MemFlow:
Memory-Driven Data Scheduling with Datapath Co-design in Accelerators for Large-scale Applications

Jade Nie

Sharad Malik

This work is sponsored in part by SONIC and C-FAR, funded centers of STARnet, a Semiconductor Research Corporation (SRC) program sponsored by MARCO and DARPA

PRINCETON UNIVERSITY

Demands of large-scale computation

Eigenface

Feature size = 500 x 500 = 250,000
Eigendecomposition on matrix 250,000x250,000

Neural networks

AlexNet	240MB
VGG16	552MB
GoogLeNet	28MB
SqueezeNet	4.8MB

Accelerator design is memory-driven

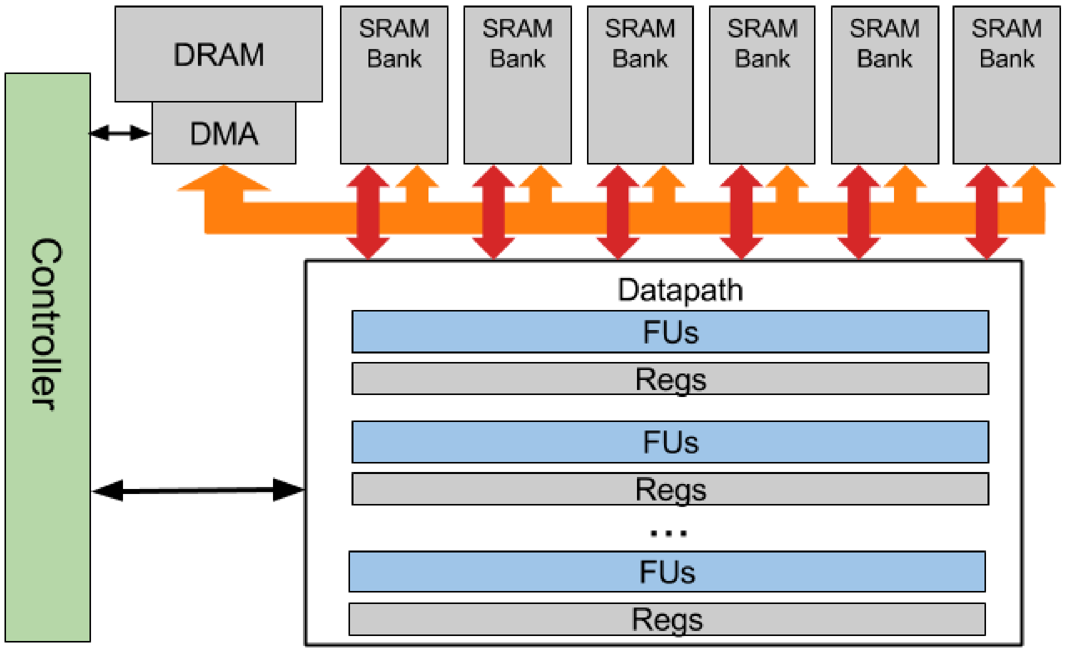
On-chip SRAM is limited ✚ DRAM access is inefficient

Intel Core i7 4-8MB Cache
Intel Core i9 25MB Cache
Nvidia Geforce 1080 GTX
    4928KB SRAM
TPU 24MB SRAM
FPGA 10MB

Latency:
    10x SRAM
    20x function units
Energy consumption:
    100x SRAM
    1000x function units

Accelerator architecture

Software-controlled scratch-pad SRAM



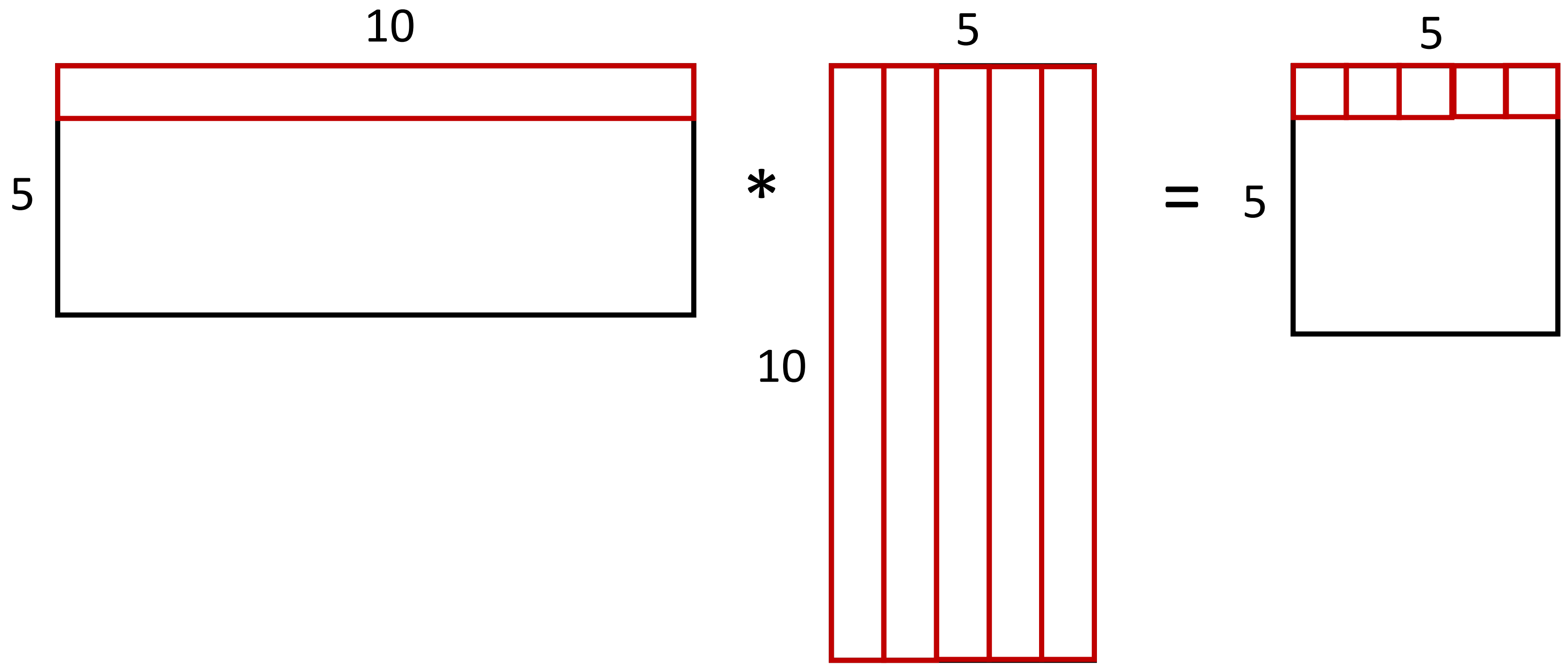
DRAM accesses:

- Compulsory accesses:
  - Load initial input
  - Store back final results
- Accesses due to SRAM limited size:
  - Load/store spilled intermediate results

Determining DRAM accesses:
Computation scale
SRAM size
Data scheduling

Effect of scheduling - DRAM accesses

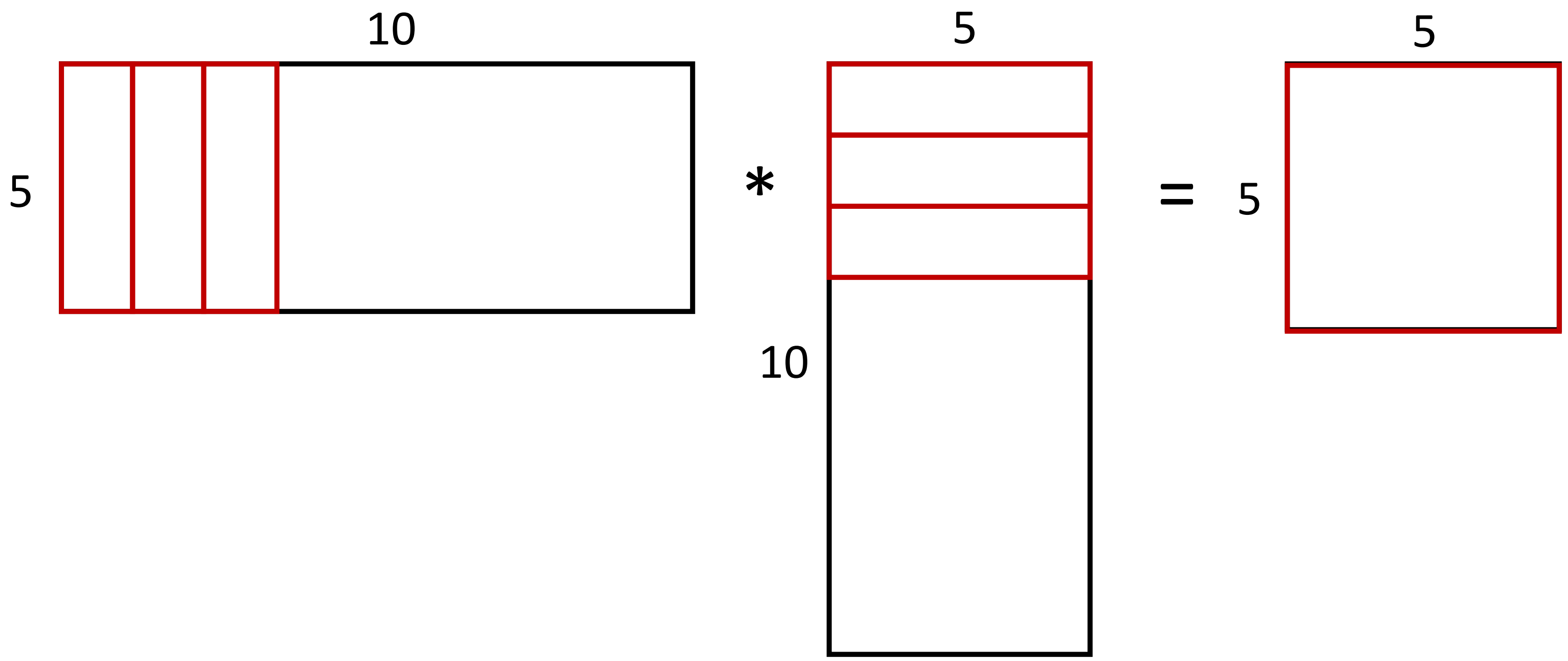
Scratch-pad size: 35 scalar data



DRAM Accesses due to spilling: 30

Effect of scheduling - DRAM accesses

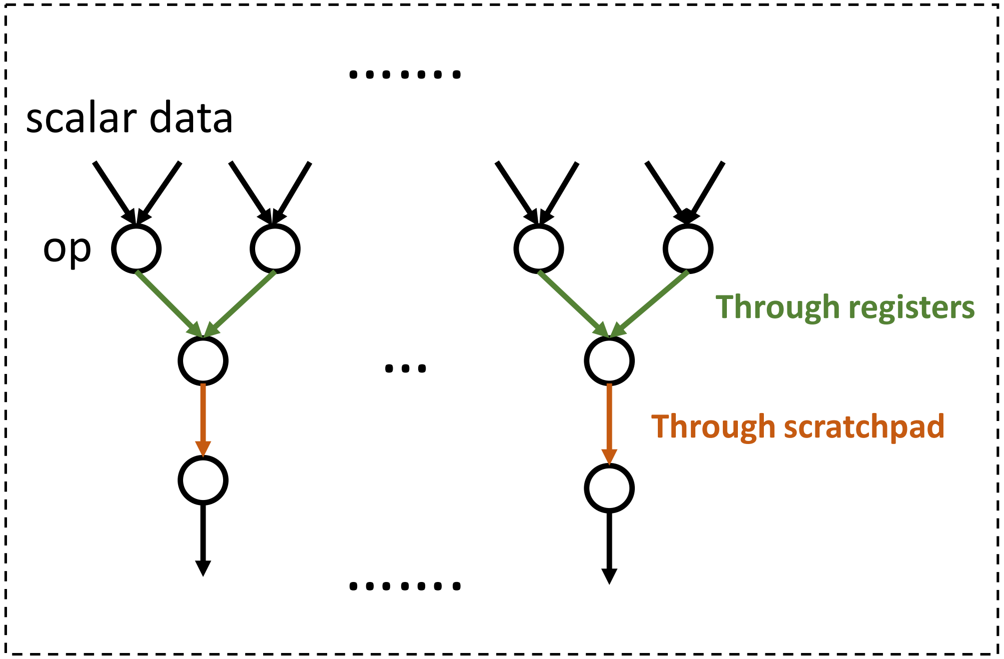
Scratch-pad size: 35 scalar data



DRAM Accesses due to spilling: 0

Global Optimization – NP-hard problem

Data dependency graph



Variables:
- Starting cycle of each op

Objective:
- Number of data spilling
- Number of cycles

Constraints:
- Bound on scratchpad size
- Bound on read/write ports
- Function units numbers

Optimization tools:
- Modern constraints solvers: SAT, SMT, ILP
- Stimulated annealing

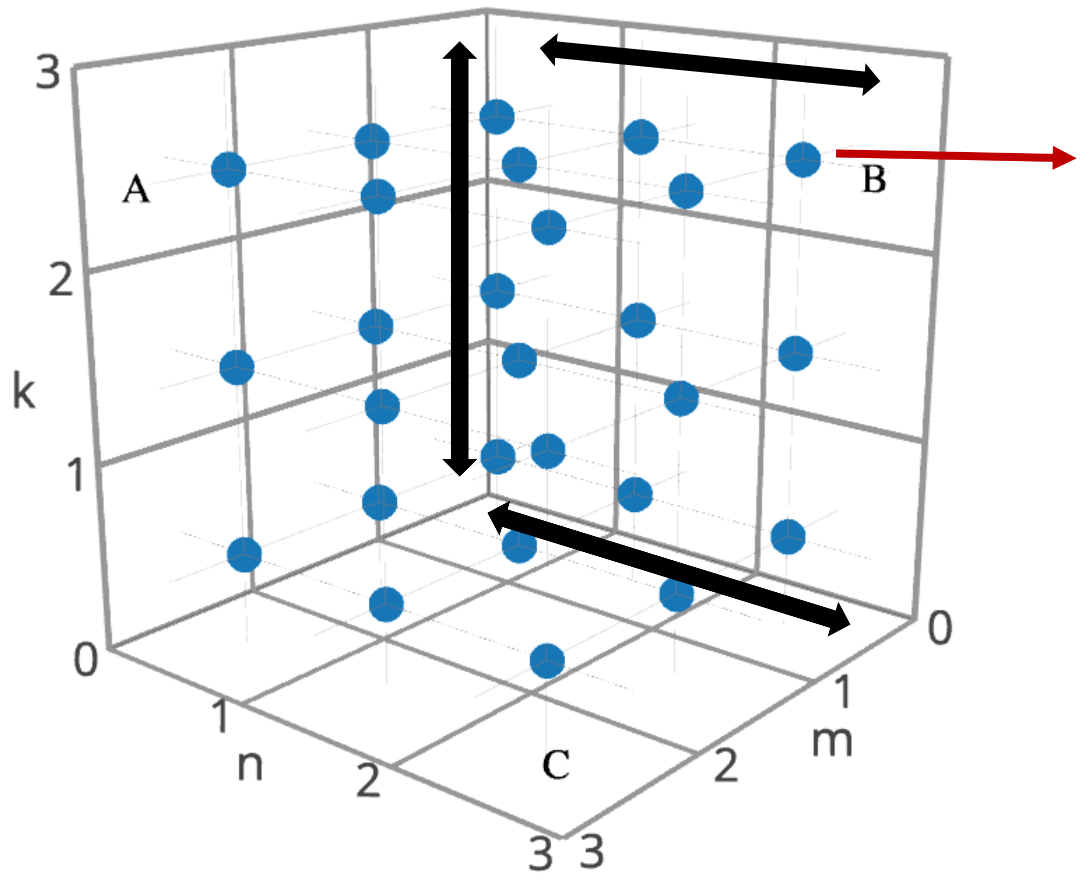
Not feasible:
- Complexity
- Irregular scheduling – hard to control

Optimization with regularity

MemFlow

- Tunable parameters from three levels
  - Level1: block dimensions
  - Level2: block scheduling
  - Level3: parallelism of local computing
- Mathematical modeling
  - DRAM accesses, SRAM accesses, cycles
- Integrated optimization

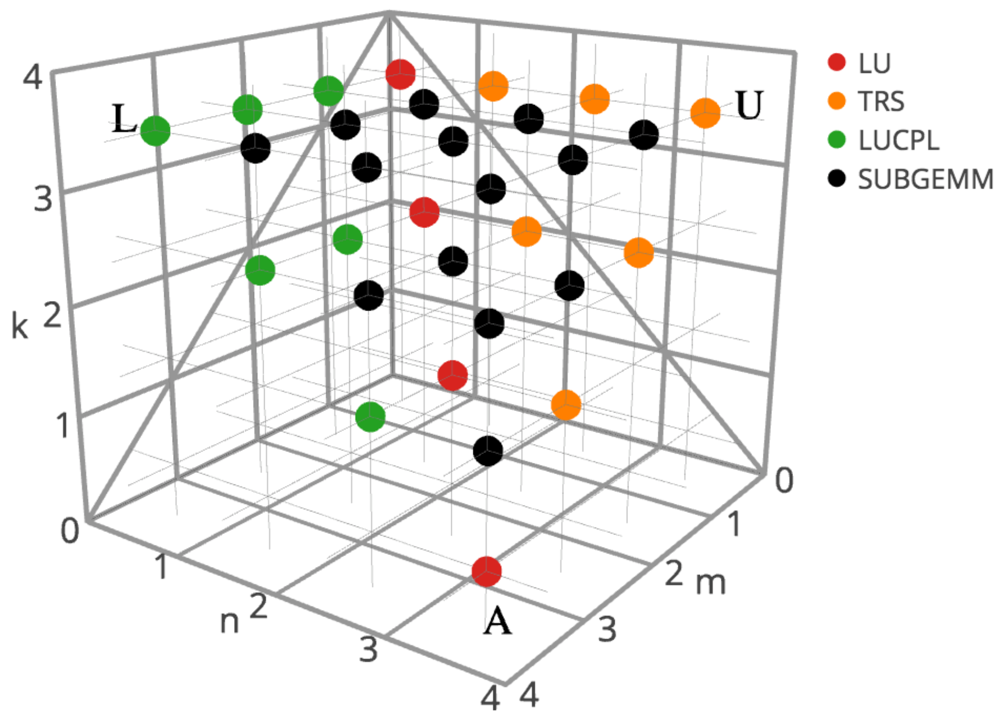
Blocking – matrix multiply



Macro node:
Block computation
$A_{Block} * B_{Block} + C_{Block} \to C_{Block}$

Blocking – LU factorization

$A \rightarrow L * U$ *L* is lower triangular matrix, *U* is upper triangular matrix



Block computation

LU

$$A \rightarrow L_{lower} * U_{upper}$$

TRS

$$\mathrm{L}_{lower}^{-1} * \mathrm{A} \rightarrow U$$

LUCPL

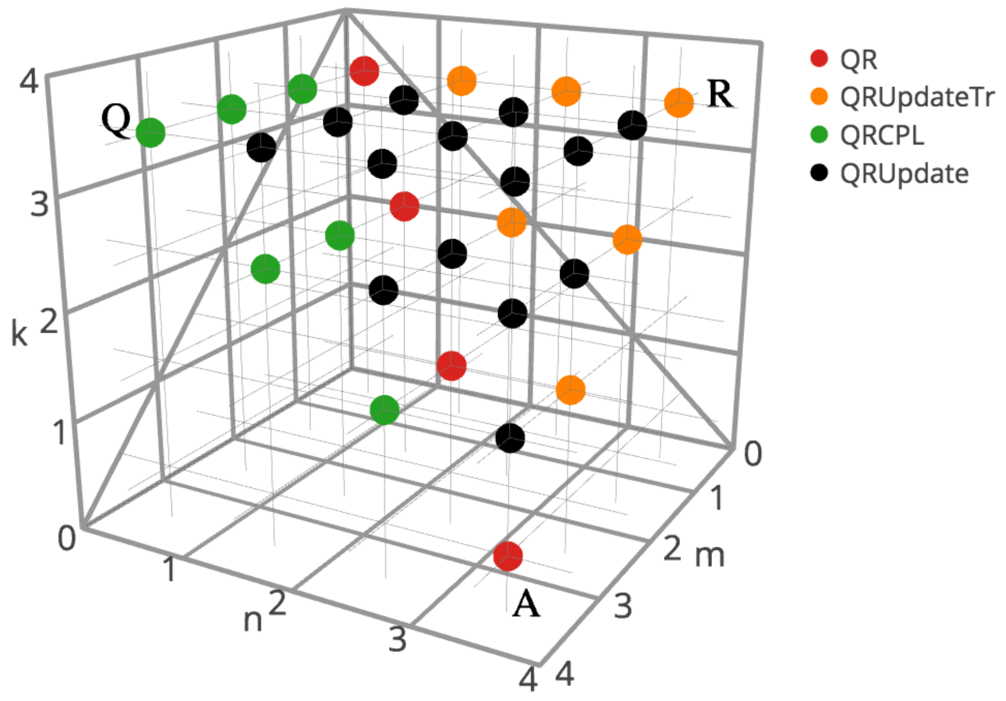
$$A * U_{upper}^{-1} \rightarrow L$$

SUBGEMM

$$A - L * U \rightarrow A$$

Blocking – QR factorization

$A \rightarrow Q \ * R$ $\quad$ *Q* is orthogonal matrix, *R* is upper triangular matrix



Block computations:

QR

$$A \rightarrow H(V_{lower}) \ * R_{upper}$$

QRUpdateTr

$$\mathrm{H}(\mathrm{V}_{\mathrm{lower}}) * \mathrm{A} \rightarrow R$$

QRCPL

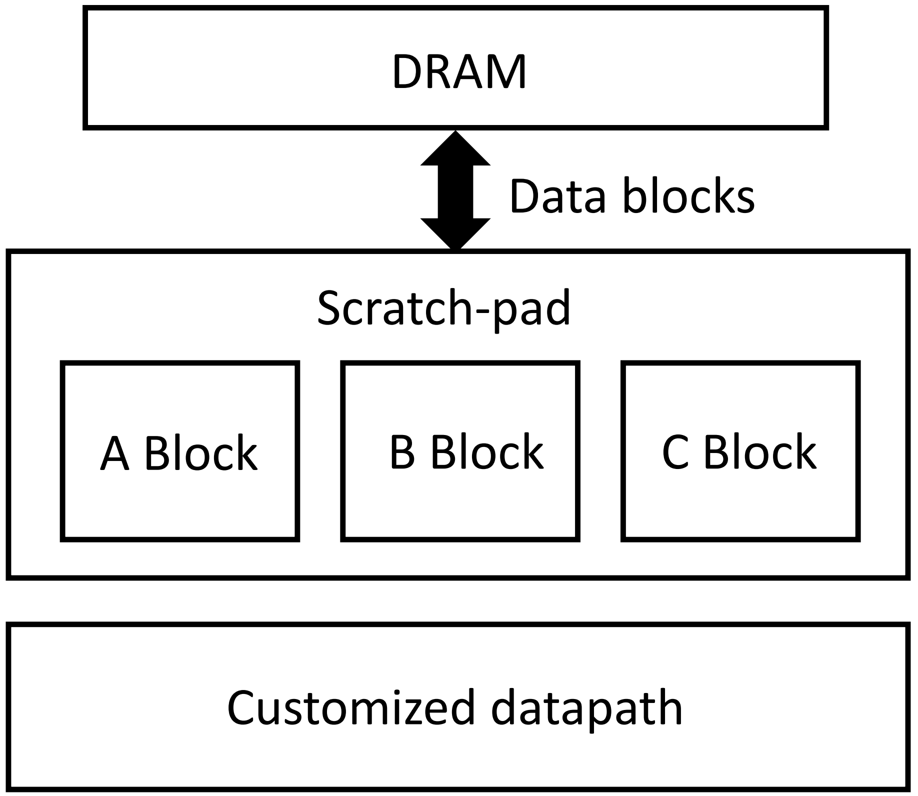
$$A * R_{upper}^{-1} \rightarrow H(V)$$

QRUpdate

$$H(V) * [R, A] \rightarrow [R, A]$$

$$V = [v_1, v_2, .., v_n], H(V) = P_n P_{n-1} .. P_1 \ where \ P_i = I - v_i v_i^T$$

Why blocking?

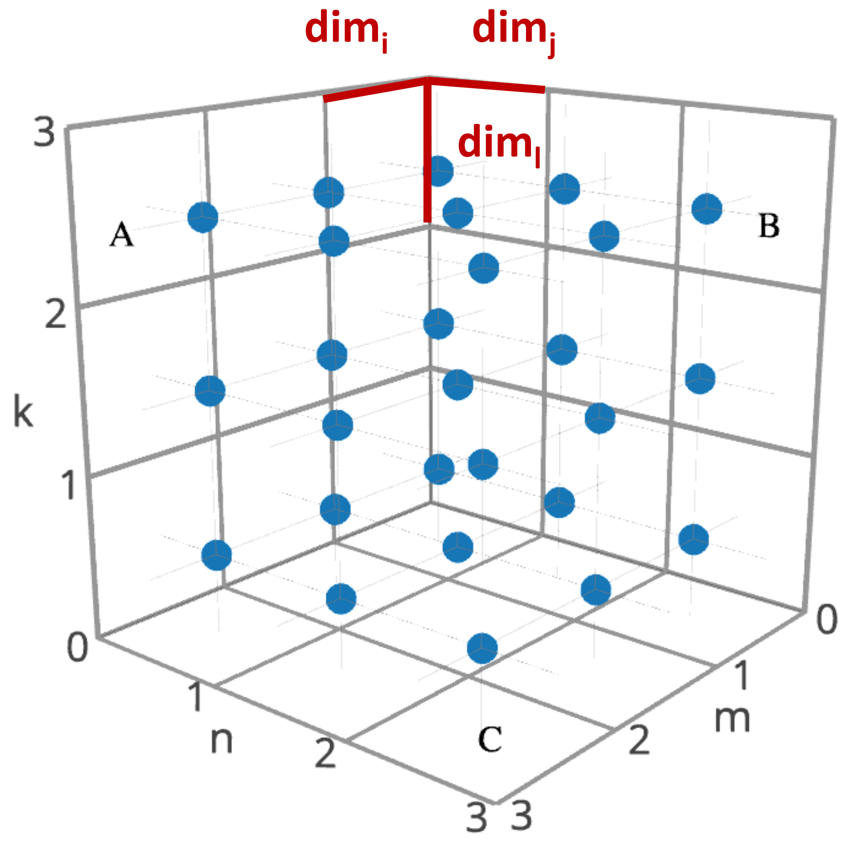


Increase data reuse for each array

Balance data reuse between arrays

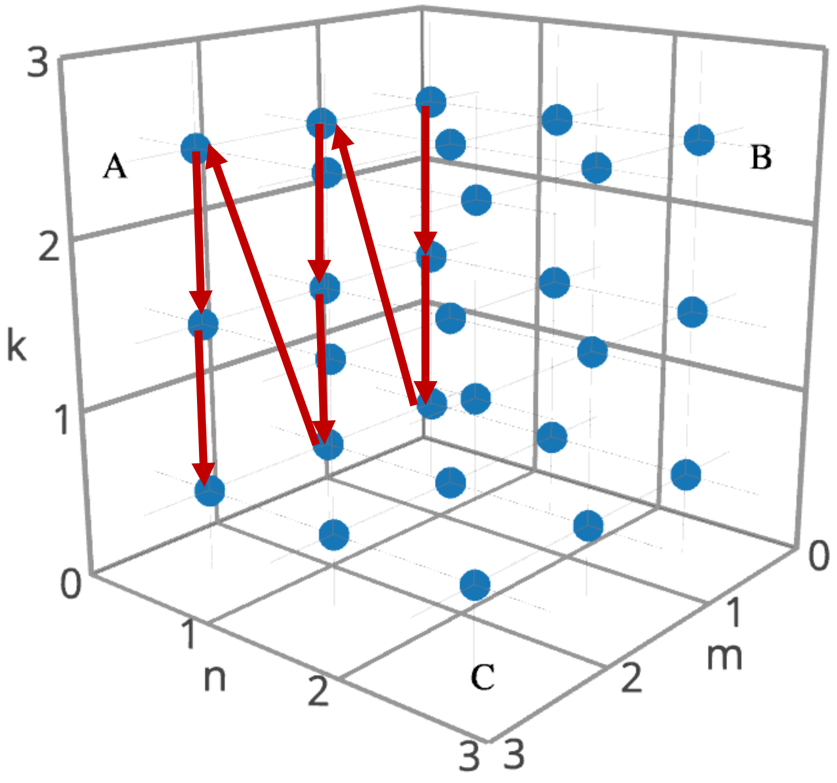
Fully local computation
- SRAM bandwidth is the key limitation

Level1: block dimensions



Not the larger the better!

Level2: block scheduling



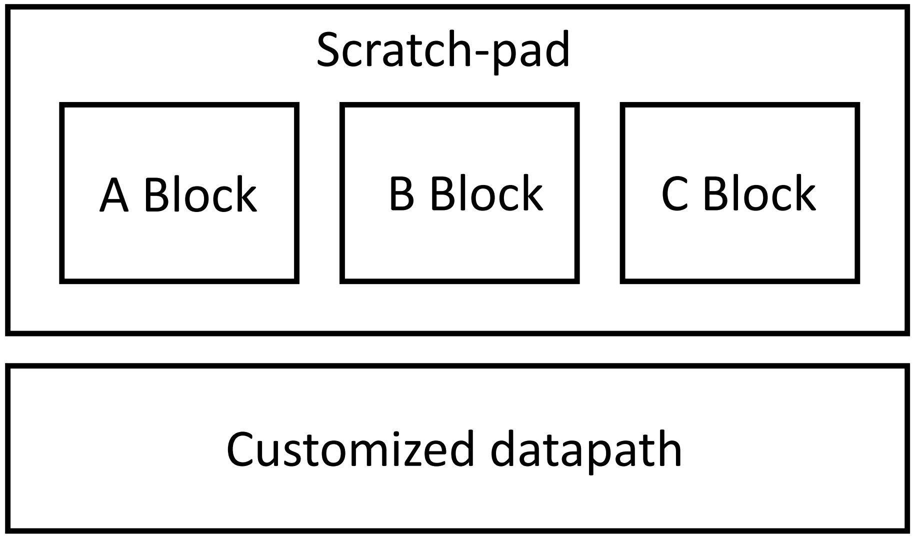
List scheduling + regularity

```
For $i$ : $0 \rightarrow num\_blk_m$
    For $j$ : $0 \rightarrow num\_blk_n$
        For $l$ : $0 \rightarrow num\_blk_k$
            executing macro node$(i, j, l)$
```

Six choices:

$i \rightarrow j \rightarrow l$, $j \rightarrow i \rightarrow l$,
$l \rightarrow i \rightarrow j$, $i \rightarrow l \rightarrow j$,
$l \rightarrow j \rightarrow i$, $j \rightarrow l \rightarrow i$

Fully local computation

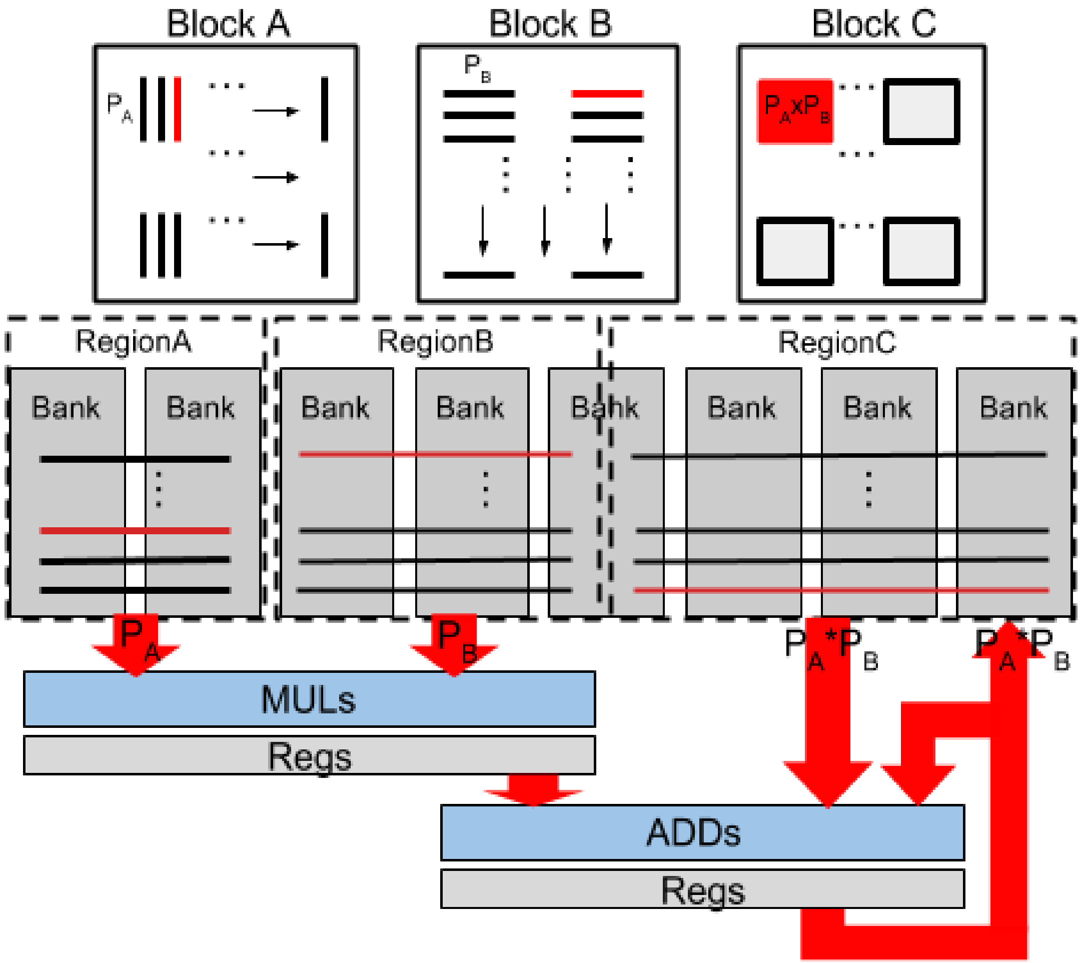


How to lay out data blocks?

How to design datapath?
How to decide its parallelism?

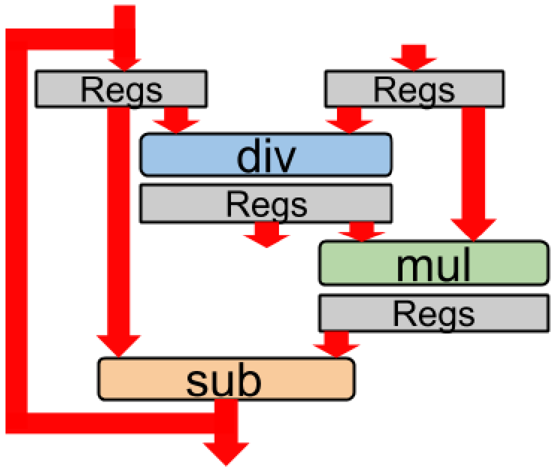
Level3: parallelism of local computation

Local computation of matrix multiply macro node

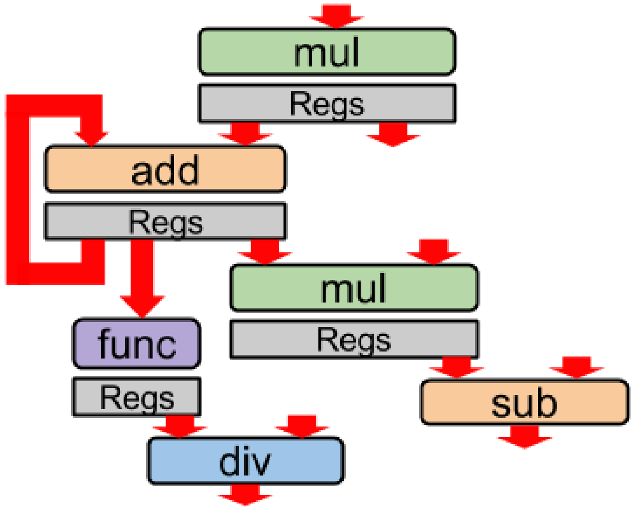


Datapath sharing

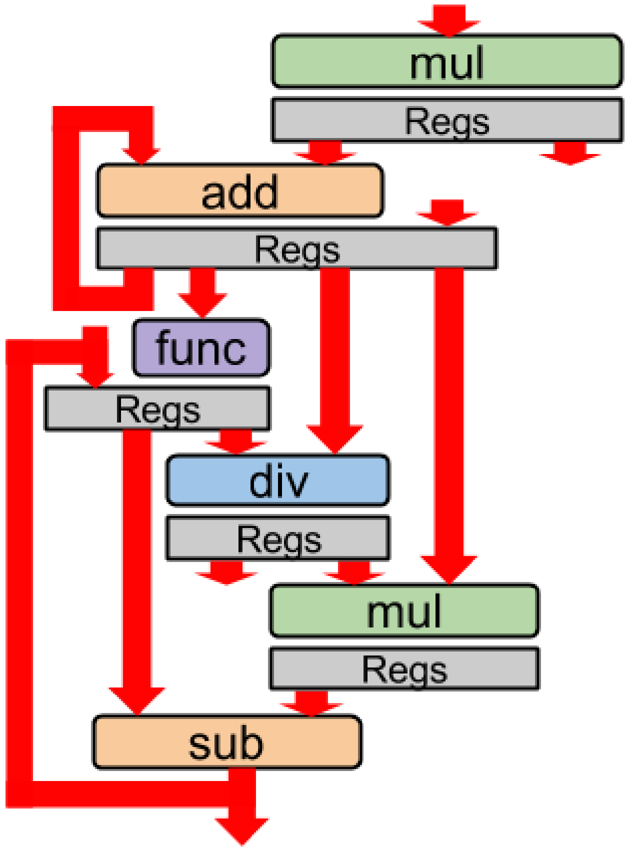
LU



QR



Datapath sharing for Matmul, LU and QR

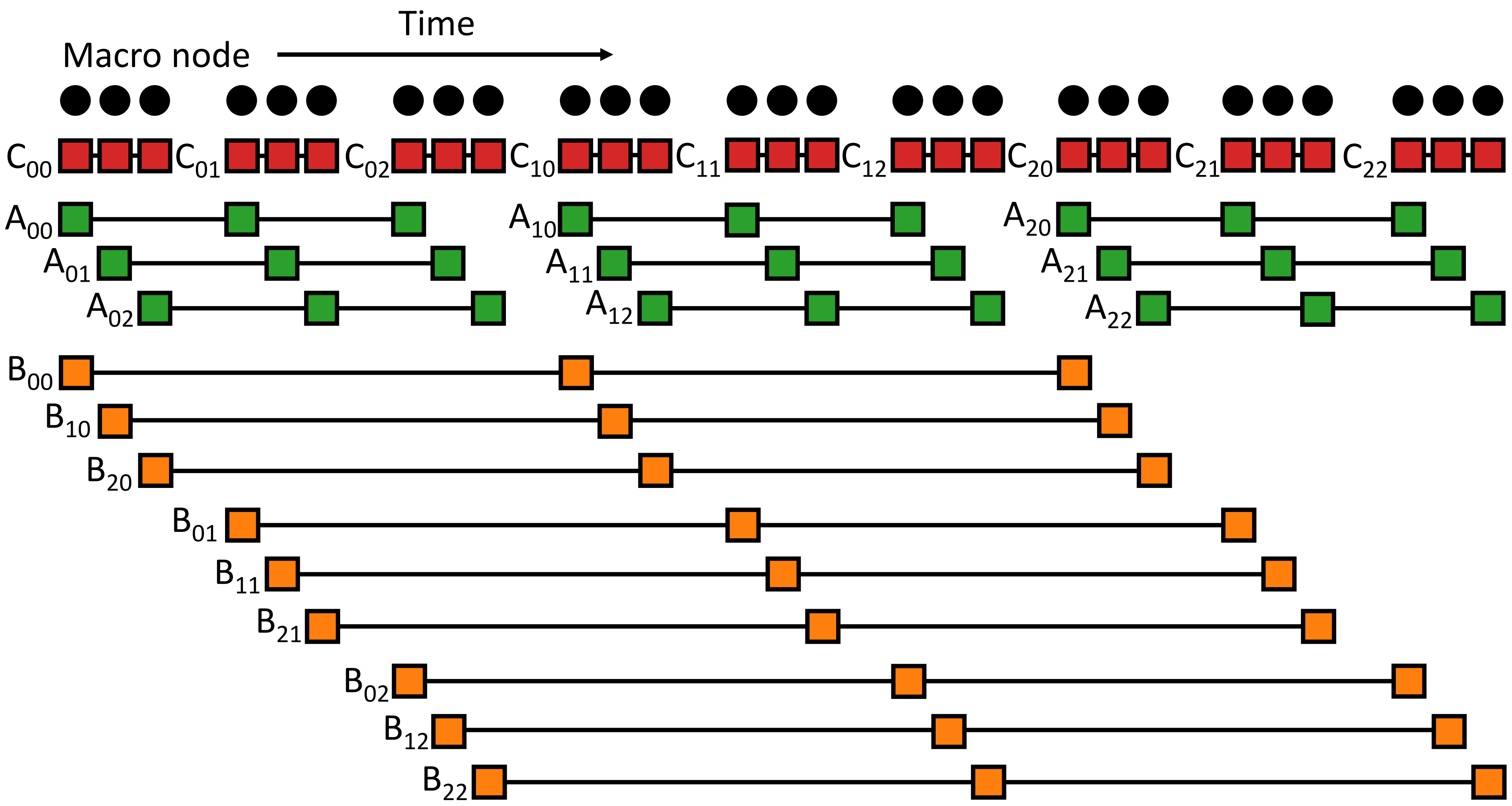


MemFlow

- Tunable parameters from three levels
  - Level1: block dimensions – blk_dim_i, blk_dim_j, blk_dim_l
  - Level2: block scheduling – six loop ordering
  - Level3: parallelism of local computing – P_A, P_B
- Mathematical modeling
  - DRAM accesses, SRAM accesses, cycles
- Integrated optimization

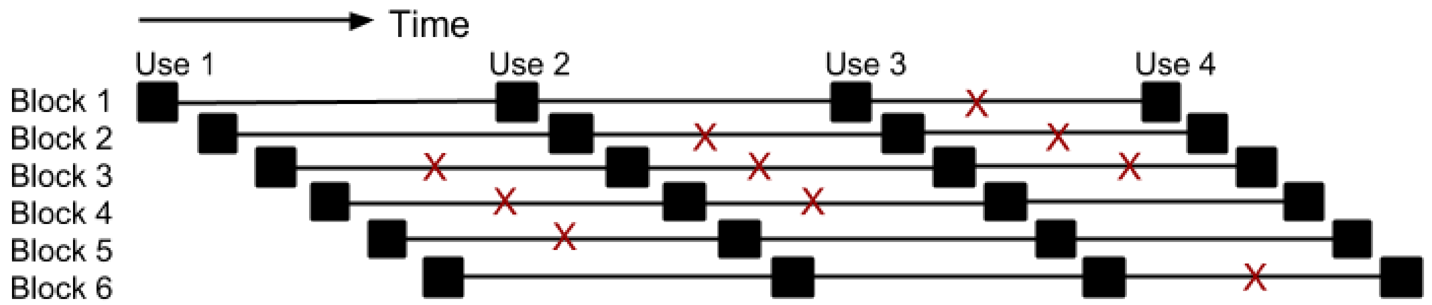
Modeling – DRAM accesses

Block reuse pattern in matrix multiply of loop ordering $i \rightarrow j \rightarrow l$



Modeling – DRAM accesses

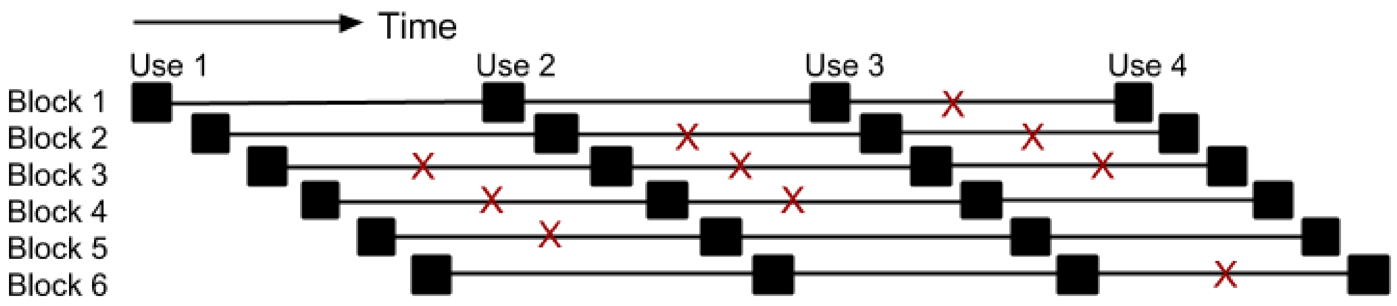
A group of alternatively used blocks



When SRAM is full, evict block based on optimal replacement policy

-- replace block with furthest next use

Modeling – DRAM accesses



#data evicted for each matrix

$$= \underline{\text{#reuse group}} * \underline{\text{#periods}} * \underline{\text{#blocks evicted per period}} * \underline{\text{block size}}$$

Input size, Block dimension, block ordering, scratchpad partitioning

Mathematical Modeling

- DRAM accesses:
  - input size, block size, loop ordering, memory partitioning factor

- SRAM Accesses:
  - input size, block dimension, SRAM partitioning

- Cycles:
  - input size, SRAM partitioning

MemFlow

- Tunable parameters from three levels
  - Level1: block dimensions – blk_dim_i, blk_dim_j, blk_dim_l
  - Level2: block scheduling – six loop ordering
  - Level3: parallelism of local computing – P_A, P_B
- Mathematical modeling
  - DRAM accesses, SRAM accesses, cycles
- Integrated optimization

Integrated Optimization

Optimization is sensitive to input size

For specific kernel and input size:

- Parameters:
  - Block dimensions, loop ordering, memory partitioning
- Objectives
  - Primary: weighted total of DRAM accesses and SRAM accesses
  - Secondary: cycles
- Optimization methods
  - Gradient descent to optimal magnitude
  - Parameter sweeping for global optimum
    - Worst-case search space:
      $O(N^{1.5} PlogP)$, N input size, P total SRAM ports
    - ~90 secs for P=16, N=1000

Results Comparison – cache reuse

Metric of evaluating data reuse:

Number of SRAM allocations:

- Counts each time a new word is stored in SRAM
- For cache, equals to (read misses + write misses)*cache line size

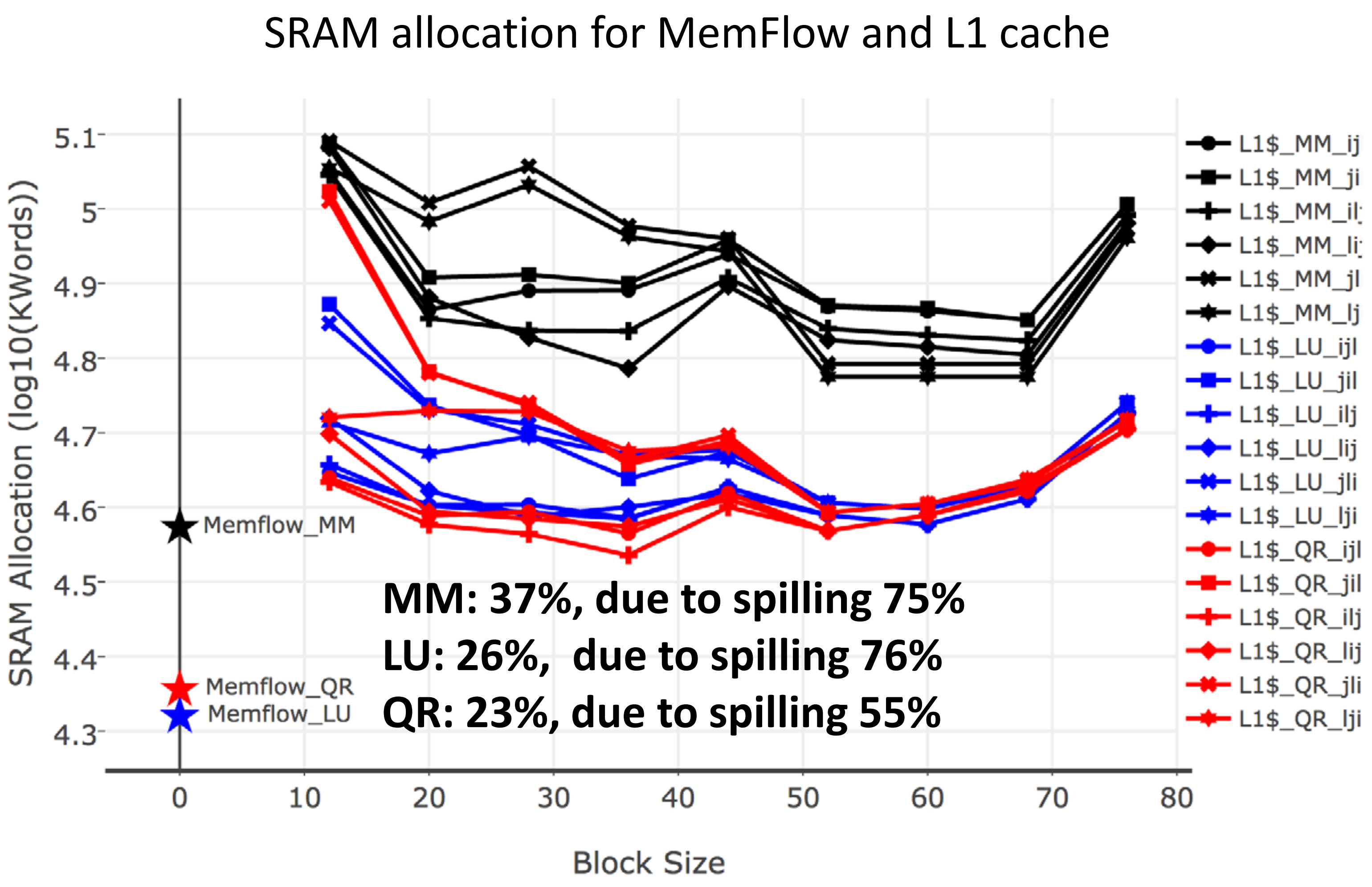
L1 cache:

32KB, 8-way associative, 64B cache line size

MemFlow:

8 4KB dual-port banks

Results Comparison – cache reuse



MM: 37%, due to spilling 75%
LU: 26%, due to spilling 76%
QR: 23%, due to spilling 55%

Results – GPU+cuBLAS

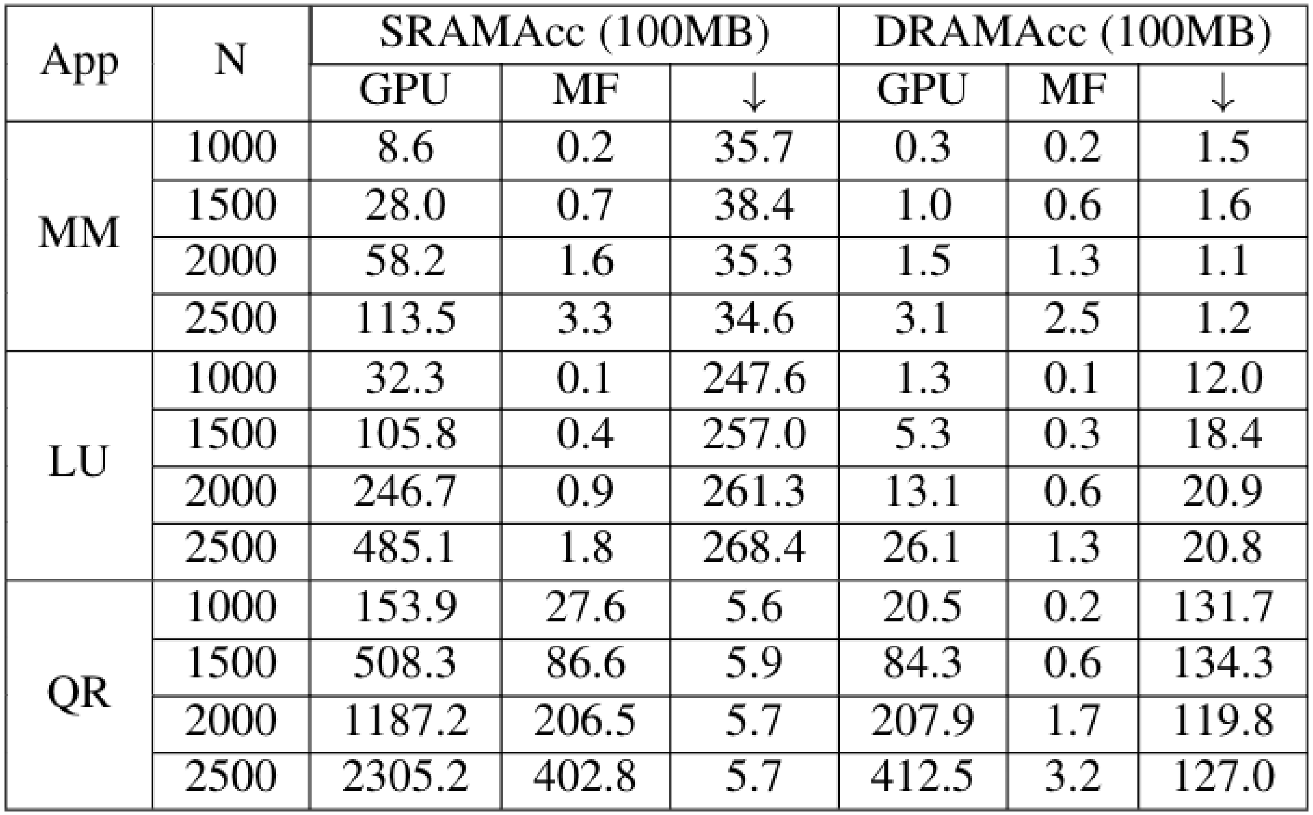
GPU: Nvidia Geforce 1080 GTX

- Total SRAM size: 4928 KB
  - L1 cache, shared memory, L2 cache
- cuBLAS library:
  - Matrix multiply: cublasSgemm
  - LU decomposition: cublasSgetrfBatched
  - QR decomposition: cublasSgeqrfBatched

MemFlow: 1232 4KB dual-port SRAM banks

Results – GPU+cuBLAS

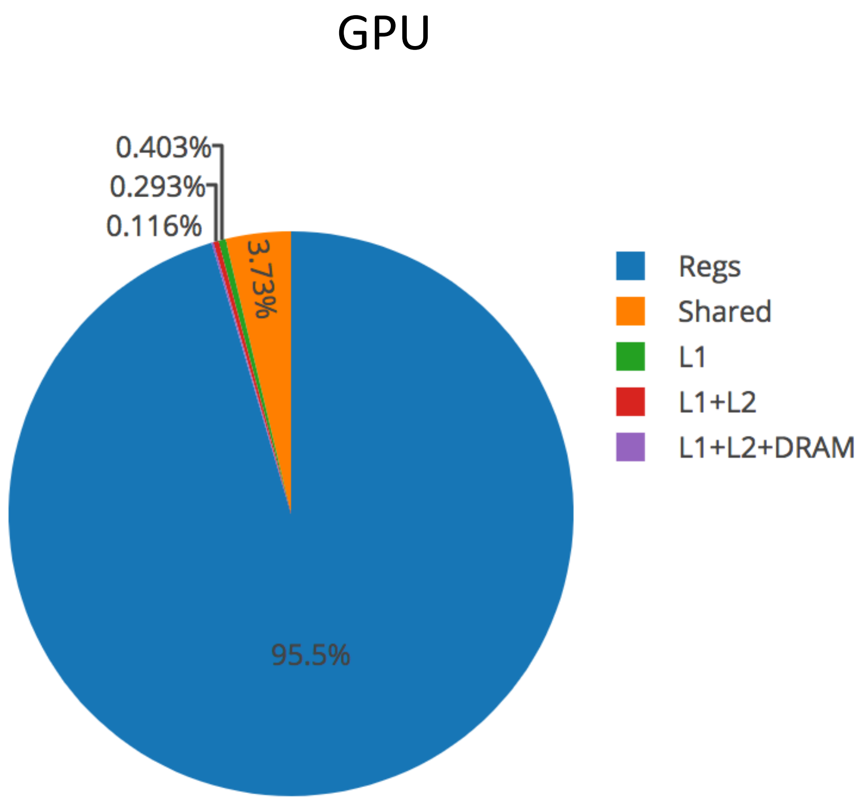
Improvements of SRAM and DRAM accesses.

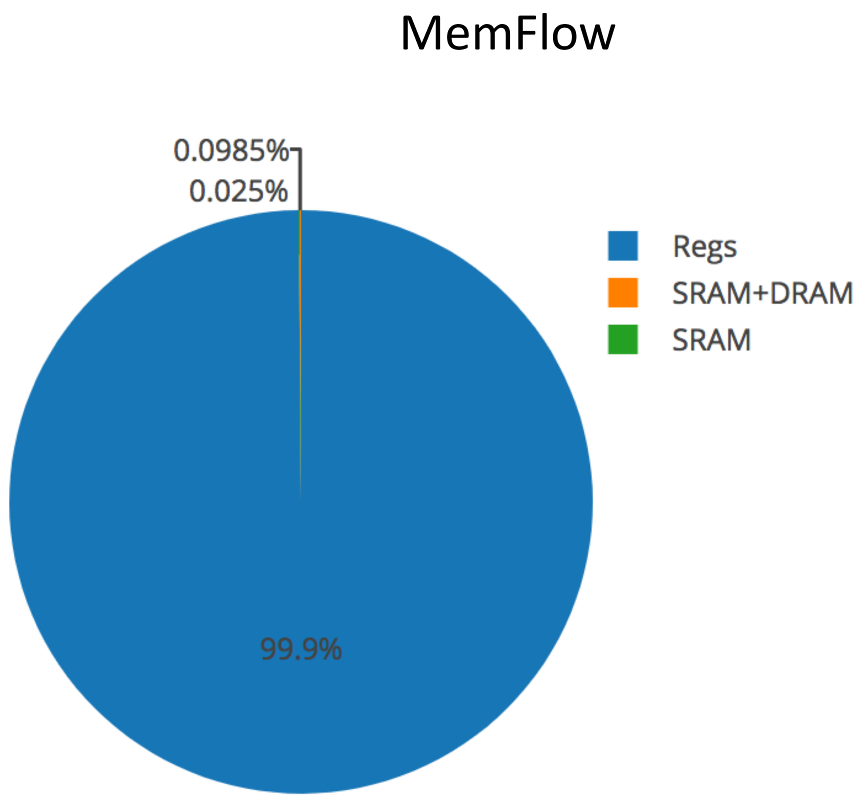


App	N	SRAMAcc (100MB)			DRAMAcc (100MB)		
		GPU	MF	↓	GPU	MF	↓
MM	1000	8.6	0.2	35.7	0.3	0.2	1.5
	1500	28.0	0.7	38.4	1.0	0.6	1.6
	2000	58.2	1.6	35.3	1.5	1.3	1.1
	2500	113.5	3.3	34.6	3.1	2.5	1.2
LU	1000	32.3	0.1	247.6	1.3	0.1	12.0
	1500	105.8	0.4	257.0	5.3	0.3	18.4
	2000	246.7	0.9	261.3	13.1	0.6	20.9
	2500	485.1	1.8	268.4	26.1	1.3	20.8
QR	1000	153.9	27.6	5.6	20.5	0.2	131.7
	1500	508.3	86.6	5.9	84.3	0.6	134.3
	2000	1187.2	206.5	5.7	207.9	1.7	119.8
	2500	2305.2	402.8	5.7	412.5	3.2	127.0

Results – GPU+cuBLAS

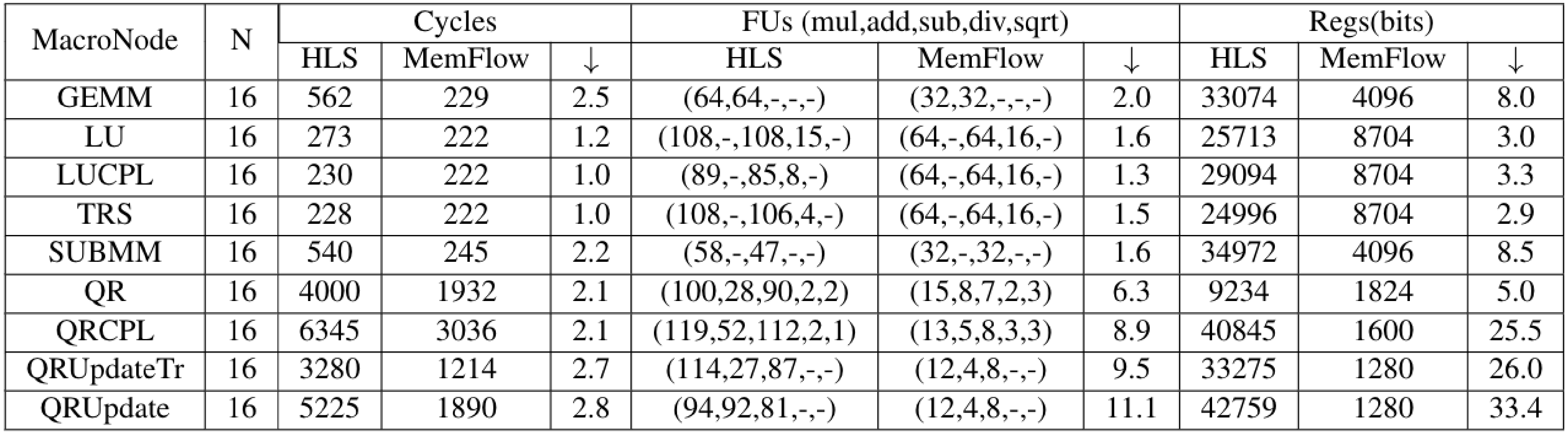
Matrix multiply data use distribution





Results – Vivado HLS

Datapath comparison with Vivado HLS



MacroNode	N	Cycles			FUs (mul,add,sub,div,sqrt)			Regs(bits)		
		HLS	MemFlow	↓	HLS	MemFlow	↓	HLS	MemFlow	↓
GEMM	16	562	229	2.5	(64,64,-,-,-)	(32,32,-,-,-)	2.0	33074	4096	8.0
LU	16	273	222	1.2	(108,-,108,15,-)	(64,-,64,16,-)	1.6	25713	8704	3.0
LUCPL	16	230	222	1.0	(89,-,85,8,-)	(64,-,64,16,-)	1.3	29094	8704	3.3
TRS	16	228	222	1.0	(108,-,106,4,-)	(64,-,64,16,-)	1.5	24996	8704	2.9
SUBMM	16	540	245	2.2	(58,-,47,-,-)	(32,-,32,-,-)	1.6	34972	4096	8.5
QR	16	4000	1932	2.1	(100,28,90,2,2)	(15,8,7,2,3)	6.3	9234	1824	5.0
QRCPL	16	6345	3036	2.1	(119,52,112,2,1)	(13,5,8,3,3)	8.9	40845	1600	25.5
QRUpdateTr	16	3280	1214	2.7	(114,27,87,-,-)	(12,4,8,-,-)	9.5	33275	1280	26.0
QRUpdate	16	5225	1890	2.8	(94,92,81,-,-)	(12,4,8,-,-)	11.1	42759	1280	33.4

Future Work

- More efficient optimization methods
- Optimizing data scheduling for a flow of computation kernels
- Verilog validation