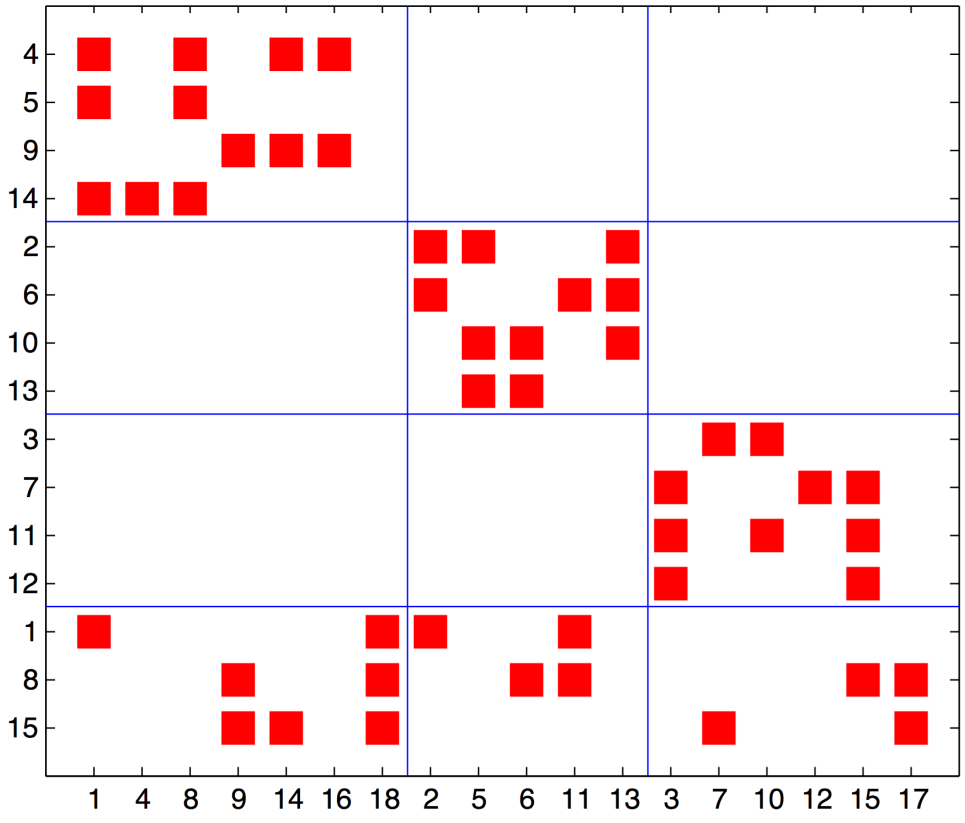
# Du partionnement au réordonnancement

- On suppose un partitionnement d'hypergraphe $\{\mathcal{V}_1, \cdots, \mathcal{V}_K\}$, et $\{\mathcal{N}_1, \cdots, \mathcal{N}_K; \mathcal{N}_{\text{cut}}\}$
- On réindexe les colonnes de $A$ dans l'ordre des $\mathcal{V}_i$, puis les lignes dans l'ordre des $\mathcal{N}_i$, avec $\mathcal{N}_{cut}$ en dernier.

# Effet du partitionnement sur la matrice



# Effet du partitionnement sur la matrice

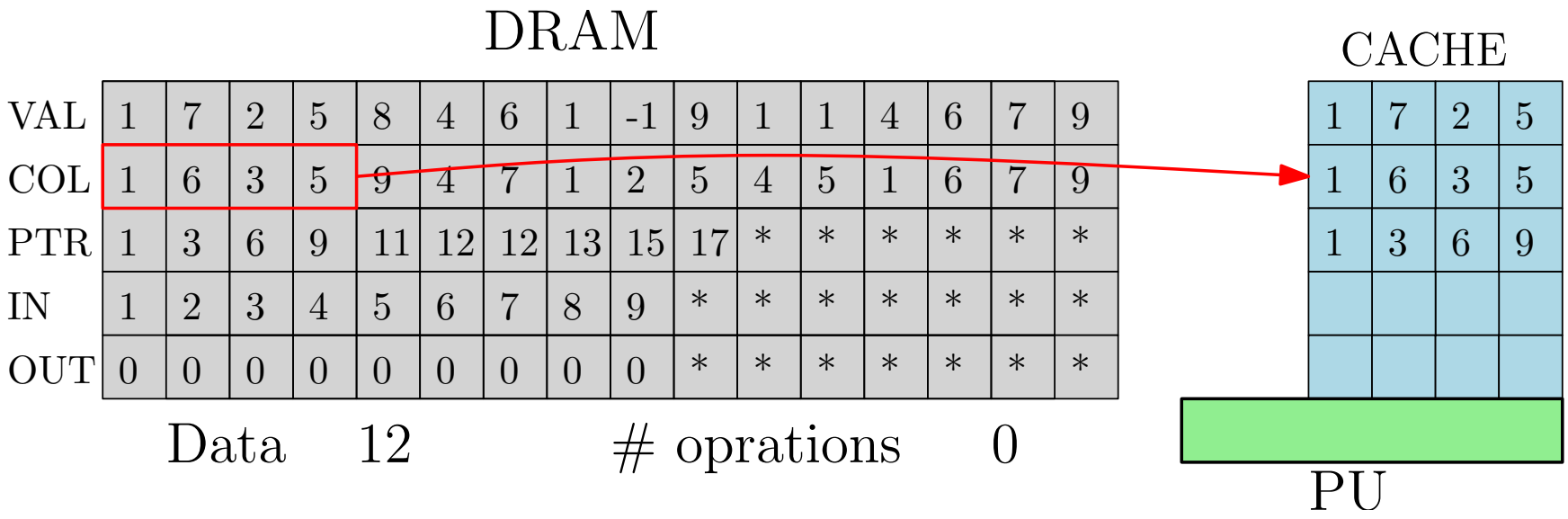


# Exemple d'exécution spMv

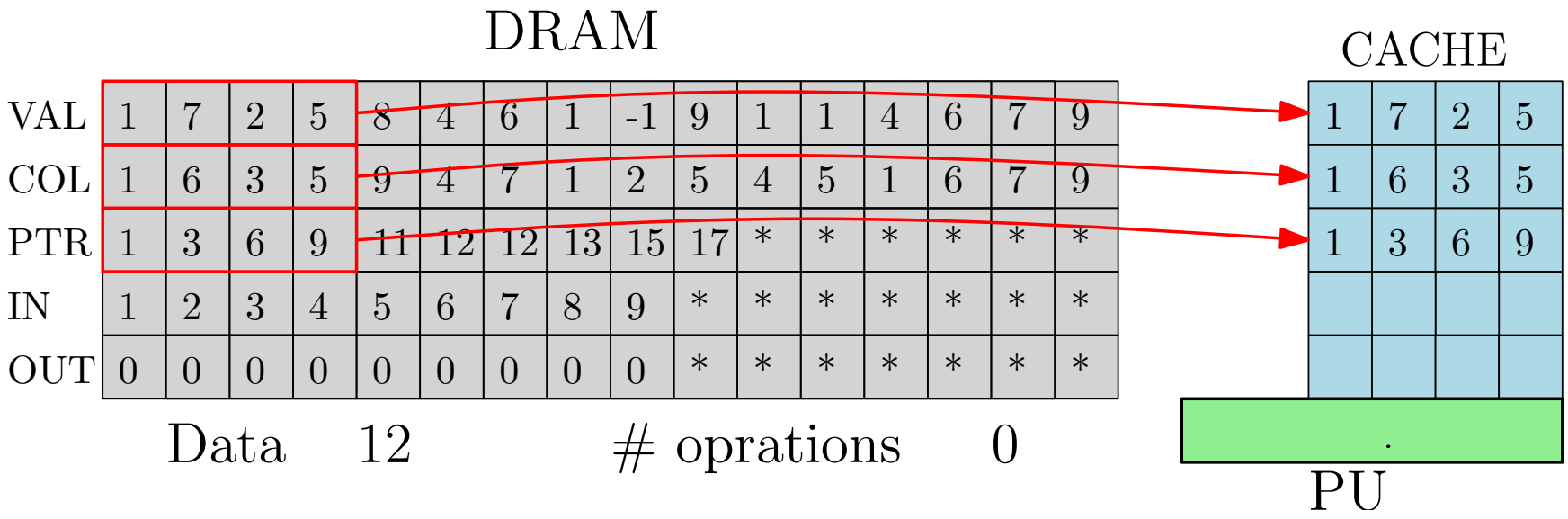
Listing 2 : Code du spMv avec le format CSR

```
1    for(i = 1 ; i <= rowDim ; ++i)
2      for(j = rows_ptr[i] ; j < rows_ptr[i+1] ; ↩
           ++j)
3        out[i] += val[j]*in[col[j]];
```

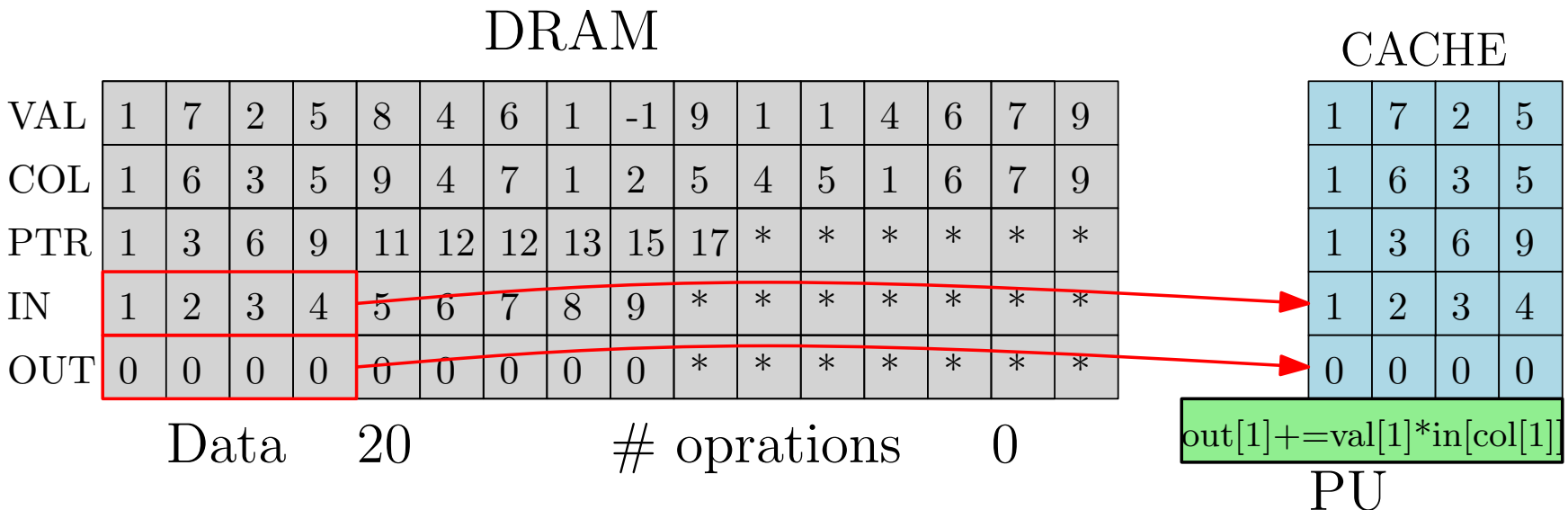
# Exemple d'exécution spMv



# Exemple d'exécution spMv



# Exemple d'exécution spMv



# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 20 # oprations 2

CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 1 | 2 | 3 | 4 |
| 0 | 0 | 0 | 0 |

out[1]+=val[1]*in[col[1]]

PU

# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 20 # oprations 2

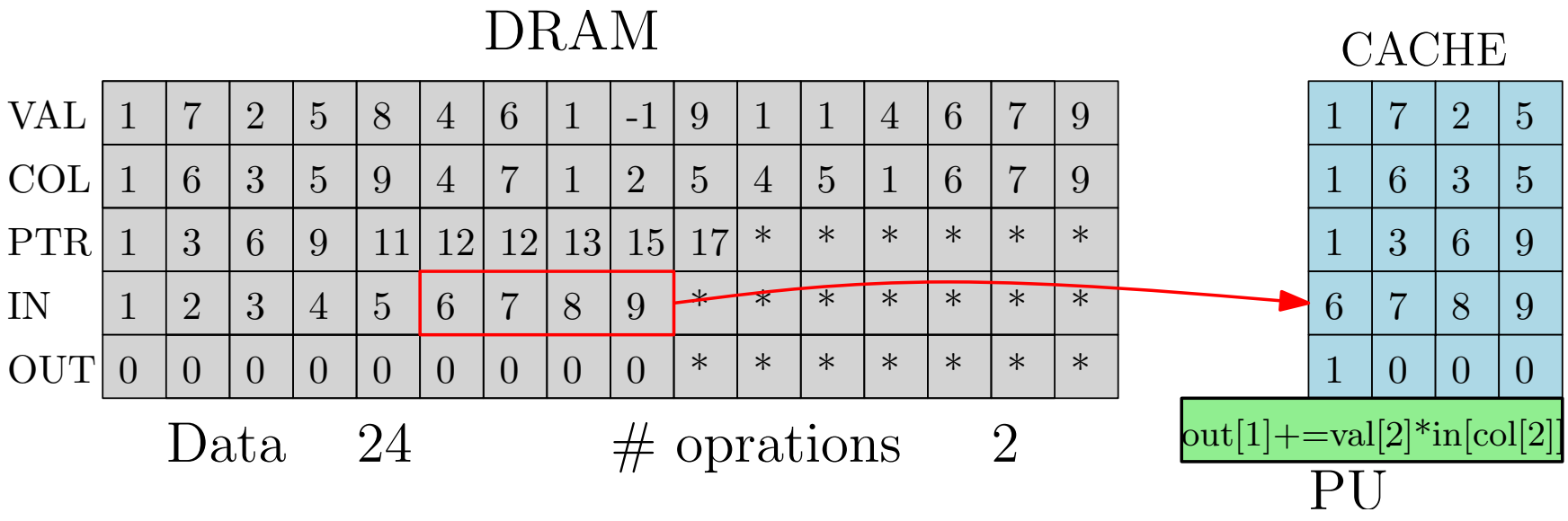
CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 1 | 2 | 3 | 4 |
| 1 | 0 | 0 | 0 |

out[1]+=val[1]*in[col[1]]

PU

# Exemple d'exécution spMv



# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 24 # oprations 4

CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 6 | 7 | 8 | 9 |
| 1 | 0 | 0 | 0 |

out[1]+=val[2]*in[col[2]]

PU

# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 24 # oprations 4

CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 6 | 7 | 8 | 9 |
| 43 | 0 | 0 | 0 |

out[1]+=val[2]*in[col[2]]

PU

# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 24 # oprations 4

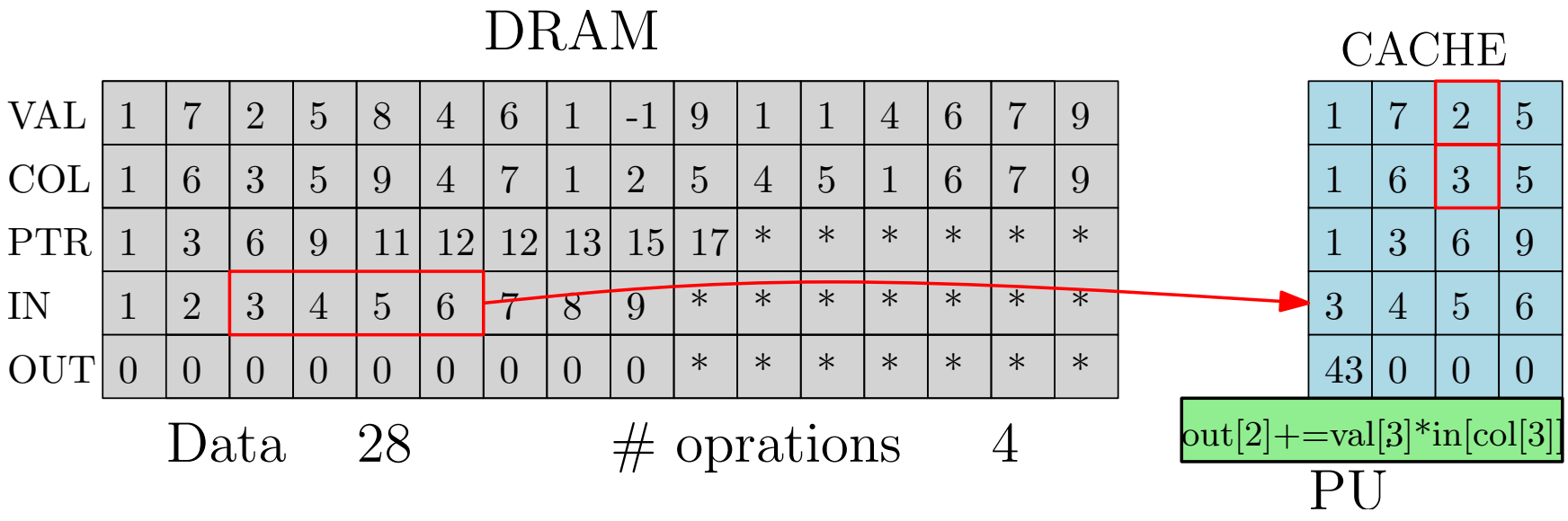
CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 6 | 7 | 8 | 9 |
| 43 | 0 | 0 | 0 |

out[2]+=val[3]*in[col[3]]

PU

# Exemple d'exécution spMv

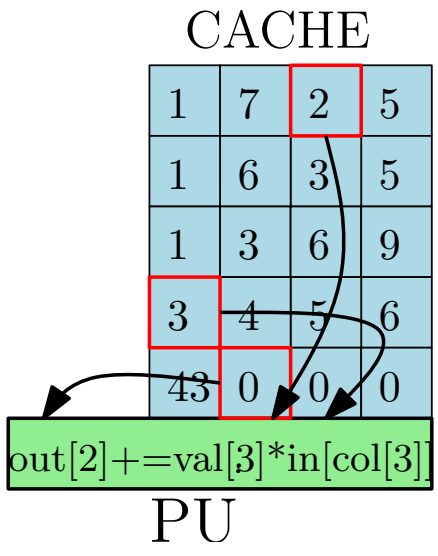


# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 28 # oprations 6



# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 28 # oprations 6

CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 3 | 4 | 5 | 6 |
| 4 | 6 | 0 | 0 |

out[2]+=val[3]*in[col[3]]

PU

# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 28 # oprations 6

CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 3 | 4 | 5 | 6 |
| 43 | 6 | 0 | 0 |

out[2]+=val[4]*in[col[4]]

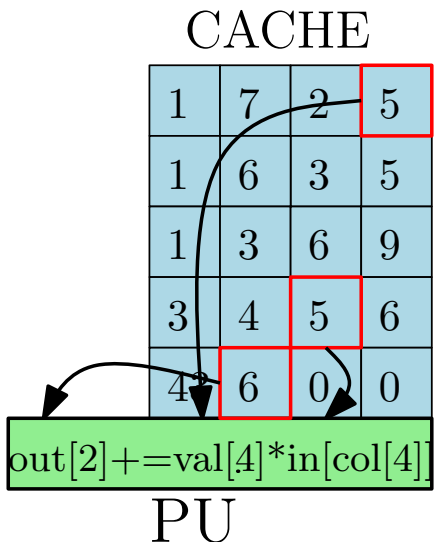
PU

# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 28 # oprations 8



# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 28 # oprations 8

CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 3 | 4 | 5 | 6 |
| 45 | 31 | 0 | 0 |

out[2]+=val[4]*in[col[4]]

PU

# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 28 # oprations 8

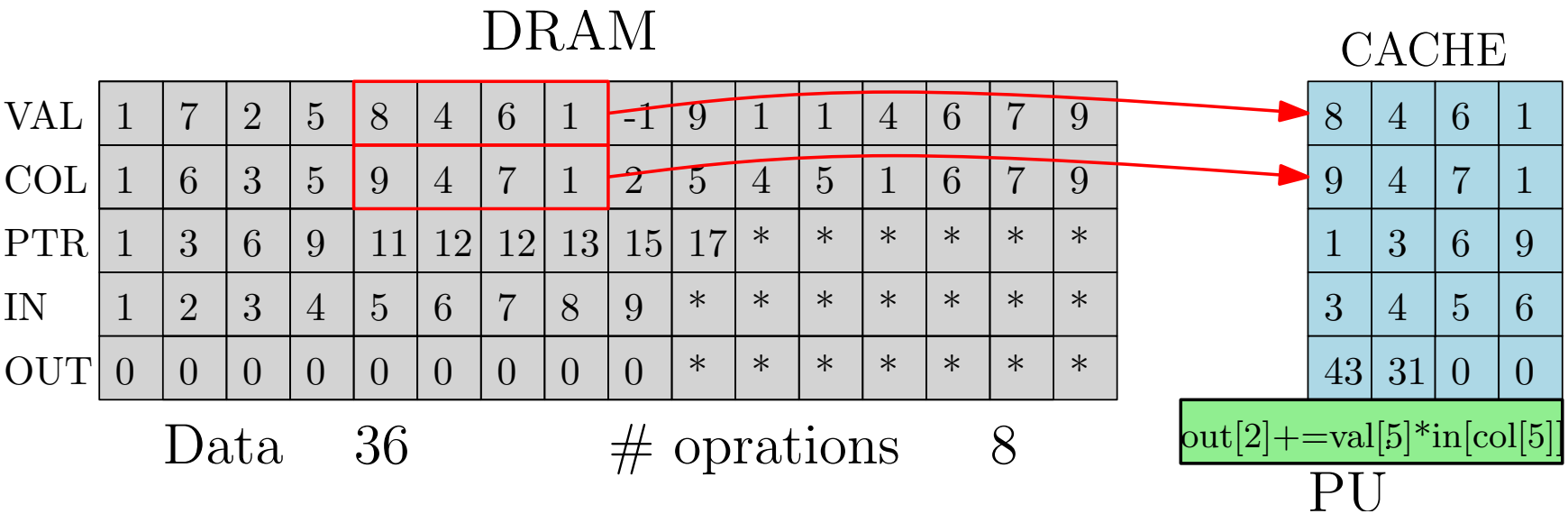
CACHE

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 5 |
| 1 | 6 | 3 | 5 |
| 1 | 3 | 6 | 9 |
| 3 | 4 | 5 | 6 |
| 43 | 31 | 0 | 0 |

out[2]+=val[5]*in[col[5]]

PU

# Exemple d'exécution spMv



# Exemple d'exécution spMv

DRAM

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VAL | 1 | 7 | 2 | 5 | 8 | 4 | 6 | 1 | -1 | 9 | 1 | 1 | 4 | 6 | 7 | 9 |
| COL | 1 | 6 | 3 | 5 | 9 | 4 | 7 | 1 | 2 | 5 | 4 | 5 | 1 | 6 | 7 | 9 |
| PTR | 1 | 3 | 6 | 9 | 11 | 12 | 12 | 13 | 15 | 17 | * | * | * | * | * | * |
| IN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | * | * | * | * | * | * | * |
| OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | * | * | * | * | * |

Data 56 # oprations 18

CACHE

| | | | |
|---|---|---|---|
| -1 | 9 | 1 | 1 |
| 2 | 5 | 4 | 5 |
| 1 | 3 | 6 | 9 |
| 1 | 2 | 3 | 4 |
| 43 | 103 | 58 | -1 |

out[4]+=val[9]*in[col[9]]

PU

# Exemple d'exécution spMv