Optimal Loop Tiling for Minimizing Write Operations on NVMs with Complete Memory Latency Hiding

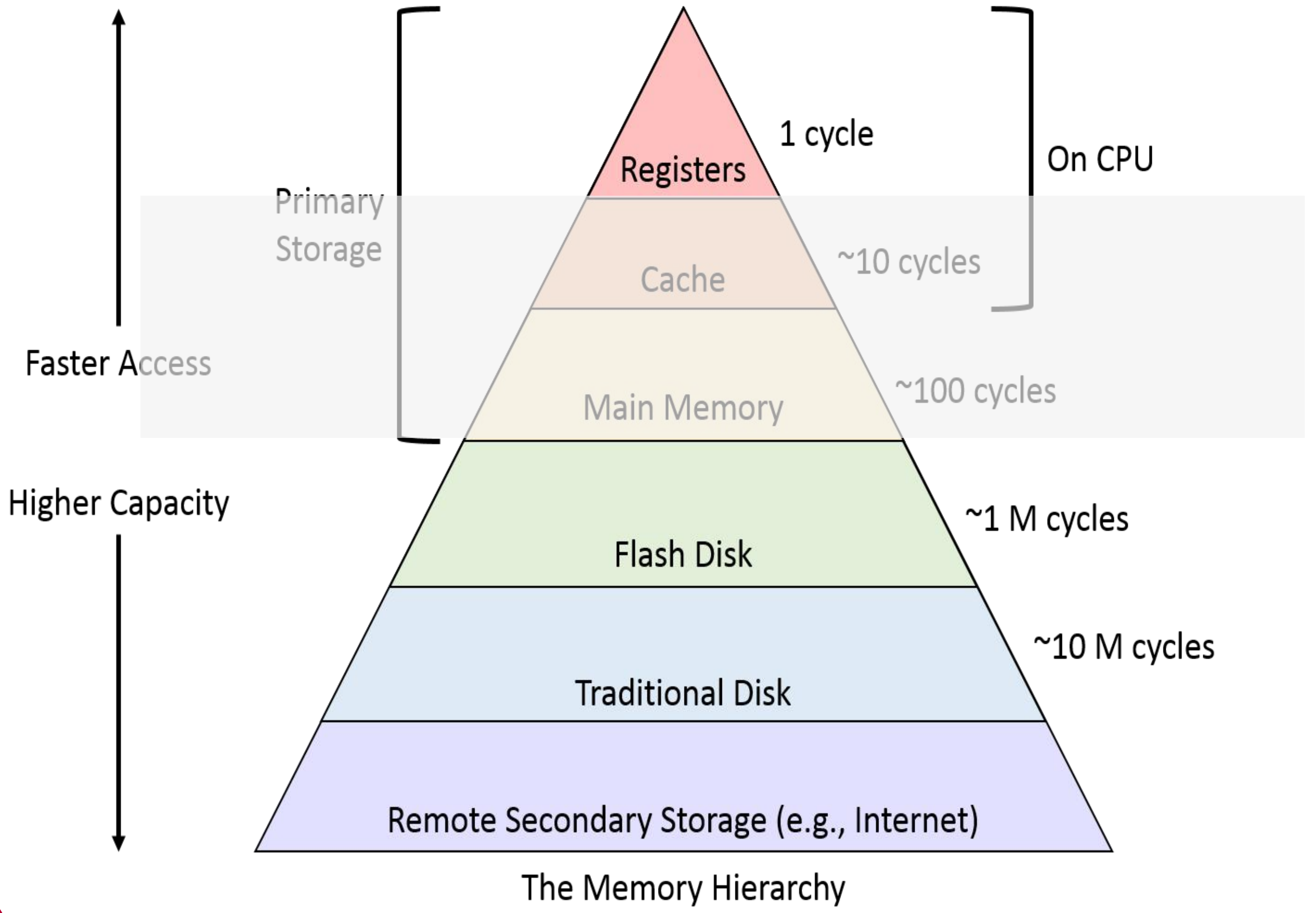
Rui Xu, Edwin H.-M. Sha, Qingfeng Zhuge, Yuhong Song, Jingzhi Lin

School of Computer Science and Technology, East China Normal University, Shanghai, China

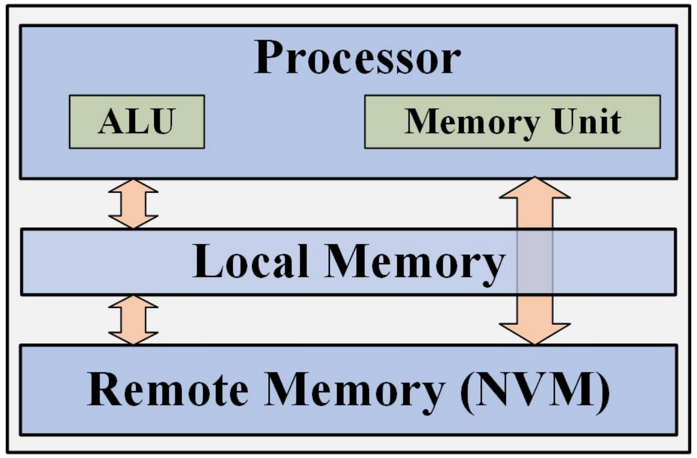
Outline

- Background
- Motivation
- Technique
- Evaluation
- Conclusion

Background



The Memory Hierarchy



Memory hierarchy
Prefetch

Background

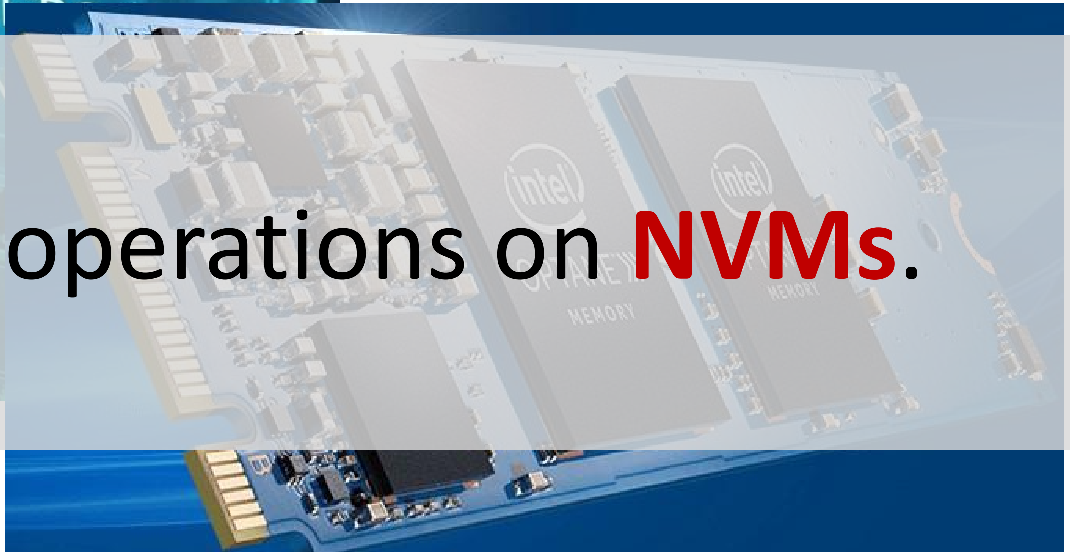
Strengths of NVM:

- ✓ Non volatile
- ✓ Large capacity
- ✓ Byte-addressability

It is vital to **reduce** the **write** operations on **NVMs**.

Shortcomes of NVM:

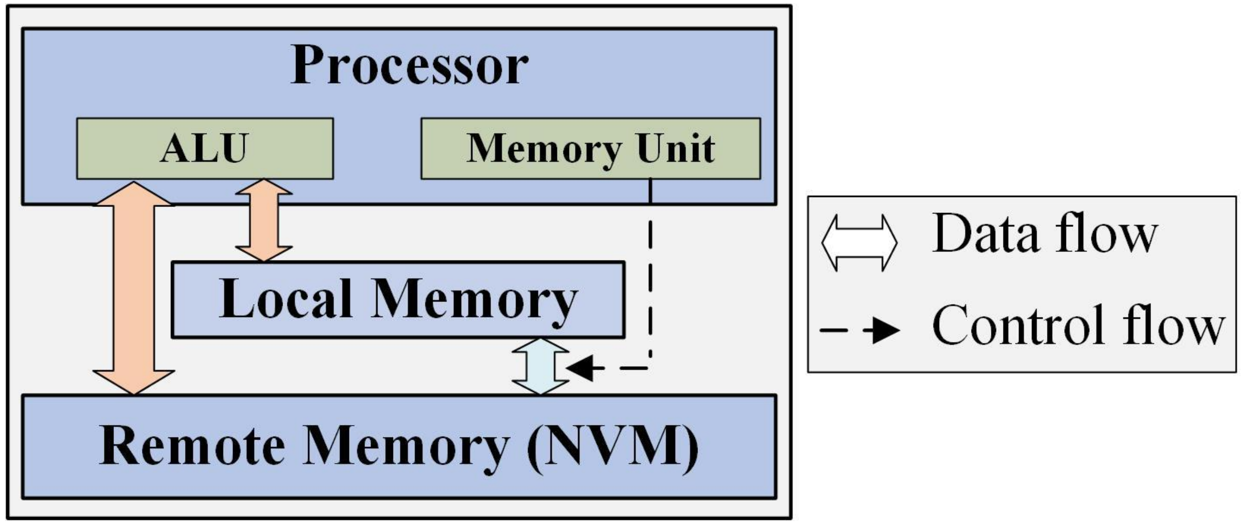
- × Limited write endurance
- × High write latency



Nested loop is the performance bottleneck in one program.
Loop tiling is a key and classic loop transformation technique for improving the data locality and reducing the comunications to remote memory.

Target architecture

ALU: computing
Memory Unit: prefetch & write
Local Memory: scratchpad memory, fast but small
Remote Memory: NVM (scratchpad memory), large but slow, limited write endurance

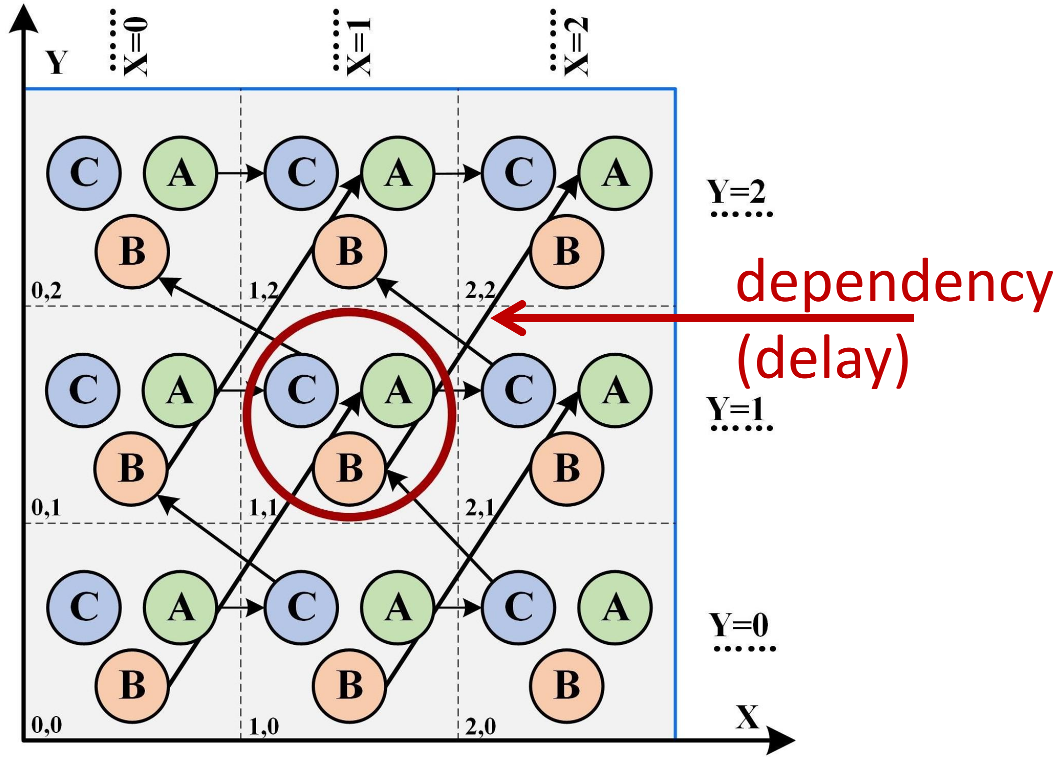


Iteration space modeling

Dependency: Data produced in one iteration will be used in other iterations.

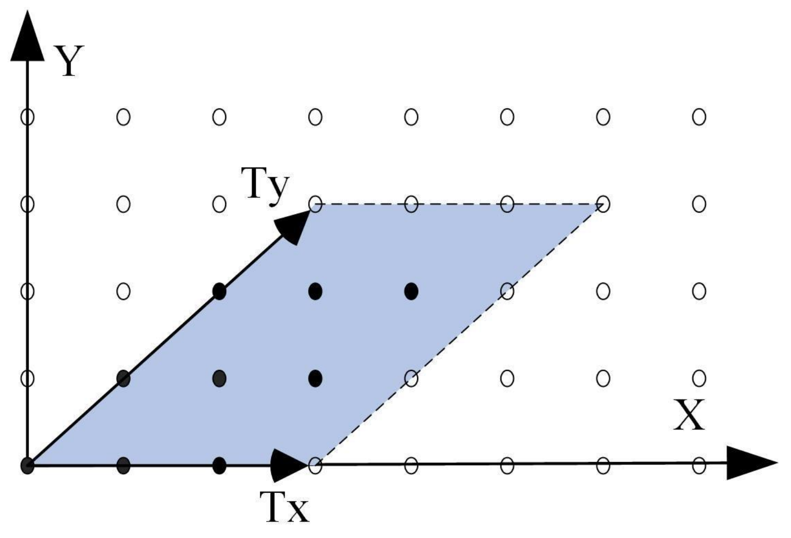
```
for(y=1;y<=m;y++):
    for(x=1;x<=n;x++):
        A[x][y] = B[x-1][y-1]*2;
        B[x][y] = C[x+1][y-1]+500;
        C[x][y] = A[x-1][y]+B[x][y]*3;
```

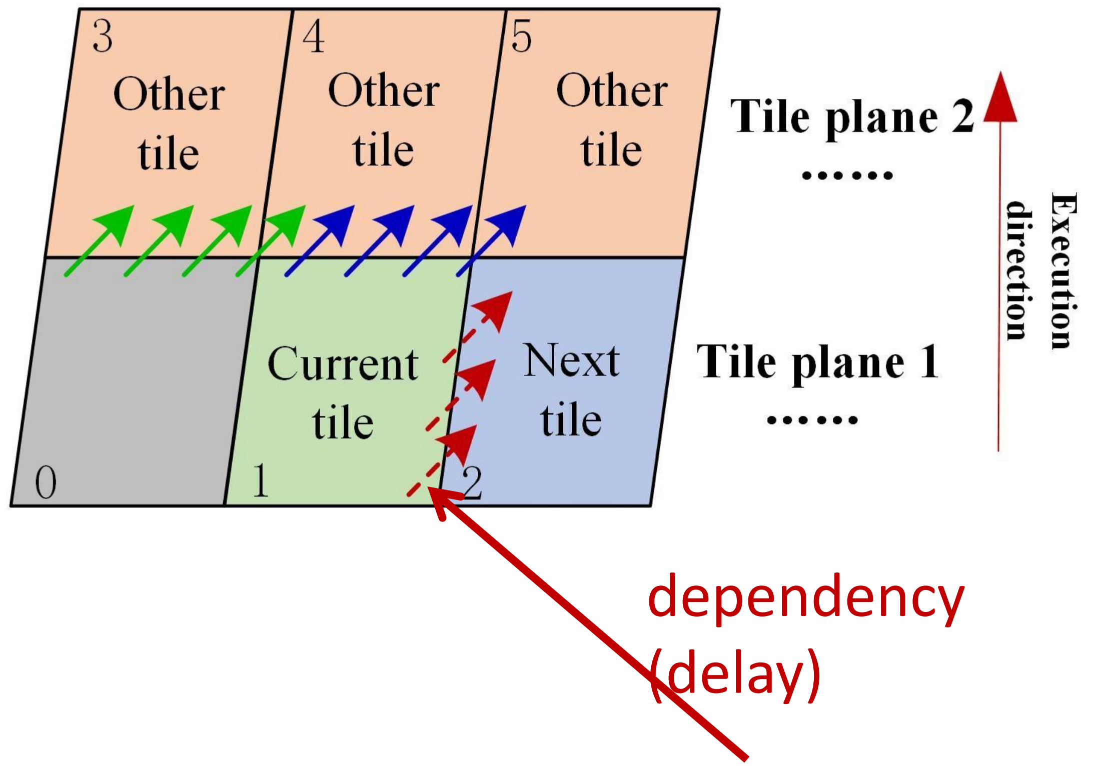
dependency: (1,1),(-1,1)(1,0)



Tile modeling

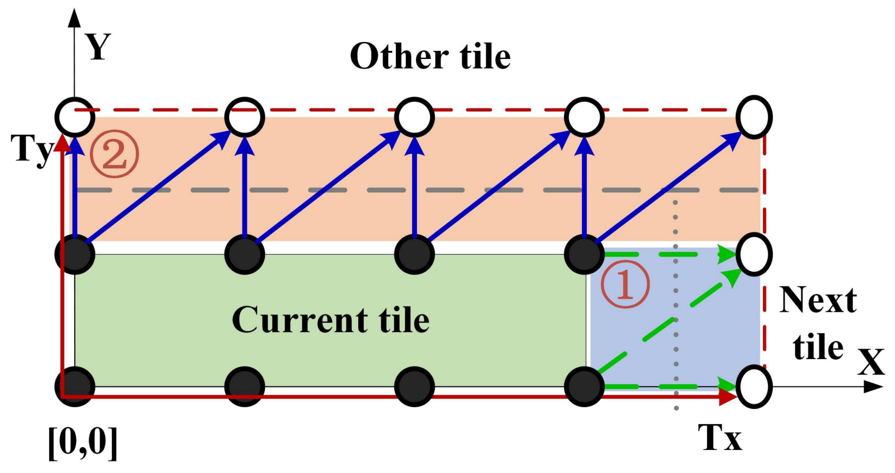
Current tile: The tile being executed.
Next tile: The tile that is executed next.
Other tile: The tile that is executed in the future.



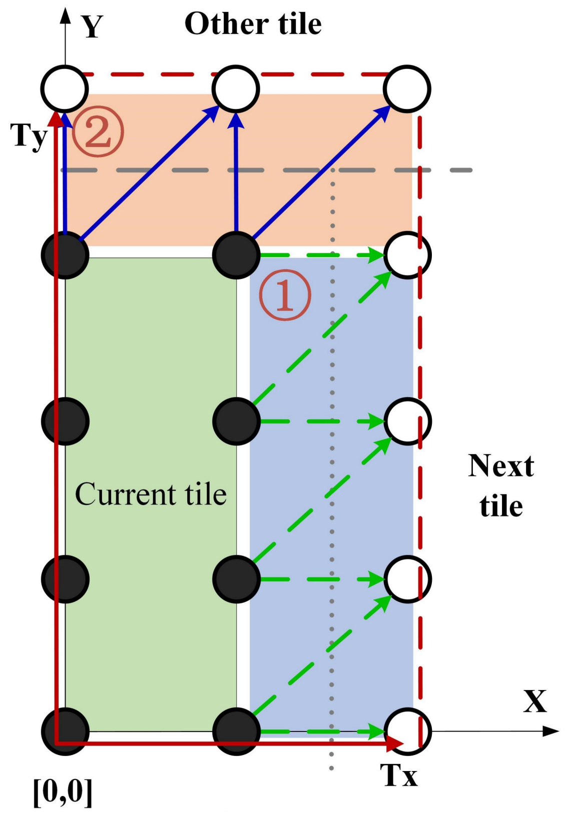


Motivation

Each iteration has 3 computations and needs one local memory.
Each loop has dependencies (1,0),(1,1),(0,1).
One computation needs 1 step.
One prefetch needs 2 steps.
One write needs 4 steps.

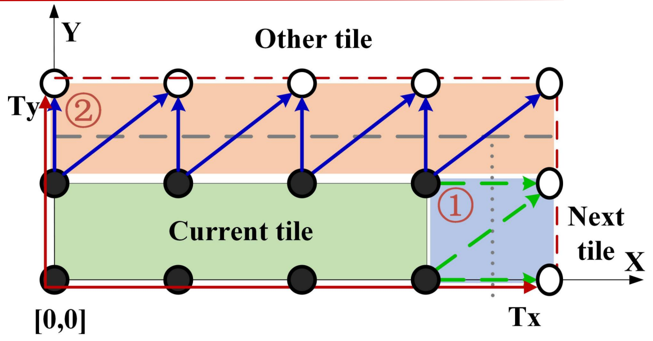


tile size 4 × 2

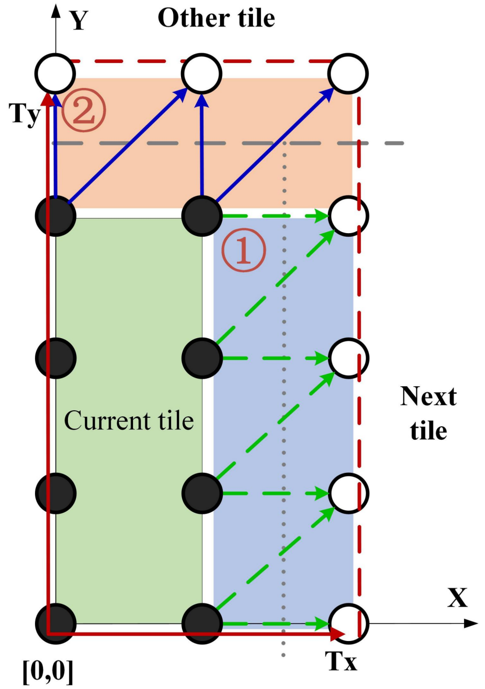


tile size 2 × 4

Motivation



(a)

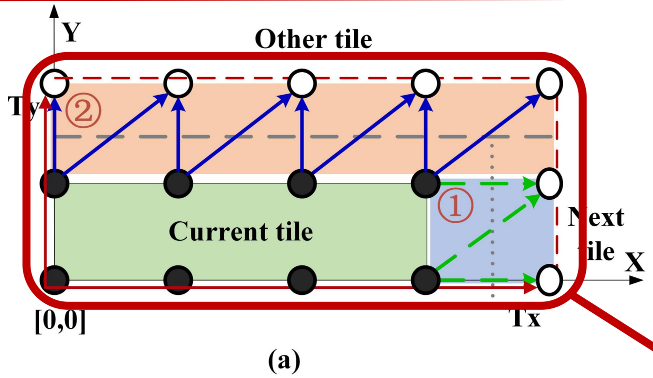


(b)

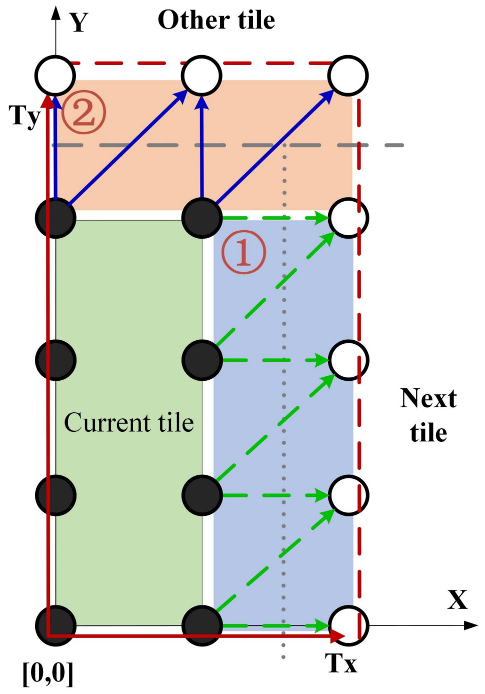
Tile size	Local memory usage	Write operations	Complete hiding memory latency
4×2			
2×4			

The less local memory usage means the larger tile size with the same local memory size.

Motivation



(a)

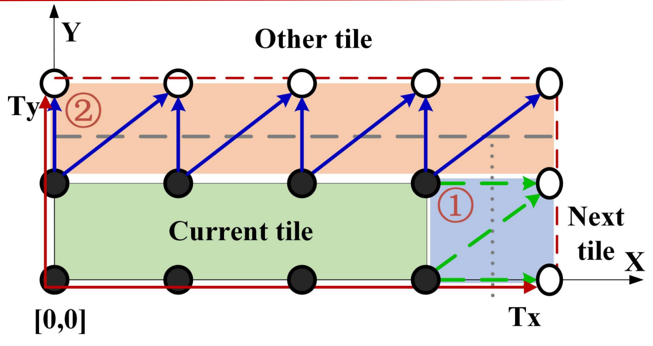


(b)

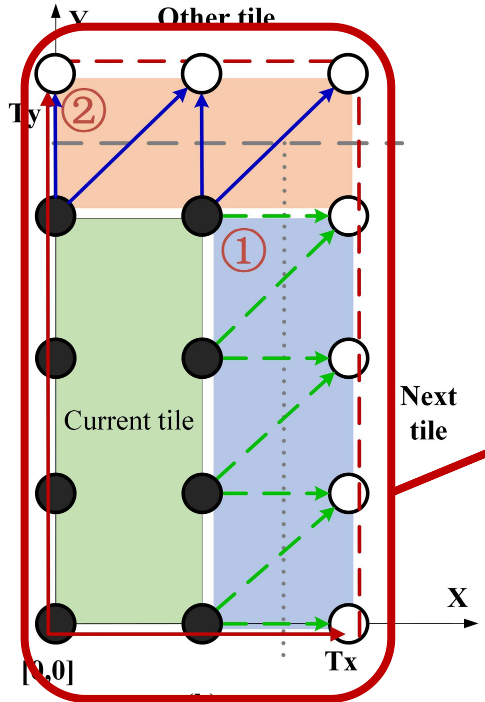
Tile size	Local memory usage	Write operations	Complete hiding memory latency
4×2	19		
2×4			

The less local memory usage means the larger tile size with the same local memory size.

Motivation



(a)

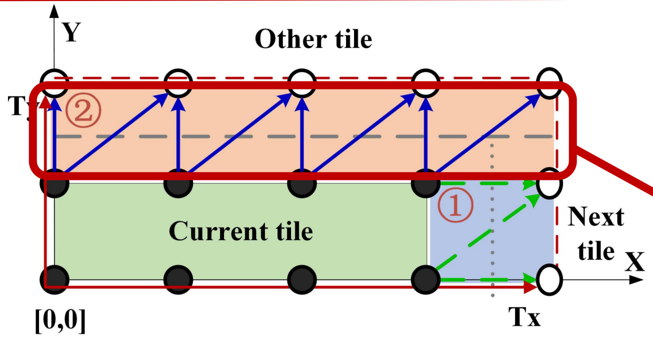


(b)

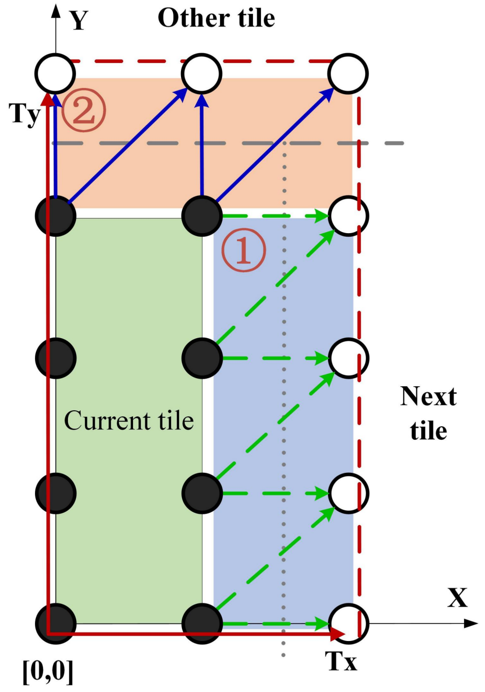
Tile size	Local memory usage	Write operations	Complete hiding memory latency
4×2	19		
2×4	15		

The less local memory usage means the larger tile size with the same local memory size.

Motivation



(a)

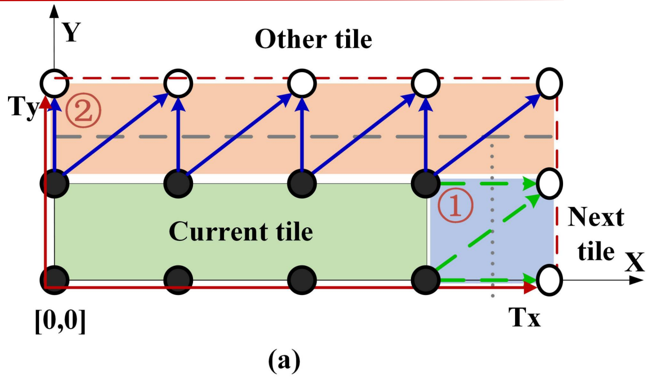


(b)

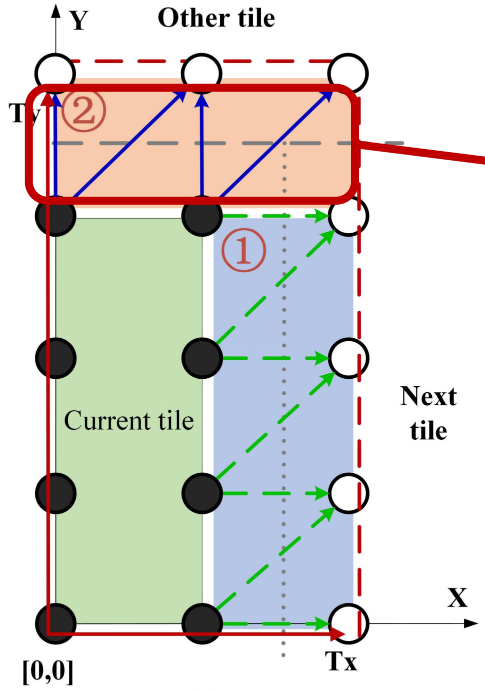
Tile size	Local memory usage	Write operations	Complete hiding memory latency
4×2	19	8	
2×4	15		

The less write operations the better.

Motivation



(a)

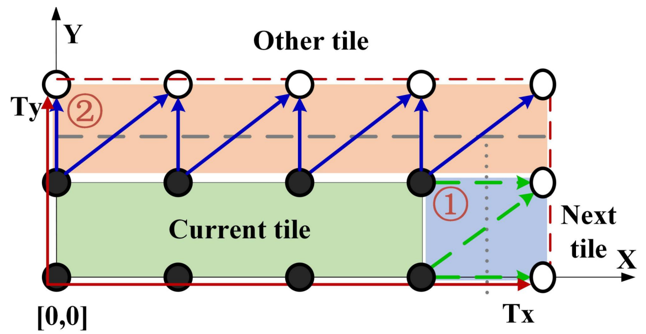


(b)

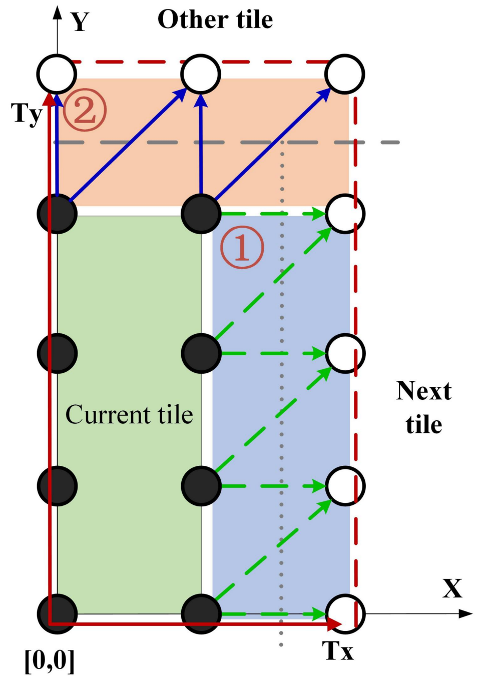
Tile size	Local memory usage	Write operations	Complete hiding memory latency
4×2	19	8	
2×4	15	4	

The less write operations the better.

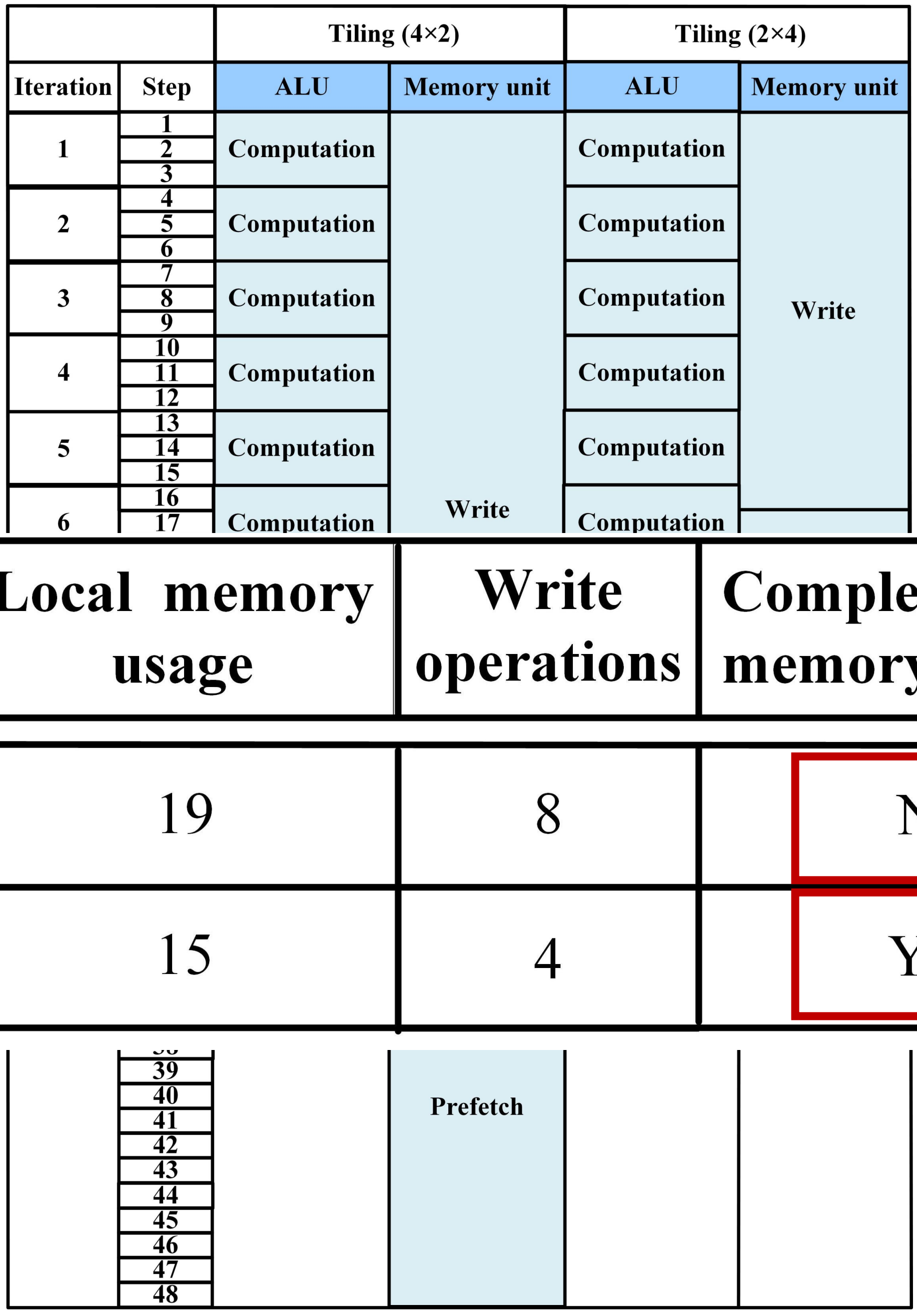
Motivation



(a)



(b)



		Tiling (4×2)		Tiling (2×4)	
Iteration	Step	ALU	Memory unit	ALU	Memory unit
1	1				
	2	Computation		Computation	
	3				
2	4				
	5	Computation		Computation	
	6				
3	7				
	8	Computation		Computation	Write
	9				
4	10				
	11	Computation		Computation	
	12				
5	13				
	14	Computation		Computation	
	15				
6	16		Write		
	17	Computation		Computation	

	39				
	40		Prefetch		
	41				
	42				
	43				
	44				
	45				
	46				
	47				
	48				

Tile size	Local memory usage	Write operations	Complete hiding memory latency
4×2	19	8	No
2×4	15	4	Yes

Goals

Obtain an **optimal loop size** for **minimizing the write** operations on NVM and pipeline schedule to **hide the memory access latency completely**.

Technique

Algorithm framework

- ✓ Tile Shape Determination
- ✓ Tile Size Determination
- ✓ Pipeline schedule

Technique

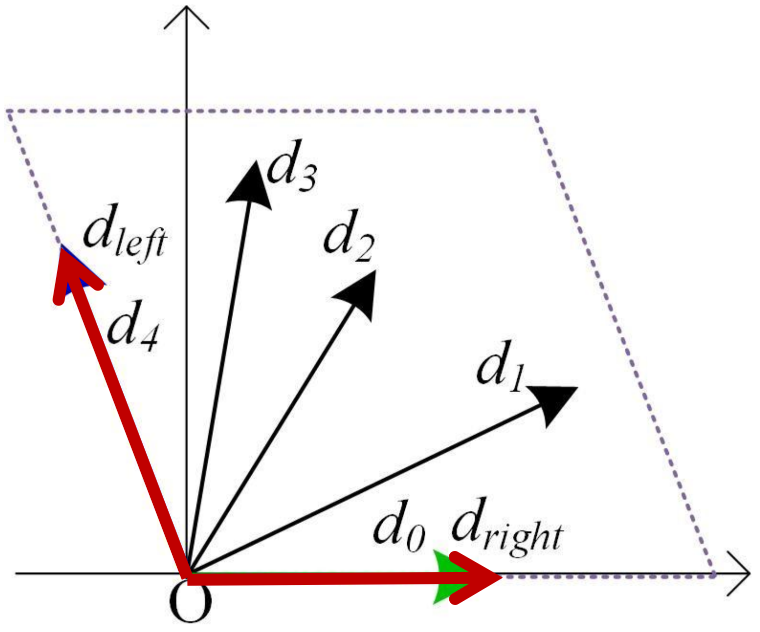
Algorithm framework

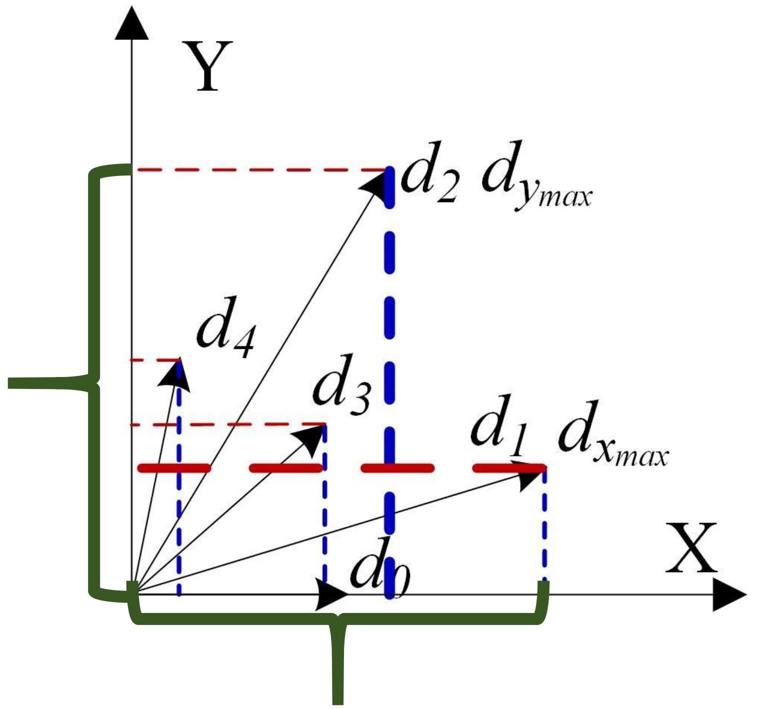
- ✓ Tile Shape Determination
- ✓ Tile Size Determination
- ✓ Pipeline schedule

Technique

Legal tile shape:

- Direction: A legal tile shape should contains all delay in one tile.
- Base tile: all delay vectors cannot pass from next tile or other tile to current tile.



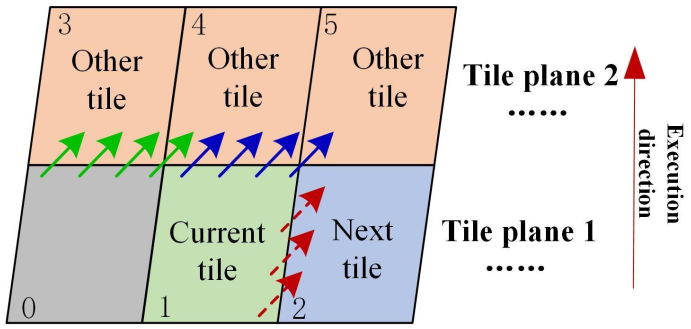


Technique

Algorithm framework

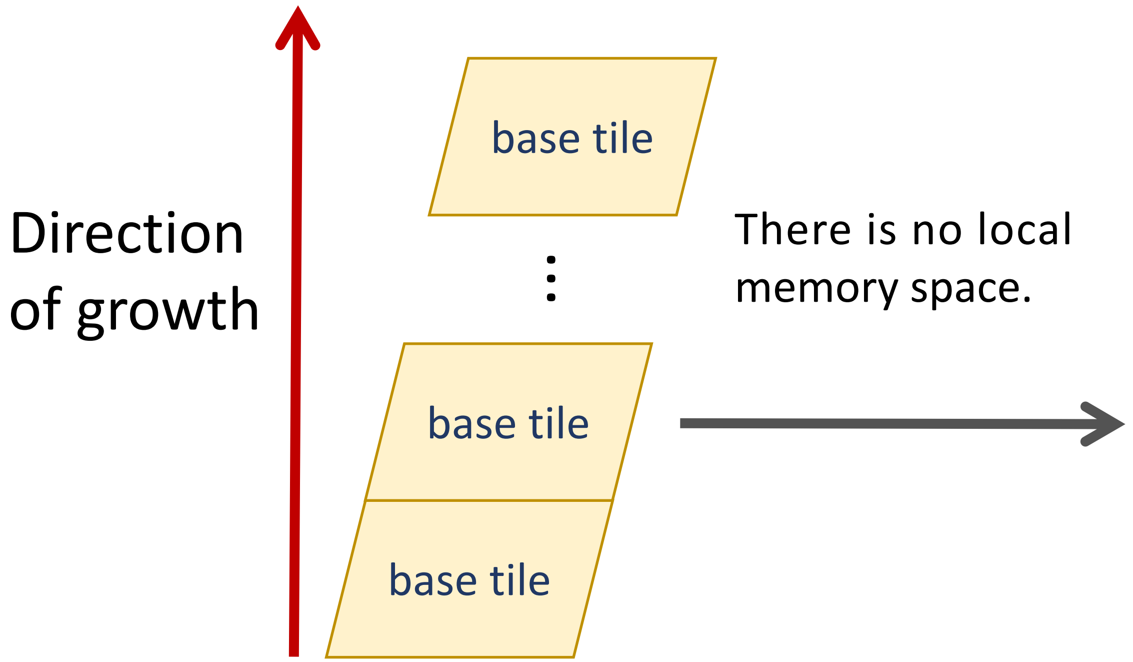
- ✓ Tile Shape Determination
- ✓ Tile Size Determination
- ✓ Pipeline schedule

Technique



Optimal tile size:

- The less the # write back, the better.
- The # values staying in local memory need to be fit local memory capacity.

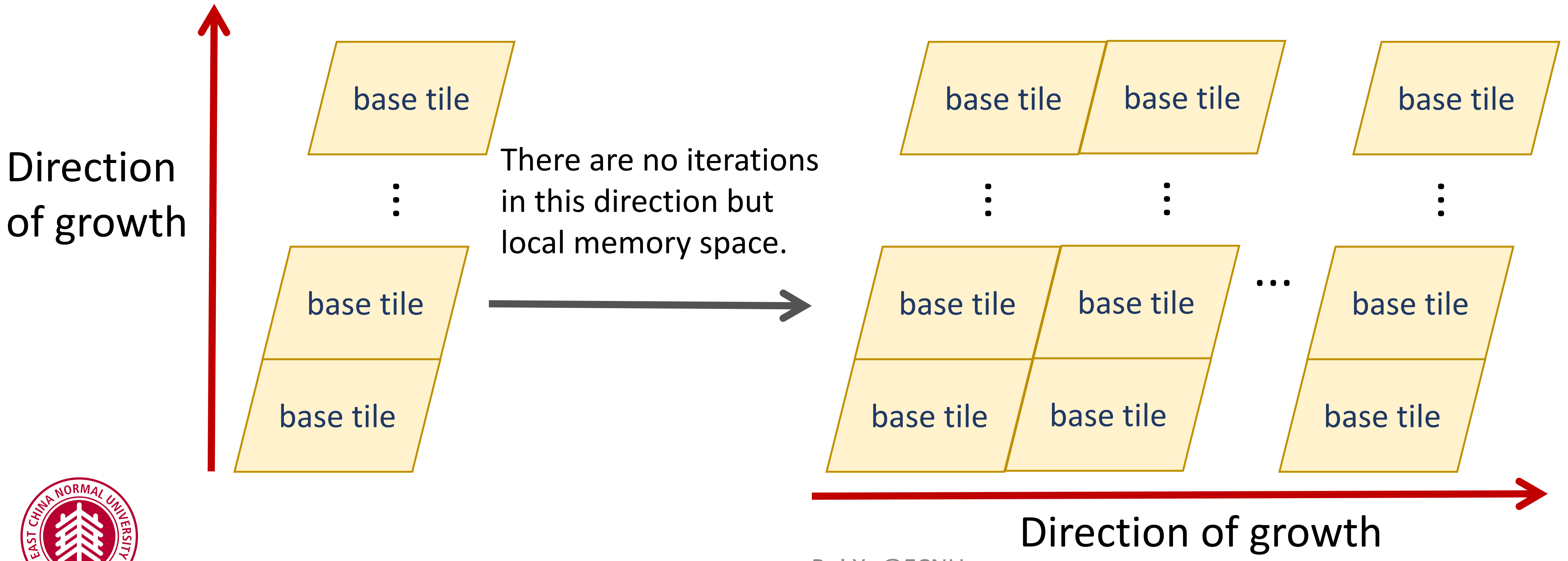


Generate the optimal loop tile size.

Technique

Optimal tile size:

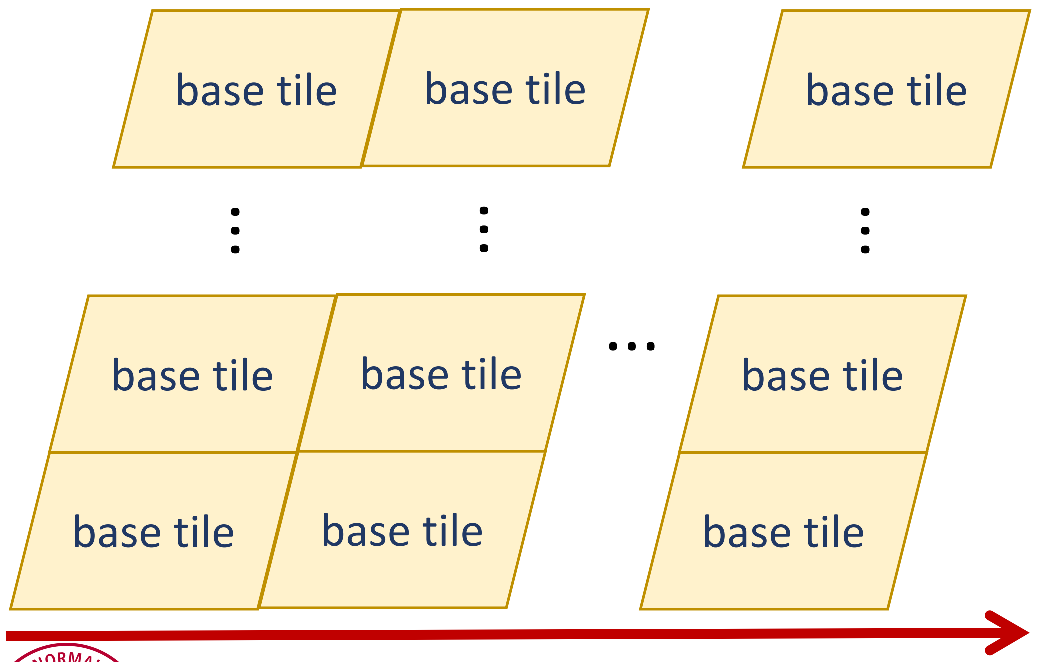
- ◆ The less the # write back, the better.
- ◆ The # values staying in local memory need to be fit local memory capacity.



Technique

Optimal tile size:

- ◆ The less the # write back, the better.
- ◆ The # values staying in local memory need to be fit local memory capacity.



Direction of growth

There are no local memory space.

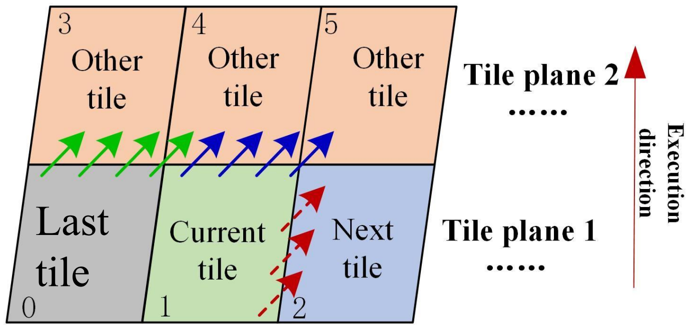
Generate the optimal loop tile size.

Technique

Algorithm framework

- ✓ Tile Shape Determination
- ✓ Tile Size Determination
- ✓ Pipeline schedule

Technique



Pipeline schedule:

◆ Hide the remote memory access latency completely, including *prefetch* and *write back*.

Step	1	2	3	4	5	6	7
ALU	computation	computation	computation	computation	computation	computation	computation
Memory unit	write back		write back	prefetch		prefetch	

For current tile

For the last tile

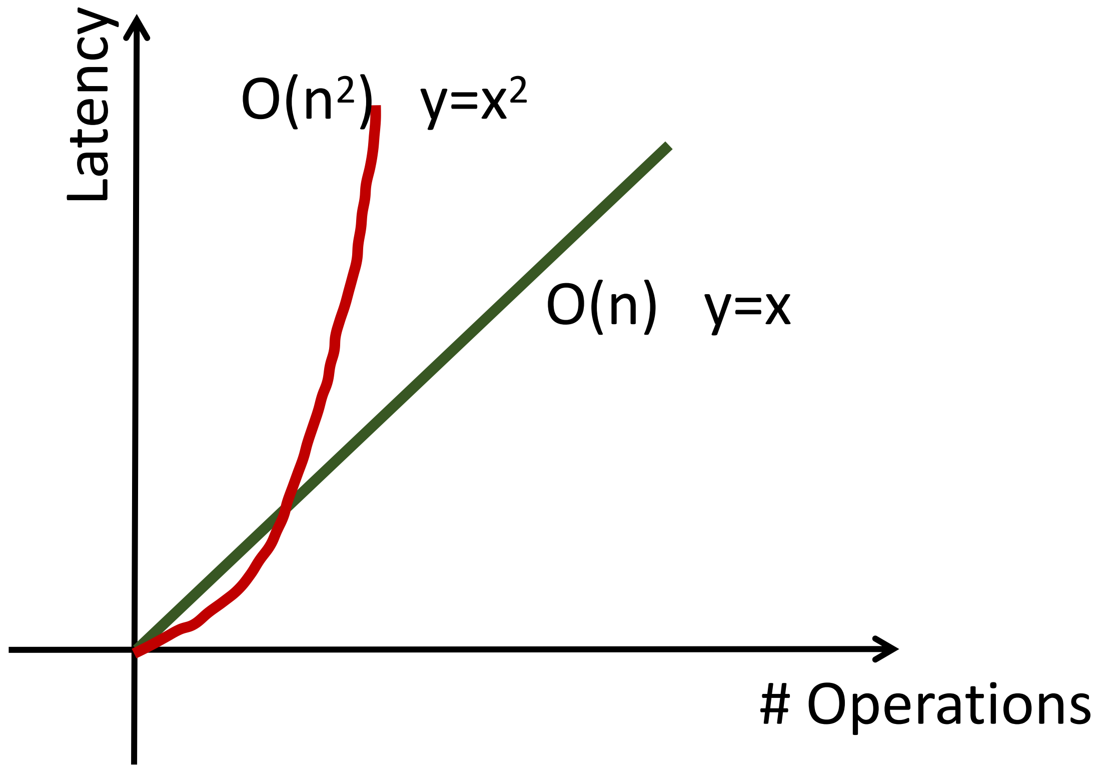
For the next tile

Technique

Pipeline schedule:

- Hide the remote memory access latency completely, including *prefetch* and *write back*.

- ☐ The growth of **# computation** is **n²**;
- ☐ The growth of **# write back** and **# prefetch** is **n**.



Evaluation

- **Configuration:**
  - **Processor**: one, equipped with one ALU and one memory unit.
  - **Local memory**: 32 KB, assume that one data is 8 bytes, thus 32 KB can hold 4096 values.
  - **Remote memory**: NVM, assume that remote memory can hold all data of each loop kernel.
  - **Latency configurations**: (1-3-5), (1-10-15), (1-10-20), and (1-20-25).

- **Workloads:**
  - 9 loop kernels from polybench are evaluated.

Evaluation

- **Compared schemes:**
  - **Baseline:** It executes iterations without loop tiling.
  - **WET[1]:** It splits nested loop to square tiles according to the local memory capacity.
  - **Kumaudha[2]:** It is a tile selection model that considers temporal and spatial reuse along dimensions of a loop nest, which will generate a rectangle tile.
  - **Split[3]:** It partitions the nested loop according to the *delay*.
  - **WMALT[4]**: It splits nested loop according to the retention time of NVM.
  - **MWCMHLT**: It is the proposed scheme.

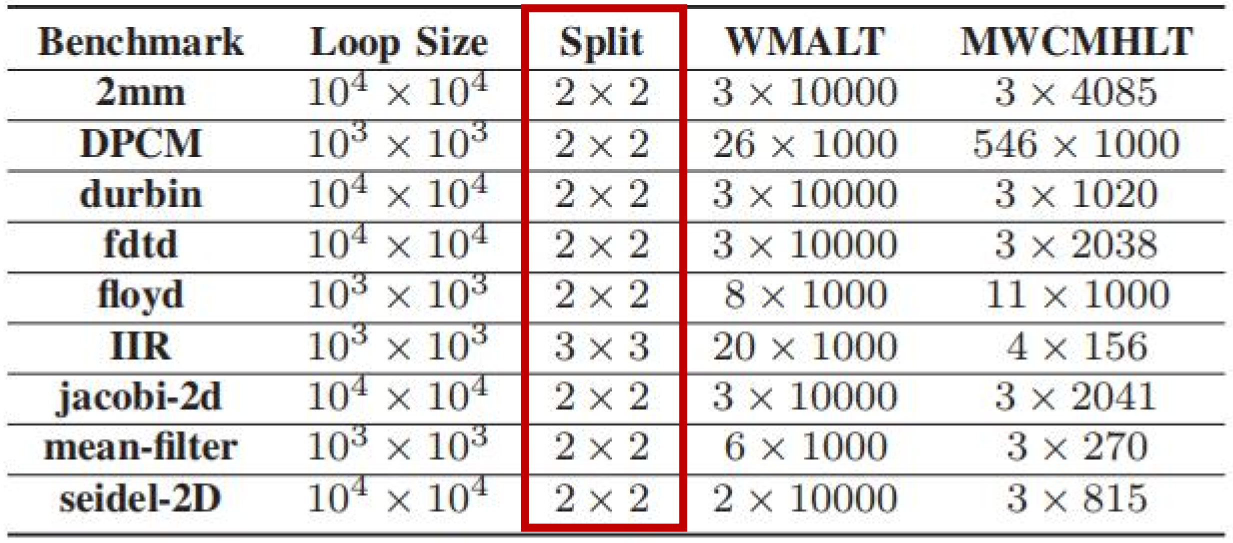
[1] Mohammad Alshboul, James Tuck, and Yan Solihin. Wet: Write efficient loop tiling for non-volatile main memory. In 2020 57th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2020.
[2] Kumudha Narasimhan, Aravind Acharya, Abhinav Baid, and Uday Bondhugula. A practical tile size selection model for affine loop nests. In Proceedings of the ACM International Conference on Supercomputing, pages 27–39, 2021.
[3] Tobias Grosser, Albert Cohen, Paul HJ Kelly, J Ramanujam, Ponuswamy Sadayappan, and Sven Verdoolaege. Split tiling for gpus: automatic parallelization using trapezoidal tiles. In Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units, pages 24–31, 2013.
[4] Keni Qiu, Qingan Li, and Chun Jason Xue. Write mode aware loop tiling for high performance low power volatile pcm. In 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2014.

Evaluation

- The tile size generated by Split is so small that the local memory cannot be fully utilized.

	WET	Kumaudha
2mm	455×455	2048 × 2048
durbin	527×527	1024 × 3072

TABLE II
TILE SIZE GENERATED BY DIFFERENT METHODS



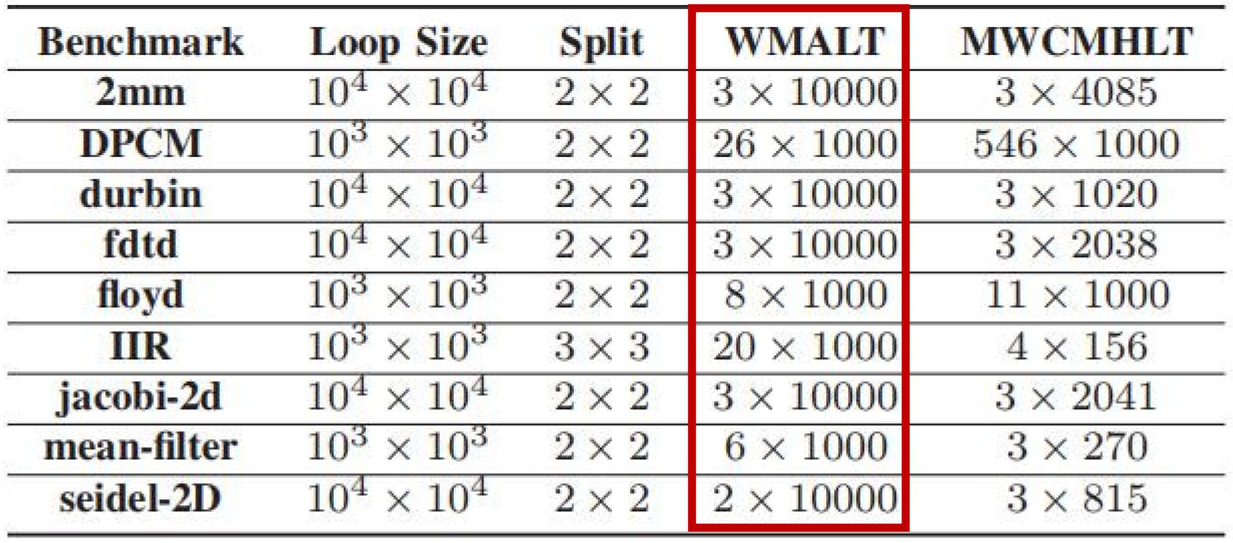
Benchmark	Loop Size	Split	WMALT	MWCMHLT
2mm	$10^4 \times 10^4$	2×2	3×10000	3×4085
DPCM	$10^3 \times 10^3$	2×2	26×1000	546×1000
durbin	$10^4 \times 10^4$	2×2	3×10000	3×1020
fdtd	$10^4 \times 10^4$	2×2	3×10000	3×2038
floyd	$10^3 \times 10^3$	2×2	8×1000	11×1000
IIR	$10^3 \times 10^3$	3×3	20×1000	4×156
jacobi-2d	$10^4 \times 10^4$	2×2	3×10000	3×2041
mean-filter	$10^3 \times 10^3$	2×2	6×1000	3×270
seidel-2D	$10^4 \times 10^4$	2×2	2×10000	3×815

Evaluation

- The tile size generated by Split is so small that the local memory cannot be fully utilized.
- The tile size generated by WMALT is too large to hold the values in one tile in local memory.

	WET	Kumaudha
2mm	455×455	2048 × 2048
durbin	527×527	1024 × 3072

TABLE II
TILE SIZE GENERATED BY DIFFERENT METHODS



Benchmark	Loop Size	Split	WMALT	MWCMHLT
2mm	$10^4 \times 10^4$	2×2	3×10000	3×4085
DPCM	$10^3 \times 10^3$	2×2	26×1000	546×1000
durbin	$10^4 \times 10^4$	2×2	3×10000	3×1020
fdtd	$10^4 \times 10^4$	2×2	3×10000	3×2038
floyd	$10^3 \times 10^3$	2×2	8×1000	11×1000
IIR	$10^3 \times 10^3$	3×3	20×1000	4×156
jacobi-2d	$10^4 \times 10^4$	2×2	3×10000	3×2041
mean-filter	$10^3 \times 10^3$	2×2	6×1000	3×270
seidel-2D	$10^4 \times 10^4$	2×2	2×10000	3×815

Evaluation

- The tile size generated by Split is so small that the local memory cannot be fully utilized.
- The tile size generated by WMALT is too large to hold the values in one tile in local memory.
- Our tile size is fit to local memory.

	WET	Kumaudha
2mm	455×455	2048 × 2048
durbin	527×527	1024 × 3072

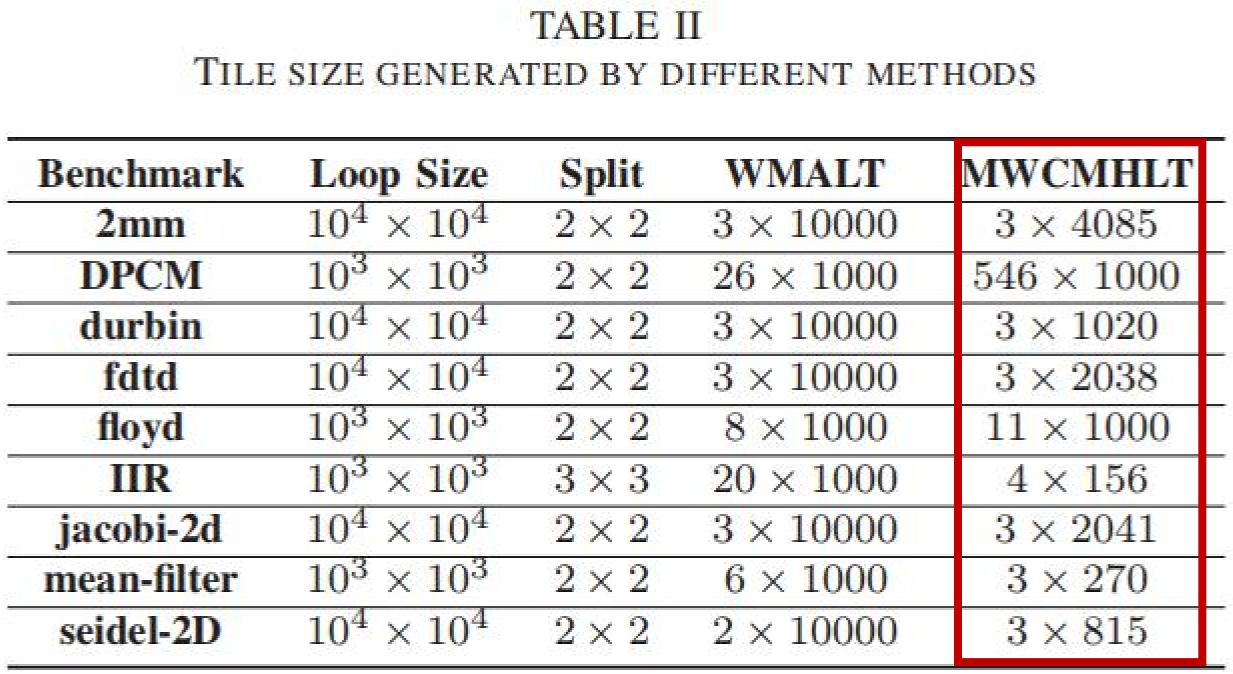
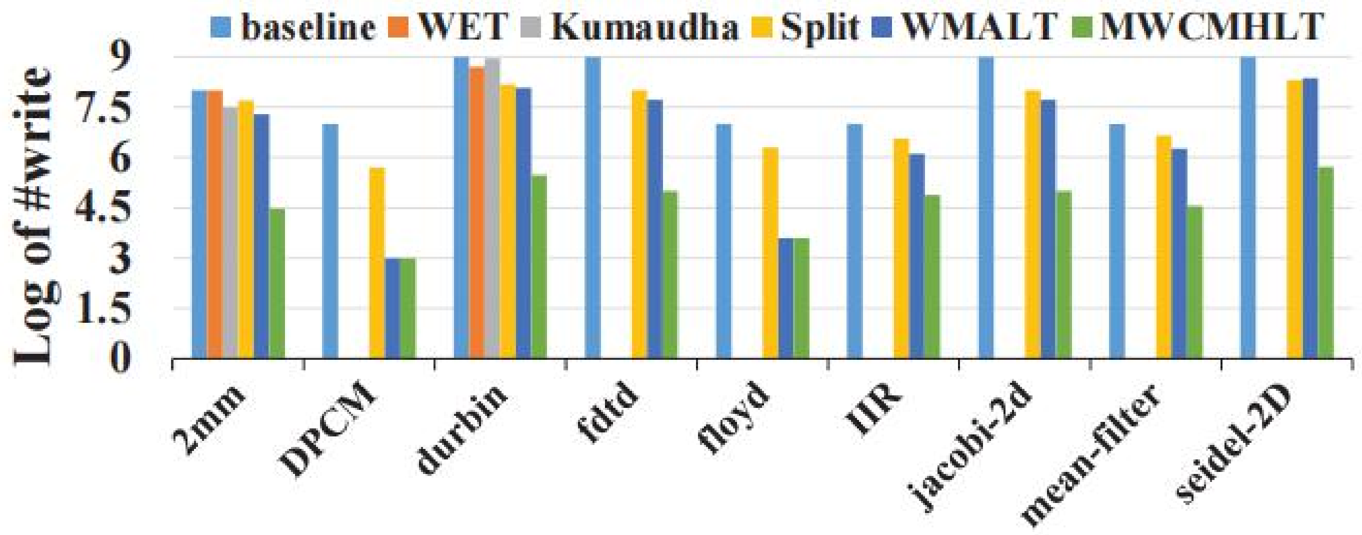


TABLE II
TILE SIZE GENERATED BY DIFFERENT METHODS

Benchmark	Loop Size	Split	WMALT	MWCMHLT
2mm	$10^4 \times 10^4$	2×2	3×10000	3×4085
DPCM	$10^3 \times 10^3$	2×2	26×1000	546×1000
durbin	$10^4 \times 10^4$	2×2	3×10000	3×1020
fdtd	$10^4 \times 10^4$	2×2	3×10000	3×2038
floyd	$10^3 \times 10^3$	2×2	8×1000	11×1000
IIR	$10^3 \times 10^3$	3×3	20×1000	4×156
jacobi-2d	$10^4 \times 10^4$	2×2	3×10000	3×2041
mean-filter	$10^3 \times 10^3$	2×2	6×1000	3×270
seidel-2D	$10^4 \times 10^4$	2×2	2×10000	3×815

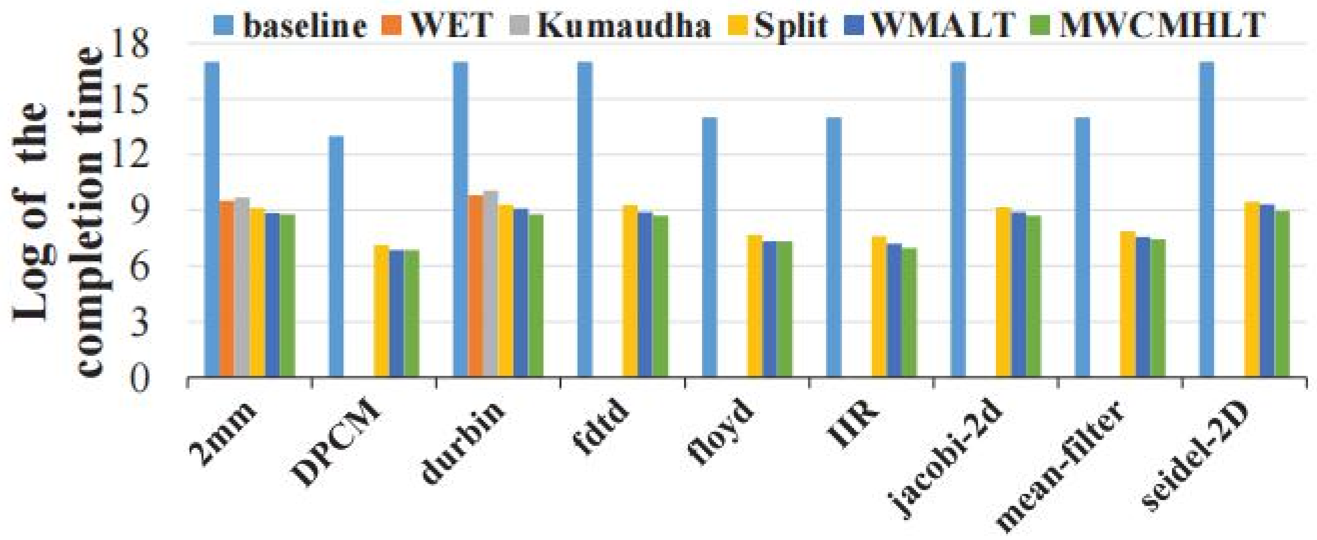
Evaluation

- **Compared to other approaches, MWCMHLT reduces the number of writes by 99.8%, 99.6%, 99.8%, 99.5%, 76.8% on average, respectively.**



Evaluation

- **Compared to other approaches, MWCMHLT reduces the completion time by 99.9%, 85.8%, 91.1%, 63.5%, 28.5% on average, respectively.**



Evaluation

- **The latency on ALU part is no less than the latency on memory unit part, the NVM access latency can be completely hidden.**

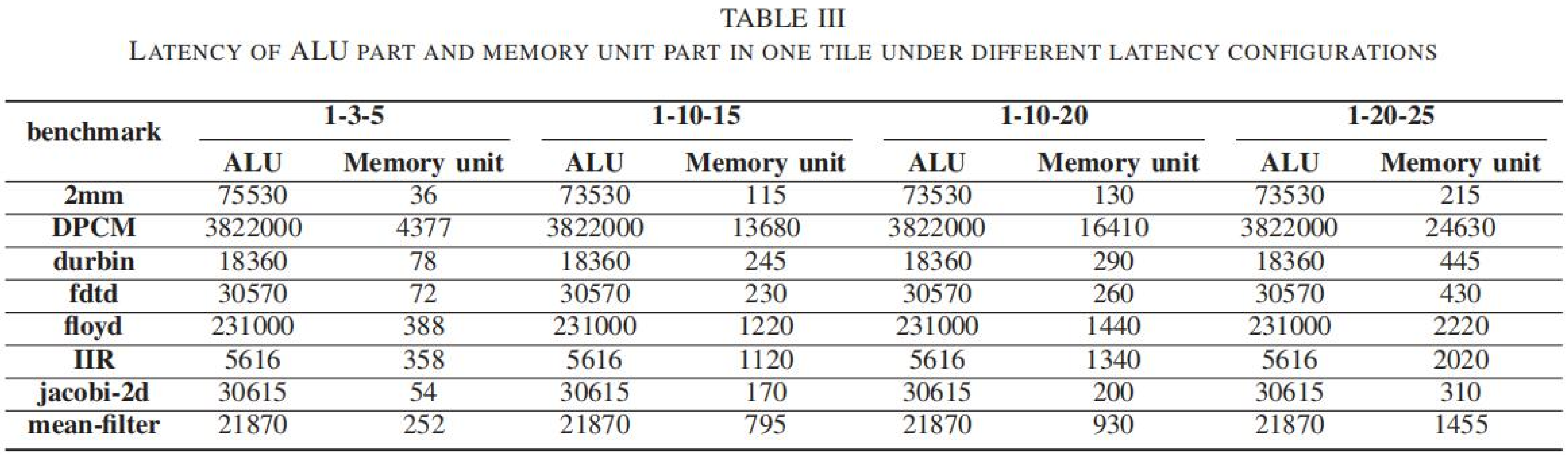


TABLE III
LATENCY OF ALU PART AND MEMORY UNIT PART IN ONE TILE UNDER DIFFERENT LATENCY CONFIGURATIONS

benchmark	1-3-5		1-10-15		1-10-20		1-20-25	
	ALU	Memory unit	ALU	Memory unit	ALU	Memory unit	ALU	Memory unit
2mm	75530	36	73530	115	73530	130	73530	215
DPCM	3822000	4377	3822000	13680	3822000	16410	3822000	24630
durbin	18360	78	18360	245	18360	290	18360	445
fdtd	30570	72	30570	230	30570	260	30570	430
floyd	231000	388	231000	1220	231000	1440	231000	2220
IIR	5616	358	5616	1120	5616	1340	5616	2020
jacobi-2d	30615	54	30615	170	30615	200	30615	310
mean-filter	21870	252	21870	795	21870	930	21870	1455

Conclusion

Observed:

- Memory hierachy and prefetch are popular in emebedded systems.
- NVM has the limited write endurance.
- Nested loop is the performance bottleneck in one program.

Proposed:

- A tile shape determination strategy to obtain a legal tile shape.
- A tile size determination scheme to generated an optimal tile size for minimizing the write operations on NVMs.
- A pipiline schedule policy to hide the remote memory latency completely.

Evaluated:

- Our scheme can effectively reduce write operations (95.1% improvements) and can always completely hide NVM access latency.

Optimal Loop Tiling for Minimizing Write Operations on NVMs with Complete Memory Latency Hiding

If any questions, please contact us!

Edwin H.-M. Sha edwinsha@cs.ecnu.edu.cn
Rui Xu ruixu412@gmail.com

Thank you!

Questions?