Efficient Memory Partitioning for Parallel Data Access via Data Reuse

Jincheng Su[1], Fan Yan[1], Xuan Zeng[1] and Dian Zhou[12]

[1]Fudan University, Shanghai, China

[2]University of Texas at Dallas, USA

Feb 22, 2016

BACKGROUND

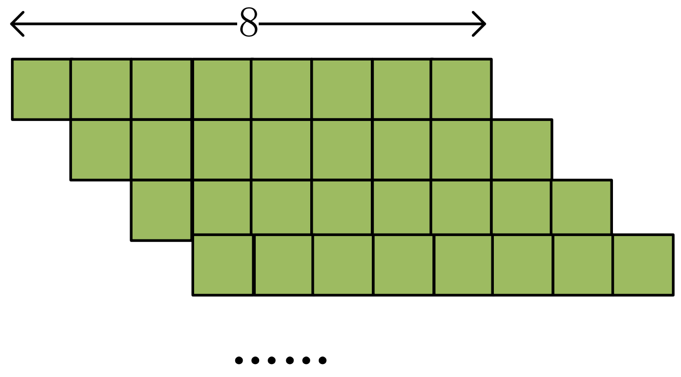
- Loop pipelining and parallel access
- Solutions for parallel access
- Memory partitioning problem
- State-of-the-art partition scheme

Loop pipelining

Given a set of references to memory in a loop nest, in order to enable loop pipelining, how to map them to on-chip memory so as to efficiently support parallel access.

```
1:  define $A[1920][1080]$, $Z[1920][1080]$
2:   for ($i_0 = 0$; $i_0 < 1918$; $i_0$++){
3:       for($i_1 = 0$; $i_1 < 1078$; $i_1$++){
4:         $Z[i_0+1][i_1+1] = foo(A[i_0][i_1], A[i_0][i_1+1], A[i_0][i_1+2], A[i_0+1][i_1],$
           $A[i_0+1][i_1+2], A[i_0+2][i_1], A[i_0+2][i_1+1], A[i_0+2][i_1+2])$
         }
     }
```

Loop pipelining

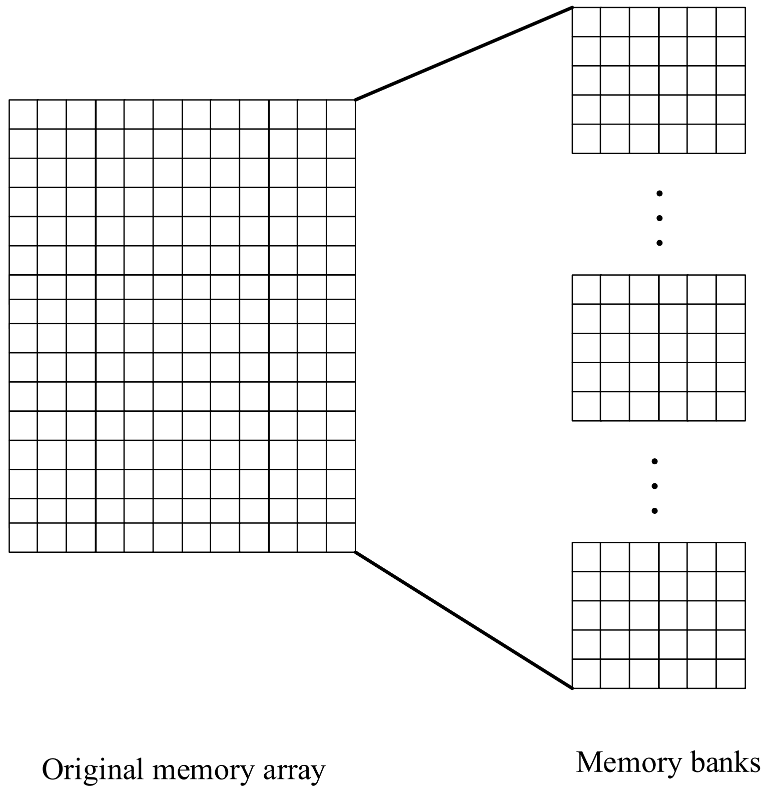


Solutions for parallel access

- Partition the array A to m memory banks given m references

Memory Partitioning

Hopefully no extra storage overhead!



Memory Partitioning Problem

Data space: $\mathbb{M}$

```
define A[1920][1080], Z[1920][1
  for ($i_0 = 0; i_0 < 1918; i_0$++){
      for($i_1 = 0; i_1 < 1078; i_1$++){
        $Z[i_0+1][i_1+1] = foo( A[i_0][i_1$
        $A[i_0+1][i_1+2], A[i_0+2][i_1], A[i_0+2][i_1+1], A[i_0+2][i_1+2])$
      }
  }
```

Iteration space:

$$\mathbb{D} = \left\{ \boldsymbol{i} \,\middle|\, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \le \boldsymbol{i} < \begin{pmatrix} 1918 \\ 1078 \end{pmatrix} \right\}$$

$\boldsymbol{i}$ is called ***iteration vector***

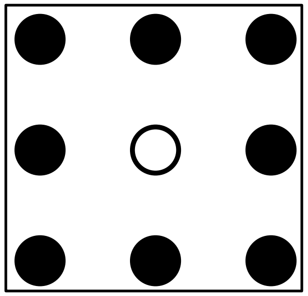
Affine memory reference

$$\boldsymbol{x} = A\boldsymbol{i} + \boldsymbol{c} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i_0 \\ i_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Access Pattern

sharing the same coefficient matrix A

$$P_A = \begin{Bmatrix} (0,0)^T & (0,1)^T & (0,2)^T \\ (1,0)^T & & (1,2)^T \\ (2,0)^T & (2,1)^T & (2,2)^T \end{Bmatrix}$$



Memory Partitioning Problem

***Bank mapping problem*:**

Minimize N

Subject To

$$\forall \vec{x}_j, \vec{x}_k \in M, \vec{x}_j \neq \vec{x}_k, B(\vec{x}_j) \neq B(\vec{x}_k) || F(\vec{x}_j) \neq F(\vec{x}_k),$$
$$\forall \vec{i} \in D, \forall \vec{p}_j, \vec{p}_k \in P, \vec{p}_j \neq \vec{p}_k, B(\vec{p}_j) \neq B(\vec{p}_k).$$

***Intra-bank offsetting problem*:**

For references mapped to the same bank, give each of them a unique offset, such that there is no access conflict and the storage overhead is minimum.

Some state-of-the-art partition schemes

For multi-dimensional memory arrays

- **LTB** (Linear Transformation Based memory partitioning)

  Bank index: $B(x) = (\alpha \cdot x)\%N$

  Intra-bank offset: padding method

- **GMP** (Generalized Memory Partitioning)

  Bank index: $B(x) = \frac{\alpha \cdot x}{B}\%N$

  Intra-bank offset: improved padding method

  Polyhedral model to find optimal α

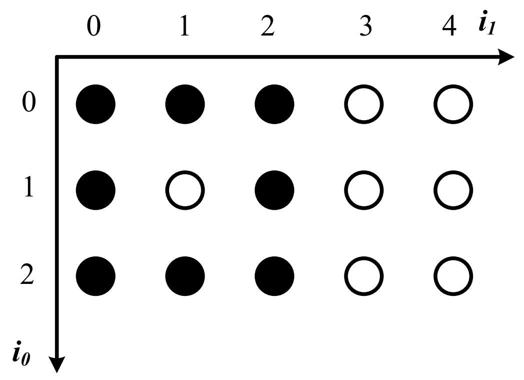
- **EMP** (Efficient Memory Partitioning)

  Bank index: $B(x) = (\alpha \cdot x)\%N$

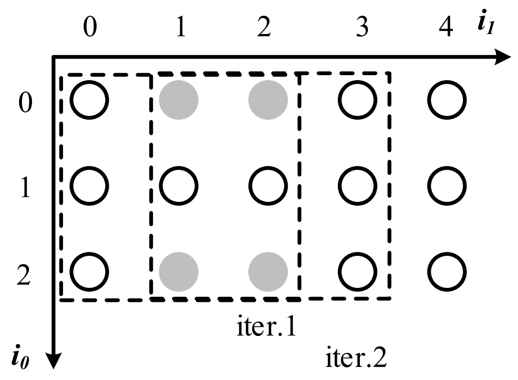
  Intra-bank offset: improved padding method

  Fast heuristic algorithm to construct a valid α

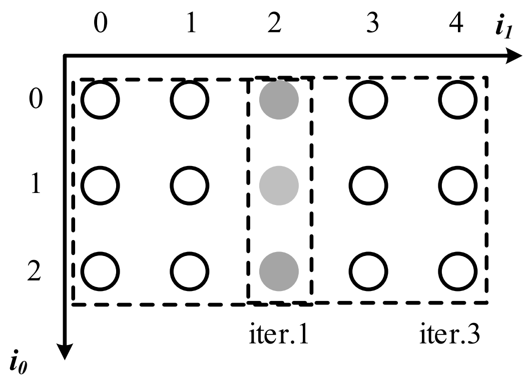
Utilize the opportunities of data reuse



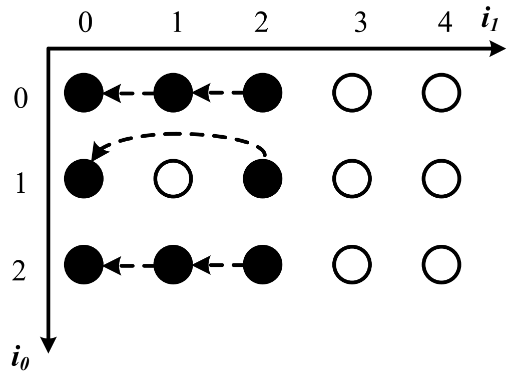
(a)



(b)

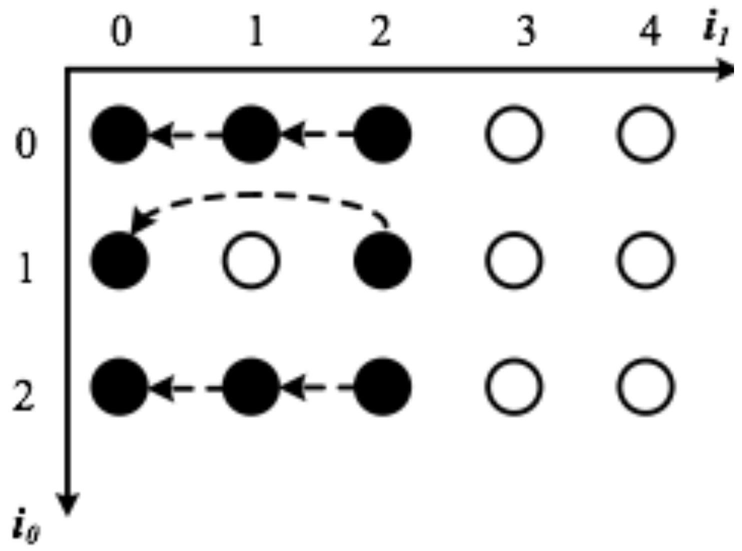


(c)



(d)

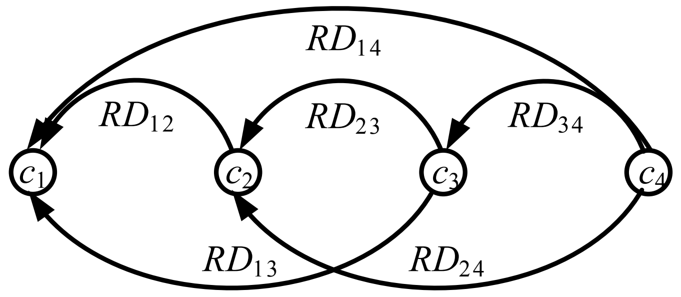
The proposed memory partitioning algorithm

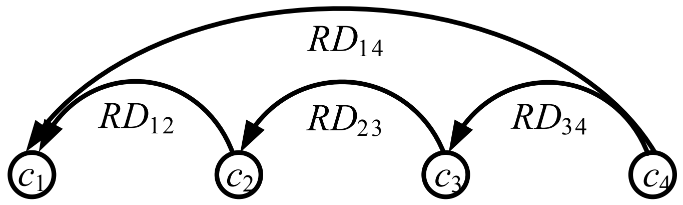


The key idea

Use the on-chip registers to cache the reusable data.

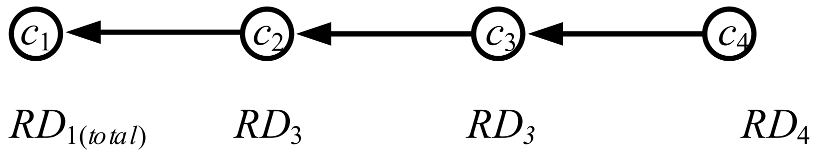
Data reuse chains



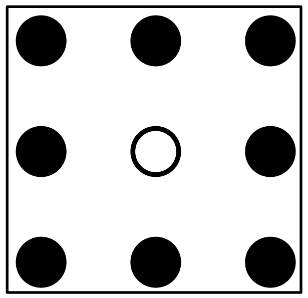


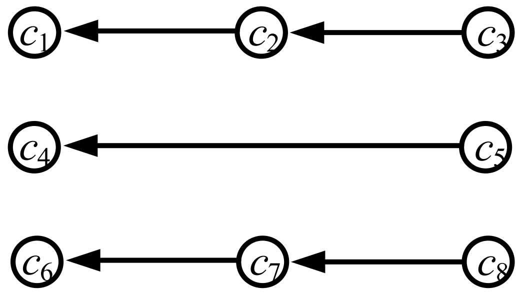
(b)

(a) Data reuse graph

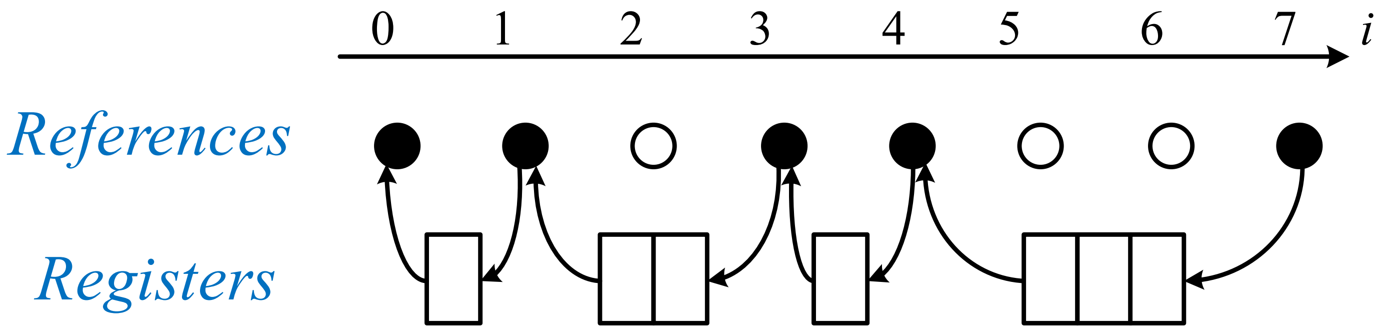


(c) Data reuse chain

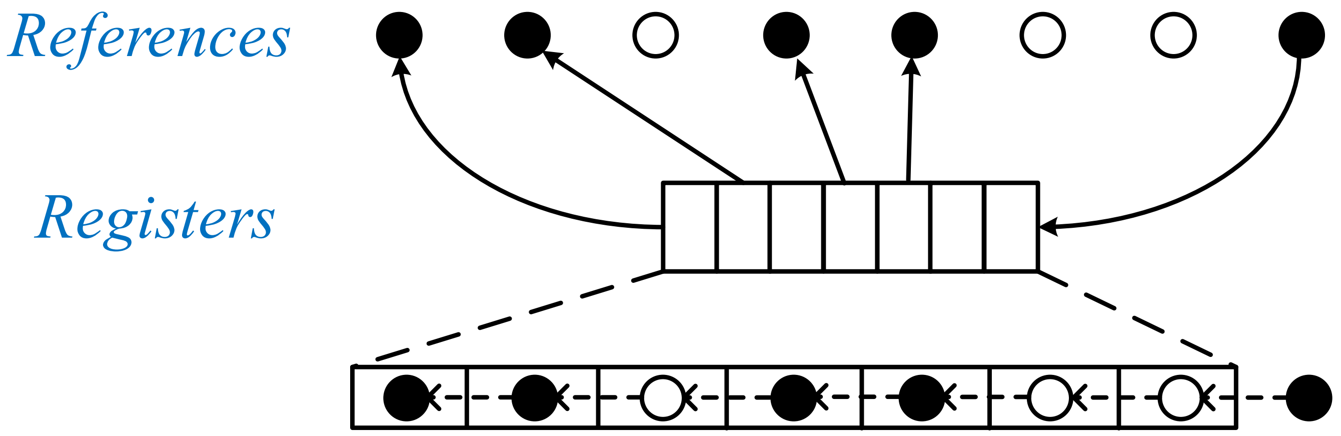




Implementing Data reuse chains



(a)



(b)

Constructing data reuse chains

Move**:** $\boldsymbol{\delta} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

```
for ($i_0 = 0$; $i_0 < 1918$; $i_0$++){
    for($i_1 = 0$; $i_1 < 1078$; $i_1$++){
        …
…
```

Move**:** $\boldsymbol{\delta} = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$

```
for ($i_0 = 0$; $i_0 < N_0$; $i_0$++){
    for($i_1 = 0$, $i_2 = 0$; $i_1 < N_1$&& $i_2 < N_2$; $i_1$++, $i_2 = i_2 + 2$){
        …
…
```

Reuse theorem:

Denote $\boldsymbol{\delta}$ *a move for iteration vector* $\boldsymbol{i}$ *of a loop nest. Given two different offsets* $\mathbf{c_j}$ *and* $\mathbf{c_k}$, $\mathbf{c_j} \neq \mathbf{c_k}$ *in an access pattern sharing the same coefficient matrix A, if* $\mathbf{c_j} - \mathbf{c_k} = \lambda A \boldsymbol{\delta}$ *where* λ *is a non-negative integer, then the data element referenced by* $\mathbf{c_j}$ *can be reused as the data referenced by* $\mathbf{c_k}$ *after* λ *iterations.*

Constructing data reuse chains

$$P_A = \left\{ \begin{matrix} (0,\ 0)^{\mathrm{T}} & (0,\ 1)^{\mathrm{T}} & (0,\ 2)^{\mathrm{T}} \\ (1,\ 0)^{\mathrm{T}} & & (1,\ 2)^{\mathrm{T}} \\ (2,\ 0)^{\mathrm{T}} & (2,\ 1)^{\mathrm{T}} & (2,\ 2)^{\mathrm{T}} \end{matrix} \right\} \qquad A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \boldsymbol{\delta} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\begin{tikzcd}
(0,\ 0)^{\mathrm{T}} & (0,\ 1)^{\mathrm{T}} \arrow[l, "\lambda = 1"'] & (0,\ 2)^{\mathrm{T}} \arrow[l, "\lambda = 1"'] \\
(1,\ 0)^{\mathrm{T}} & & (1,\ 2)^{\mathrm{T}} \arrow[ll] \\
(2,\ 0)^{\mathrm{T}} & (2,\ 1)^{\mathrm{T}} \arrow[l] & (2,\ 2)^{\mathrm{T}} \arrow[l]
\end{tikzcd}$$

The new pattern which will be delivered to the memory partitioning algorithm

Bank Mapping

Minimize N

Subject To

$$\forall \vec{x}_j, \vec{x}_k \in M, \vec{x}_j \neq \vec{x}_k, B(\vec{x}_j) \neq B(\vec{x}_k) || F(\vec{x}_j) \neq F(\vec{x}_k),$$

$$\forall \vec{i} \in D, \forall \vec{p}_j, \vec{p}_k \in P, \vec{p}_j \neq \vec{p}_k, B(\vec{p}_j) \neq B(\vec{p}_k).$$

Design choice: $B(\boldsymbol{x}) = (\boldsymbol{\alpha} \cdot \boldsymbol{x})\%N$

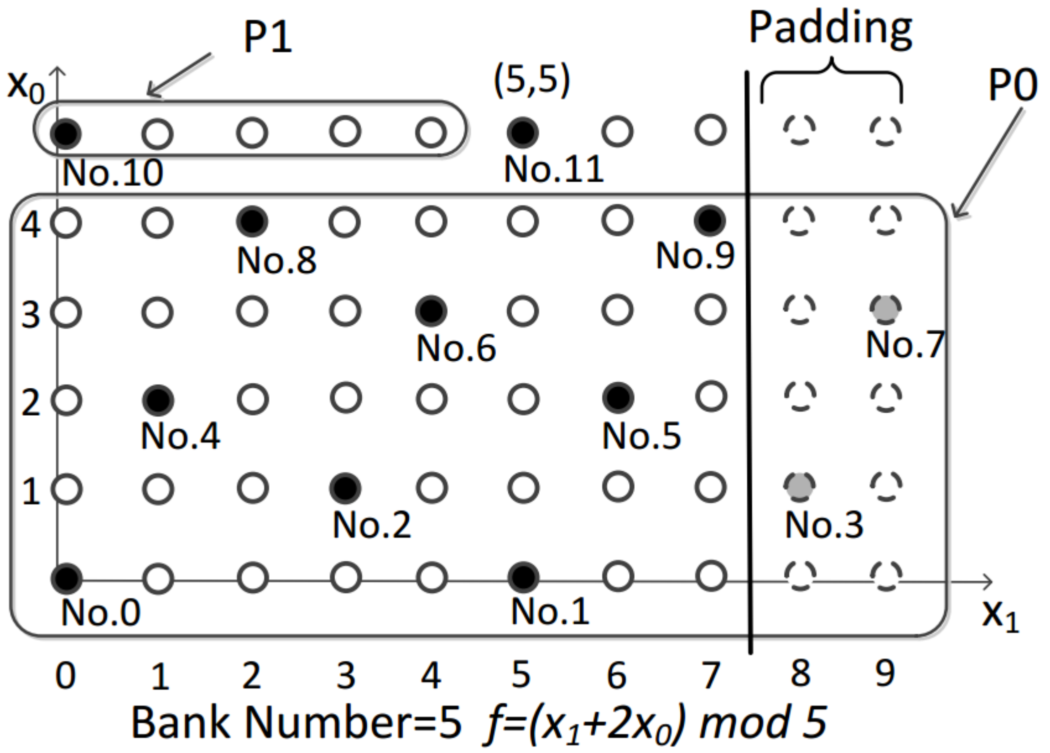
Partition vector

Partition factor

By assumptions 3 and 4, the corollary holds:

Corollary 1. *Two references* $\vec{x}_j = A \cdot \vec{j} + \vec{c}_j$ *and* $\vec{x}_k = A \cdot \vec{k} + \vec{c}_k$ *are mapped to the same bank by the mapping function* $B(\vec{x}) = (\vec{\alpha} \cdot \vec{x})\%N$*, if and only if* $\vec{\alpha} \cdot (\vec{c}_j - \vec{c}_k)$ *is exactly divisible by* N

Padding method



(this picture is selected from [1])

Storage overhead $(N \times \lceil w_{d-1}/N \rceil - w_{d-1}) \times \prod_{j=0}^{d-2} w_j$

[1] *Y. Wang, P. Li, and J. Cong, "Theory and algorithm for generalized memory partitioning in high-level synthesis," in Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2014.*

Revised padding method

The case where some component of the partition vector $\boldsymbol{\alpha}$ are zero.

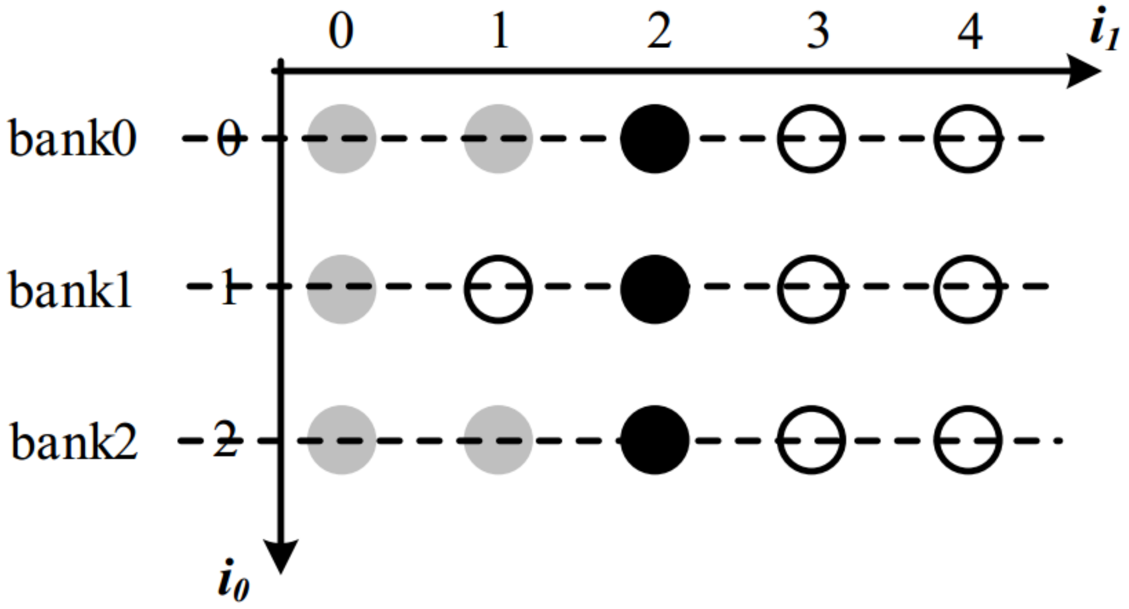
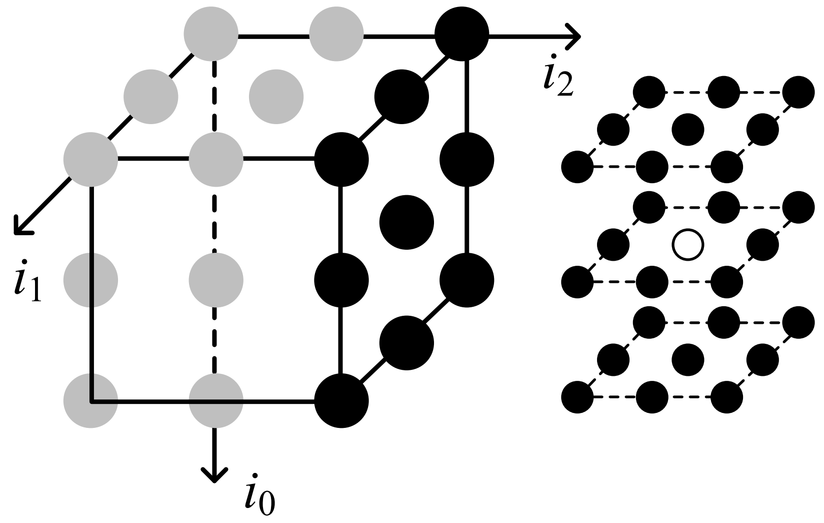
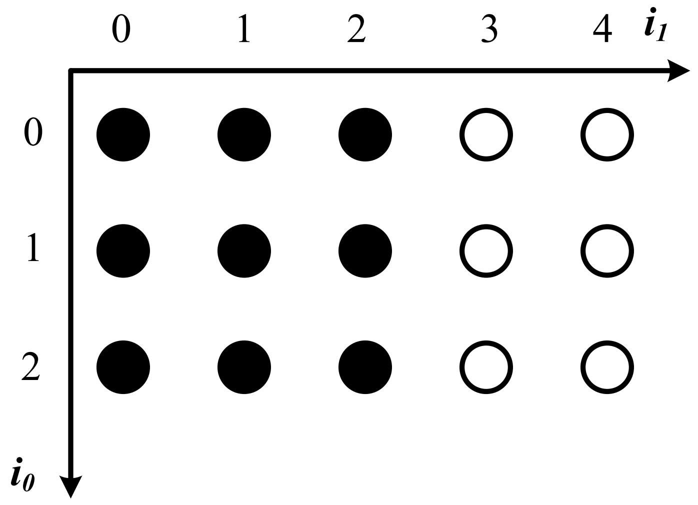


Figure 5: Mapping result for the case that $\vec{\alpha} = (1, 0)$.

Revised padding method





A 3-dimensional access pattern.

The partition vector $\boldsymbol{\alpha}$ would be $(1, 3, 0)^T$.

Partition factor N would be 9.

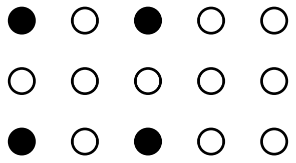
Revised padding method

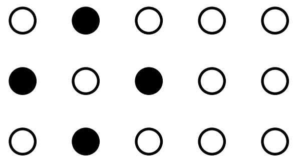
- Re-arrange $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{d-1})$, such that $\alpha_0, \alpha_1, \ldots, \alpha_{k-1} \neq 0$, $\alpha_k, \alpha_{k+1}, \ldots, \alpha_{d-1} = 0$.
- Re-arrange $\boldsymbol{x} = (x_0, x_1, \ldots, x_{d-1})$ correspondingly.
- Apply padding to the k-1 dimension.

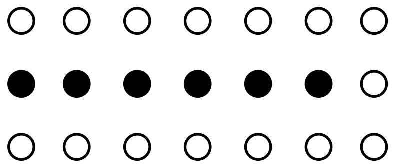
***Storage overhead*:** $$(\lceil \frac{w_{k-1}}{N} \rceil \times N - w_{k-1}) \times \prod_{j=0}^{k-2} \prod_{j=k}^{d-1} w_j .$$

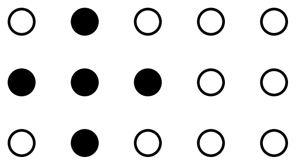
If $k = 1$, no padding is needed, which, by our memory partitioning scheme, is a common case!

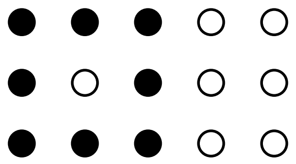
Benchmarks

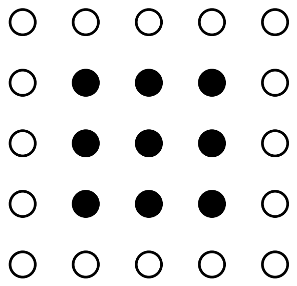

(a).BICUBIC

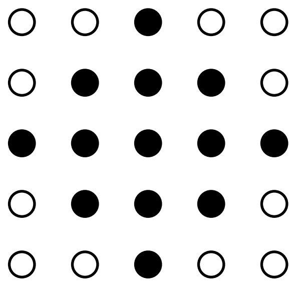

(b).DENOISE

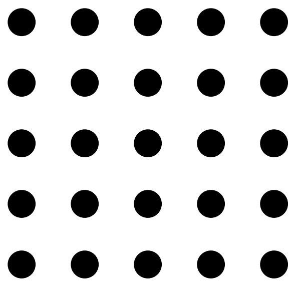

(c).MOTION_LH


(d).DECONV

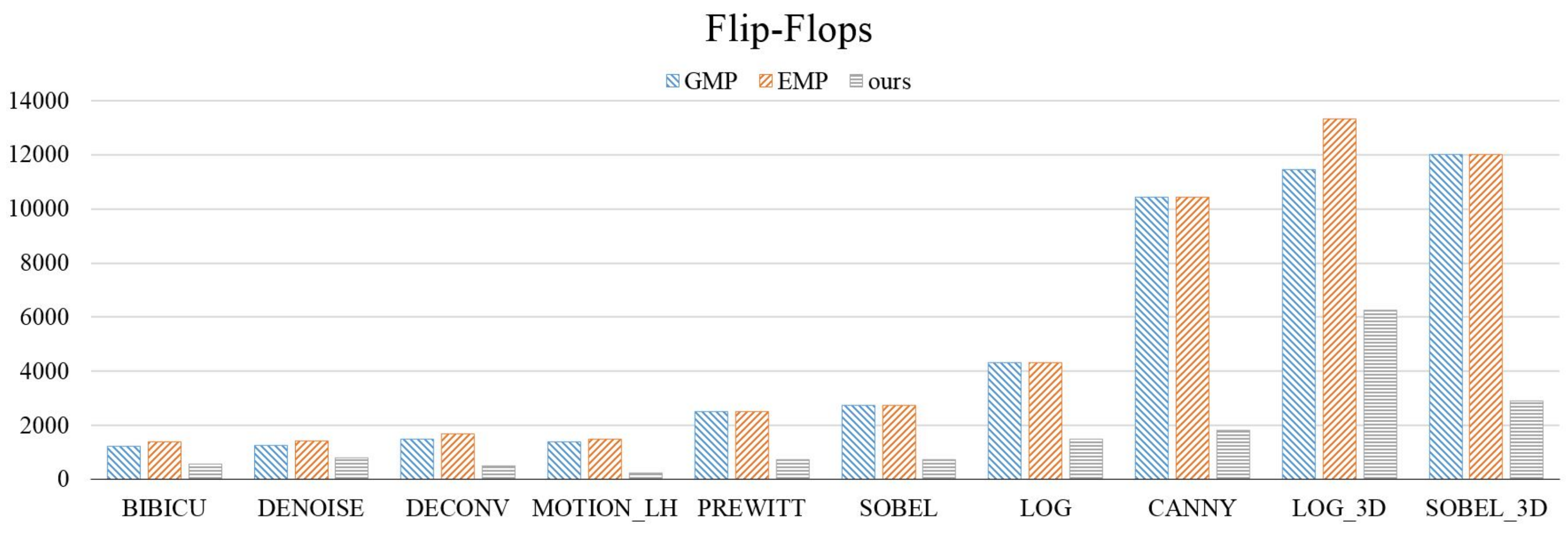

(e).PREWITT

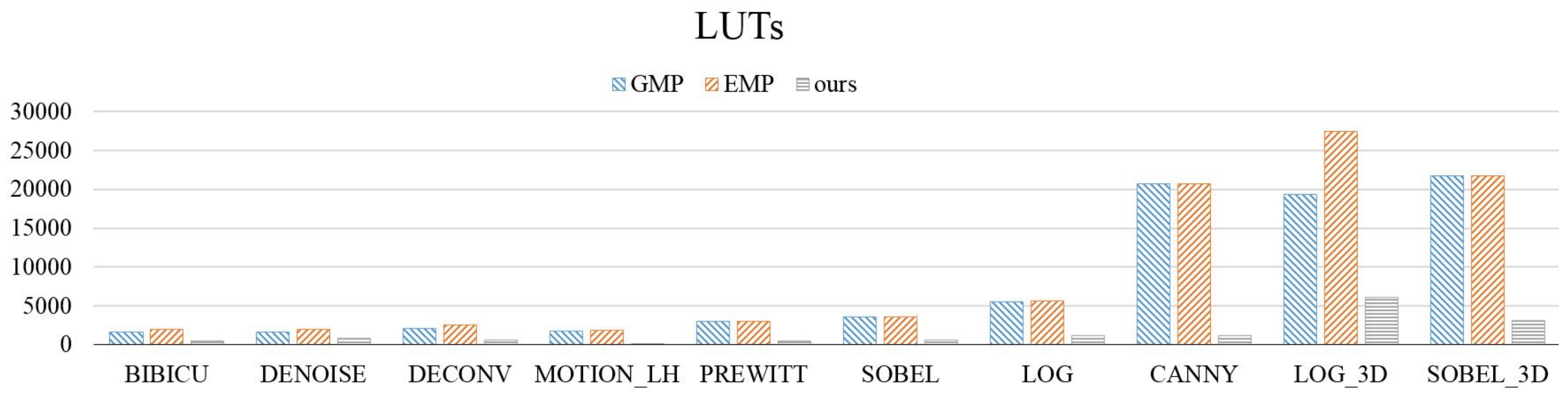

(f).SOBEL


(g).LOG


(h).CANNY

Experimental result & comparison





Experimental result & comparison

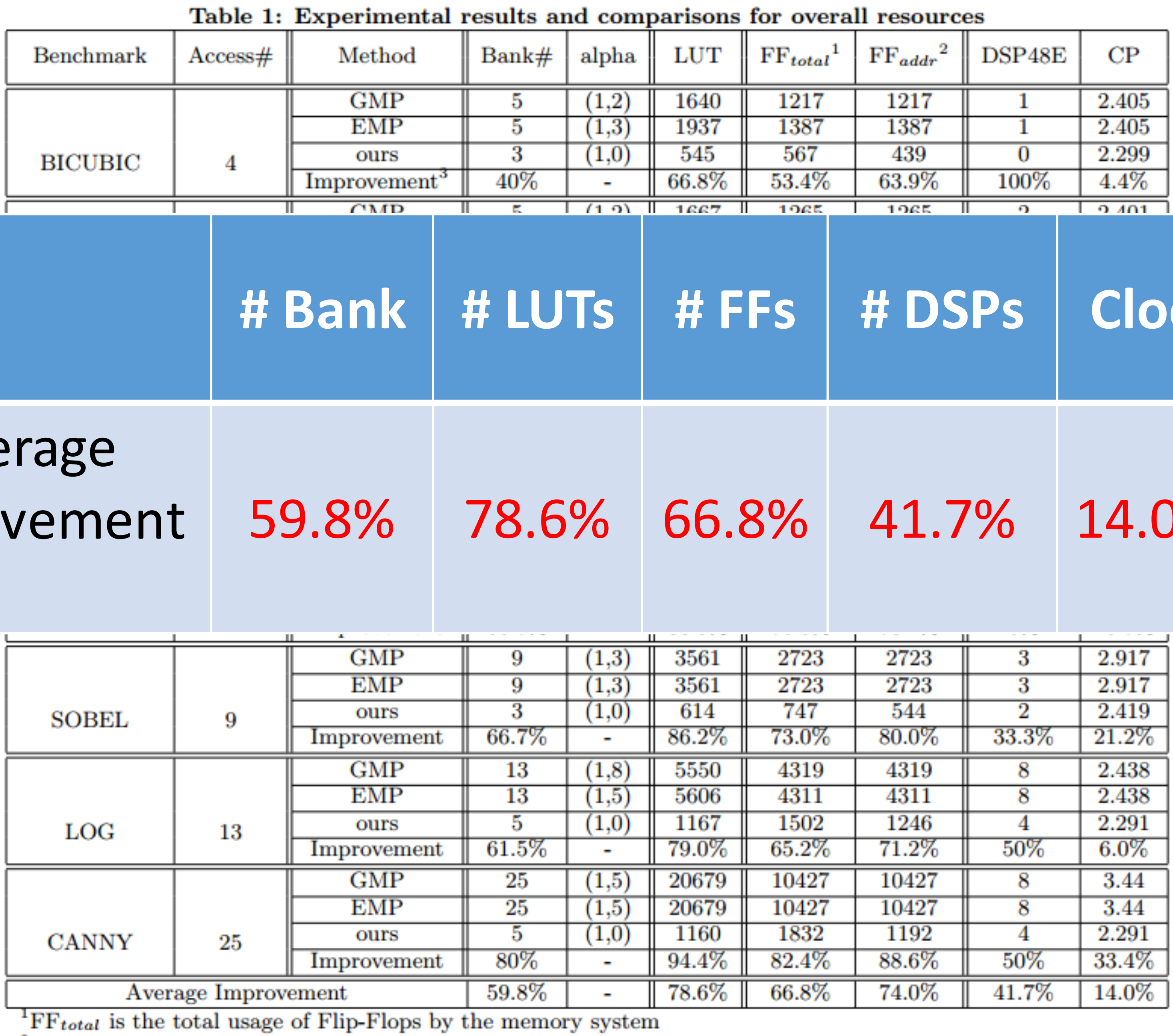


Table 1: Experimental results and comparisons for overall resources

Benchmark	Access#	Method	Bank#	alpha	LUT	FF_{total}[1]	FF_{addr}[2]	DSP48E	CP
BICUBIC	4	GMP	5	(1,2)	1640	1217	1217	1	2.405
		EMP	5	(1,3)	1937	1387	1387	1	2.405
		ours	3	(1,0)	545	567	439	0	2.299
		Improvement[3]	40%	-	66.8%	53.4%	63.9%	100%	4.4%
		GMP	5	(1,2)	1667	1265	1265	2	2.401
SOBEL	9	GMP	9	(1,3)	3561	2723	2723	3	2.917
		EMP	9	(1,3)	3561	2723	2723	3	2.917
		ours	3	(1,0)	614	747	544	2	2.419
		Improvement	66.7%	-	86.2%	73.0%	80.0%	33.3%	21.2%
LOG	13	GMP	13	(1,8)	5550	4319	4319	8	2.438
		EMP	13	(1,5)	5606	4311	4311	8	2.438
		ours	5	(1,0)	1167	1502	1246	4	2.291
		Improvement	61.5%	-	79.0%	65.2%	71.2%	50%	6.0%
CANNY	25	GMP	25	(1,5)	20679	10427	10427	8	3.44
		EMP	25	(1,5)	20679	10427	10427	8	3.44
		ours	5	(1,0)	1160	1832	1192	4	2.291
		Improvement	80%	-	94.4%	82.4%	88.6%	50%	33.4%
Average Improvement			59.8%	-	78.6%	66.8%	74.0%	41.7%	14.0%

[1] FF_{total} is the total usage of Flip-Flops by the memory system

	# Bank	# LUTs	# FFs	# DSPs	Clock
Average improvement	59.8%	78.6%	66.8%	41.7%	14.0%

Experimental result & comparison

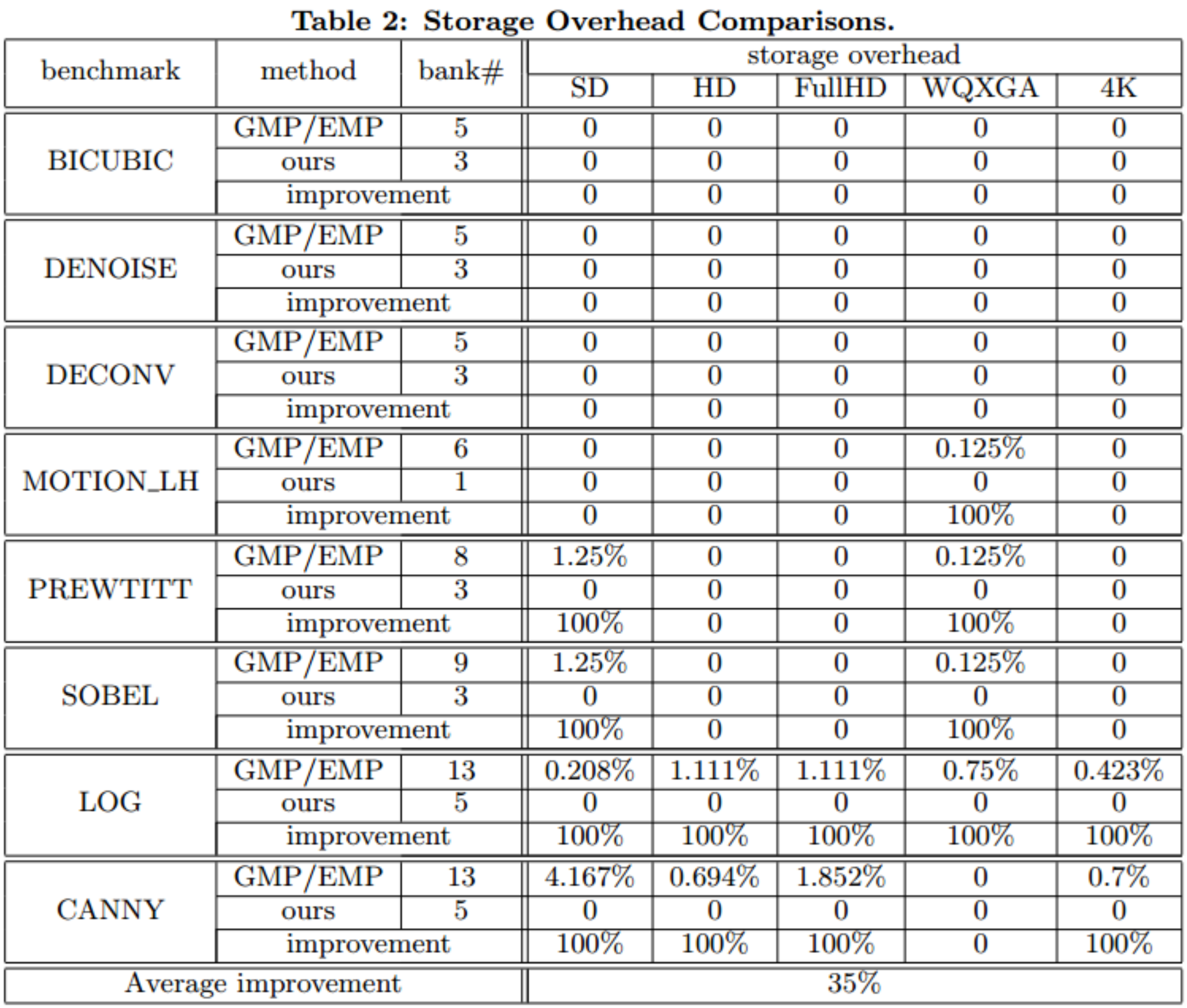


Table 2: Storage Overhead Comparisons.

benchmark	method	bank#	storage overhead				
			SD	HD	FullHD	WQXGA	4K
BICUBIC	GMP/EMP	5	0	0	0	0	0
	ours	3	0	0	0	0	0
	improvement		0	0	0	0	0
DENOISE	GMP/EMP	5	0	0	0	0	0
	ours	3	0	0	0	0	0
	improvement		0	0	0	0	0
DECONV	GMP/EMP	5	0	0	0	0	0
	ours	3	0	0	0	0	0
	improvement		0	0	0	0	0
MOTION_LH	GMP/EMP	6	0	0	0	0.125%	0
	ours	1	0	0	0	0	0
	improvement		0	0	0	100%	0
PREWTITT	GMP/EMP	8	1.25%	0	0	0.125%	0
	ours	3	0	0	0	0	0
	improvement		100%	0	0	100%	0
SOBEL	GMP/EMP	9	1.25%	0	0	0.125%	0
	ours	3	0	0	0	0	0
	improvement		100%	0	0	100%	0
LOG	GMP/EMP	13	0.208%	1.111%	1.111%	0.75%	0.423%
	ours	5	0	0	0	0	0
	improvement		100%	100%	100%	100%	100%
CANNY	GMP/EMP	13	4.167%	0.694%	1.852%	0	0.7%
	ours	5	0	0	0	0	0
	improvement		100%	100%	100%	0	100%
Average improvement			35%				

Conclusion

Contributions:

- Propose to cache the reusable data by on-chip registers of FPGA
- Propose a new data reuse strategy
- Revise the padding method to generate intra-bank offset more efficiently.

Results:

Compared with the GMP partition scheme,

- ours can reduce the required number of banks by 59.8% on average.
- The number of LUTs is reduced by 78.6%, Flip-Flops by 66.8%, DSP48Es by 41.7% on average.
- While the performance is improved slightly.

Thanks

Q&A