Matrix Extension Modelling

Miles Rusch, Dongjie Xie, Alex Rucker

Tenstorrent

Outline

Introduction to Matrix Multiplication (GEMM) Kernels

- Overview of BLAS decomposition of large GEMM into outer product micro-kernels

Outer Product Instruction Extension

- Matrix ISA extension targets outer product GEMM kernels
- Functional unit layout

Modeling Resources and Performance

- **Model Inputs:** datatype, memory latency, matrix dimensions, functional unit size, etc
- **Estimated Resource Requirements:** memory bandwidth, instruction decode bandwidth, matrix register file (MRF) capacity, multiply-accumulate (MACC) logic size
- **Estimated Performance Metrics:** Operations per Cycle (OPC), Kernel Latency, MMU Utilization

Extending Ocelot

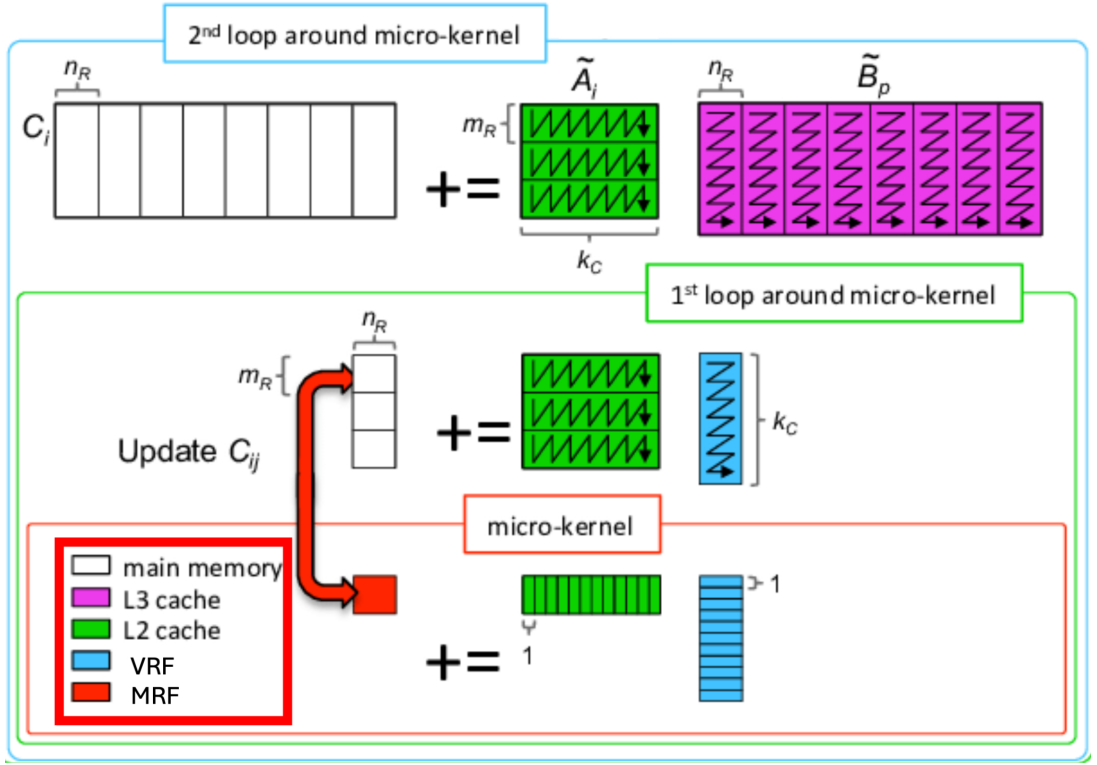
- RTL Implementation of RISC-V Vector Extension (RVV)
- Extending RVV with Matrix ISA

Introduction to Matrix Multiplication (GEMM) Kernels

BLAS libraries decompose large GEMMs into a series of micro-kernels.

The outer product decomposition achieves (asymptotically) optimal memory data re-use

The outer product accumulate micro-kernel (**opacc**) can be executed with just one new instruction (by grouping vector registers, eg LMUL>1)



[1] F. G. Van Zee and T. M. Smith, “Implementing high-performance complex matrix multiplication,” ACM Transactions on Mathematical Software, 2016

Memory Efficiency

Memory Efficiency

OPACC operation:

m_c += $v_a \otimes v_b$

For vector inputs with *vl* of elements per vector, opacc performs *vl*^2 MACC operations, and *2vl* memory operations.

Thus the operational intensity (OI) or number of MACC operations per memory operation is,

OI = vl/2

OI improves as the vector length increases

	B_0	B_1	...	B_{ml}
A_0	C_{00} += $A_0 * B_0$	C_{01} += $A_0 * B_1$	...	$C_{0,ml}$ += $A_0 * B_{ml}$
A_1	C_{10} += $A_1 * B_0$	C_{11} += $A_1 * B_1$	...	$C_{1,ml}$ += $A_1 * B_{ml}$
...	...	...	...	...
A_{vl}	$C_{vl,0}$ += $A_{vl} * B_0$	$C_{vl,0}$ += $A_{vl} * B_0$	...	$C_{vl,ml}$ += $A_{vl} * B_{ml}$

Outer Product Architecture Extension

Add matrix register state (MRF) to RVV facilitate a scalable vector length for OPACC operations

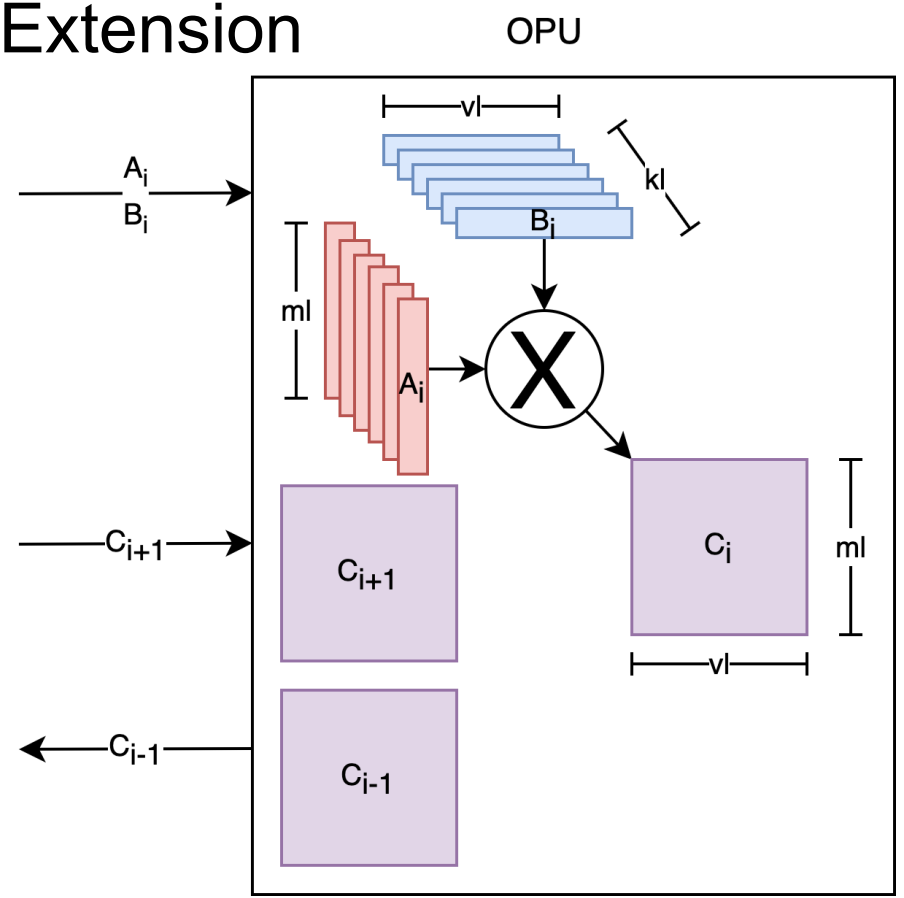
Add vmv instructions that move vectors between the VRF nad MRF:

vmv.m.v md vs

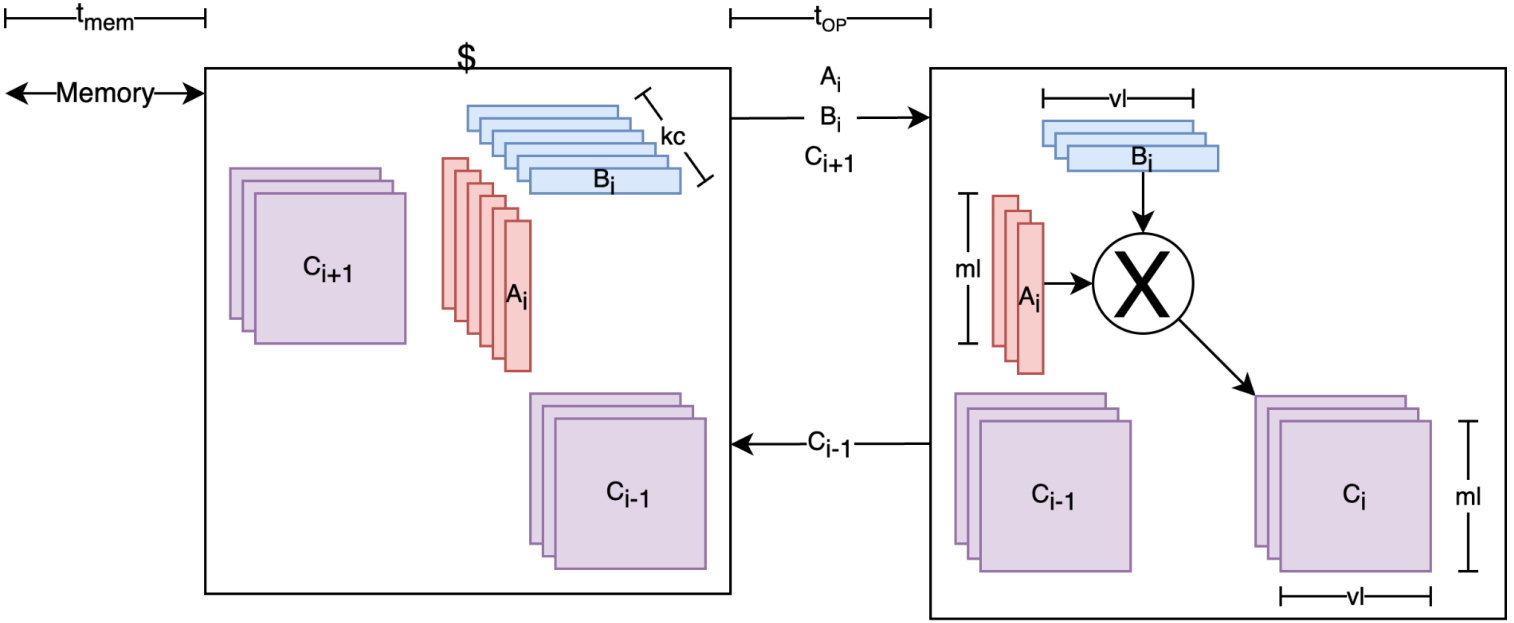
vmv.v.m vd ms

Add oppac instruction:

opacc md vs1 vs2



Performance Model



Model inputs:

- memory latency,
- OP functional unit latency,
- Datatype bit-width
- Scalable parameters vl, ml
- Software kernel matrix dimensions

Model output performance:

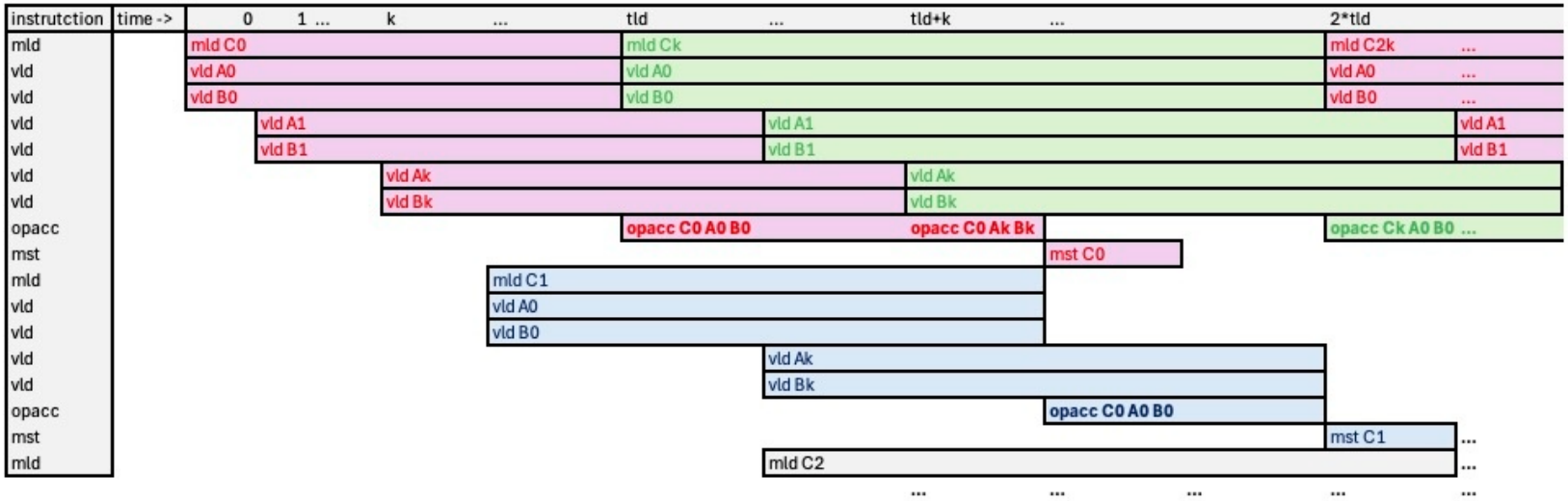
- MACC operations per cycle
- OPU latency
- utilization
- instructions per cycle

Model output resource requirements:

- MRF capacity
- L2 cache capacity
- Memory bandwidth
- CMOS gate count

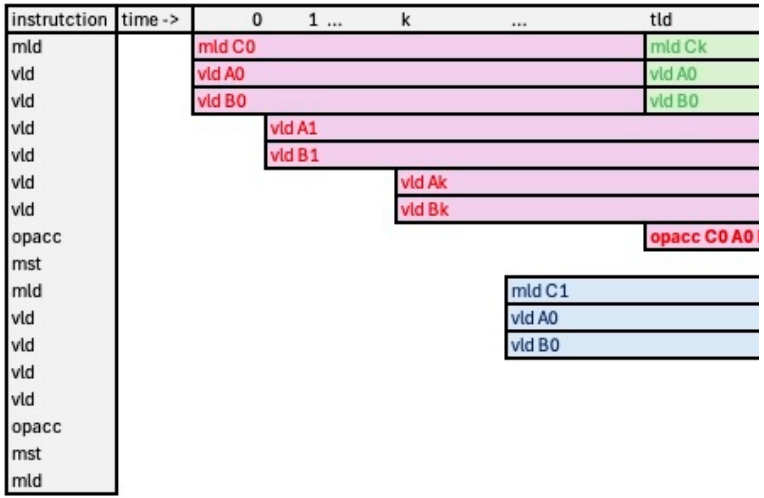
Hiding Memory Latency

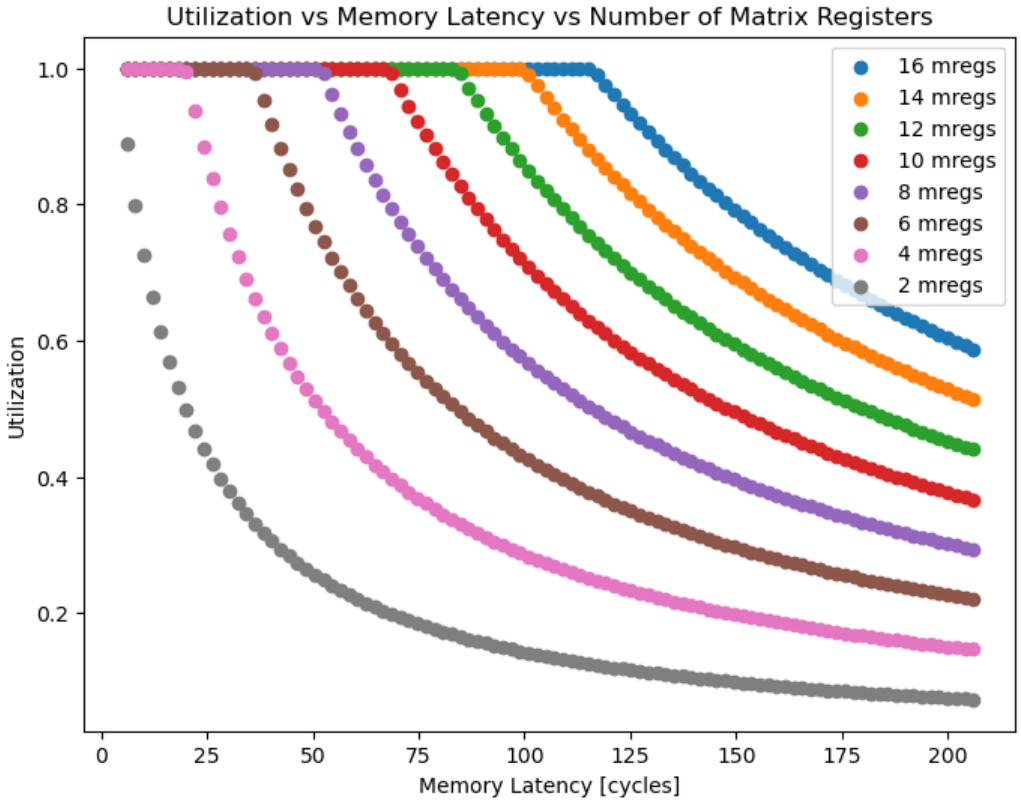
- To hide memory latency, multiple load requests must be sent to memory
- The number of requests depends on the ratio of memory latency to opacc latency



Memory Latency

If there aren't sufficient matrix registers to hide memory latency, the OPU utilization decreases.

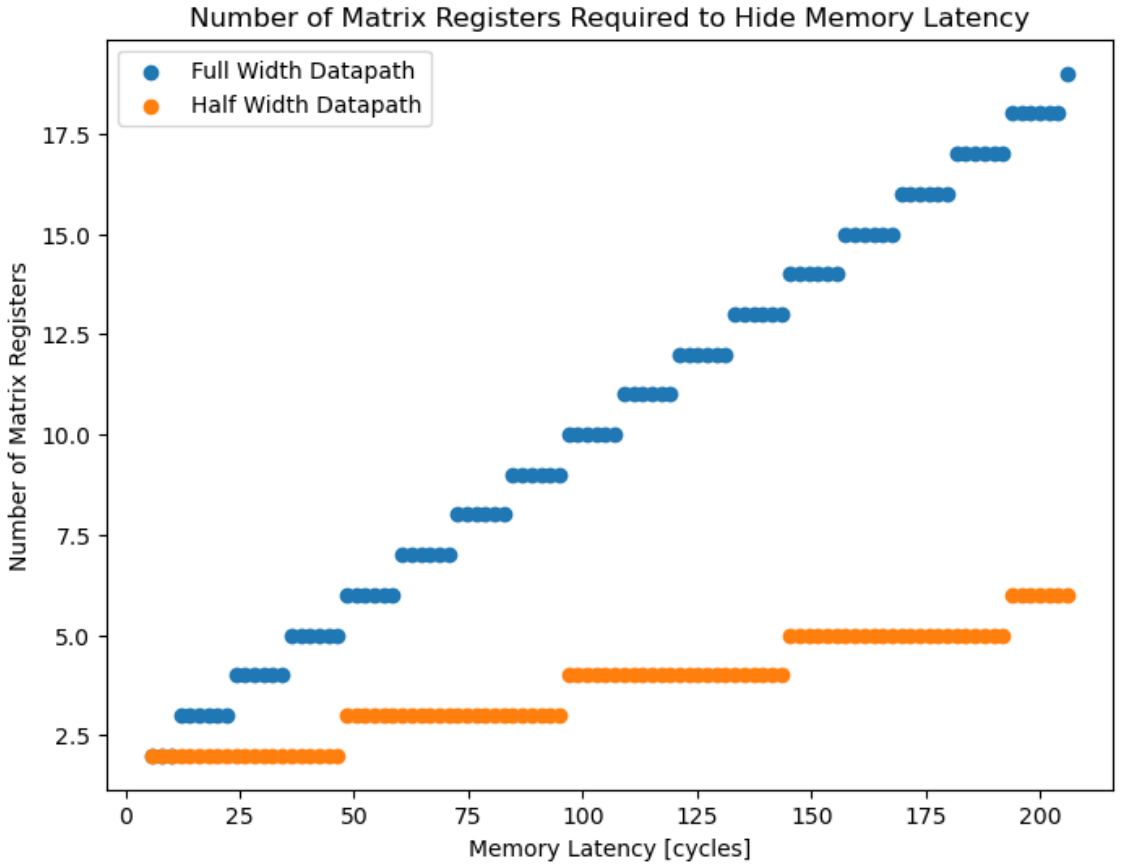




Memory Latency

If there aren't sufficient matrix registers to hide memory latency, the OPU utilization decreases.

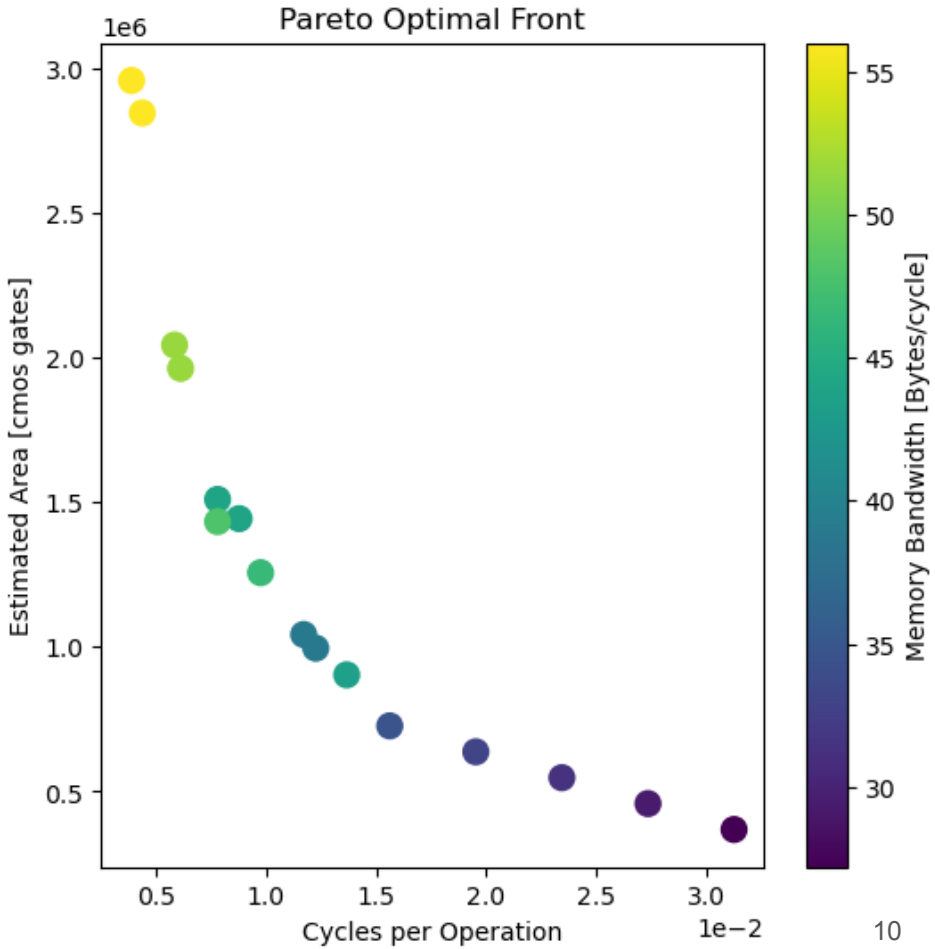
If a higher memory latency must be tolerated, the latency of the OPACC operation can be increased



Pareto-Optimal Front

To find optimal design points in a complex space with many tradeoffs, we find the pareto-optimal front given three performance metrics,

- memory bandwidth,
- logic gate count (both macc logic and registers),
- cycles per macc operation (1/ OPS)



Optimal Parameters

vlB:

vector length [Bytes]

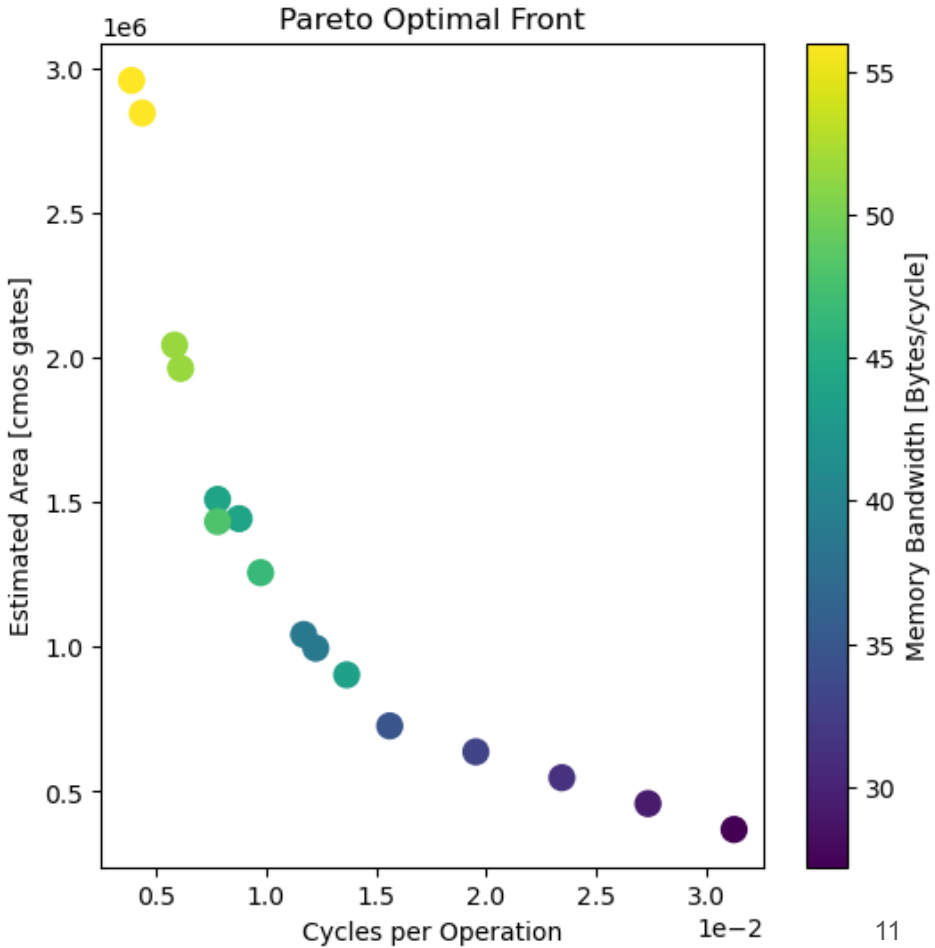
mlB:

matrix length [Bytes]

num_mregs:

matrix register count

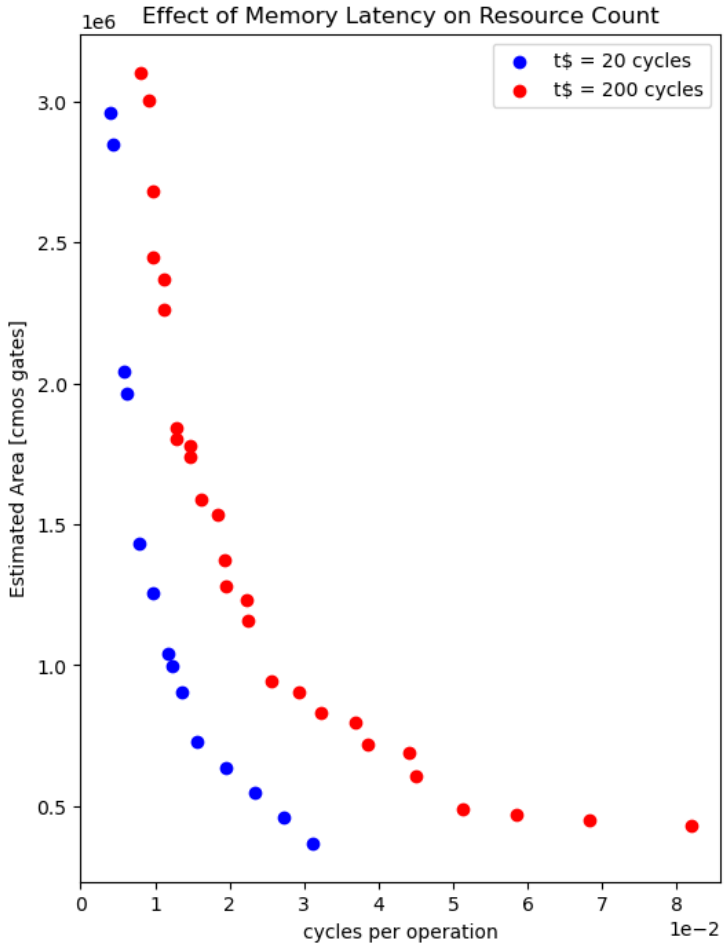
vlB	mlB	num_mregs
16.0	16.0	4
	20.0	4
	24.0	4
	28.0	4
	32.0	4
	44.0	4
		6
	64.0	4
		6
32.0	20.0	4
	28.0	4
	32.0	4
	44.0	4
		6
	64.0	4
		6



Memory Latency Shifts Pareto-Front

Increasing memory latency:

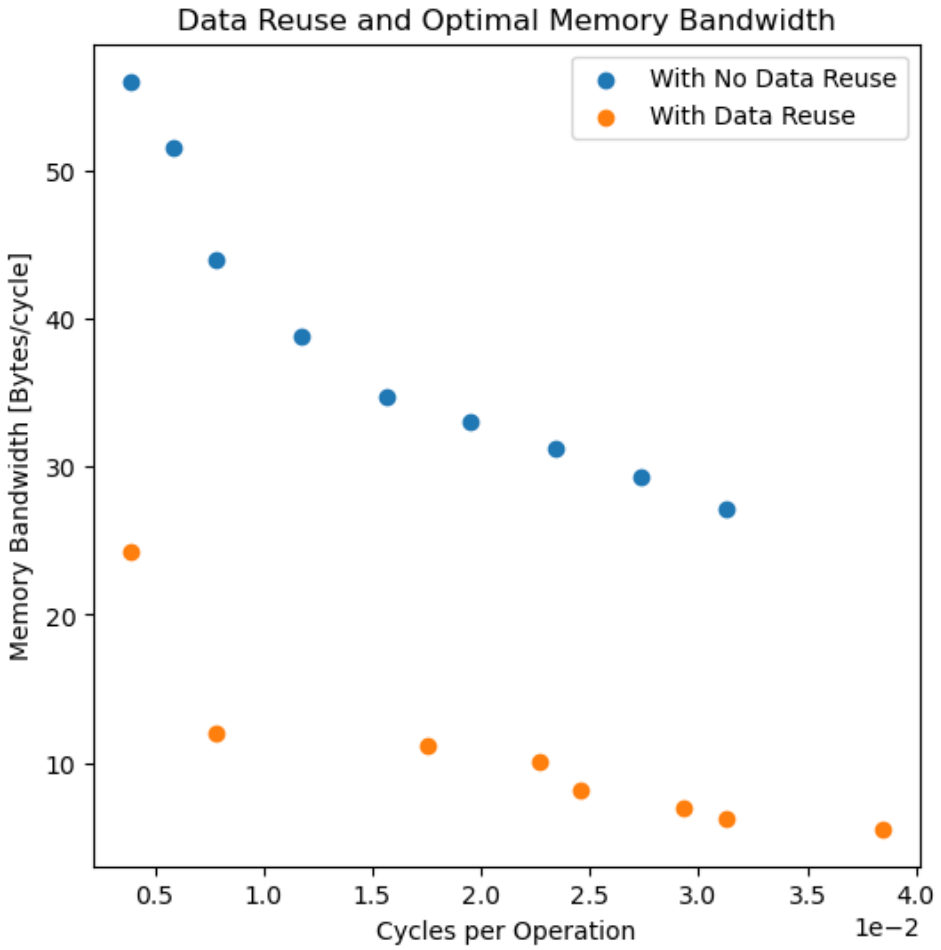
- moves the Pareto-front further from optimal, because it:
  - requires more MRF capacity, which limits performance given constrained resources



Data Reuse

If GEMM kernel has sufficiently large matrices, data can be reused from cache by the kernel

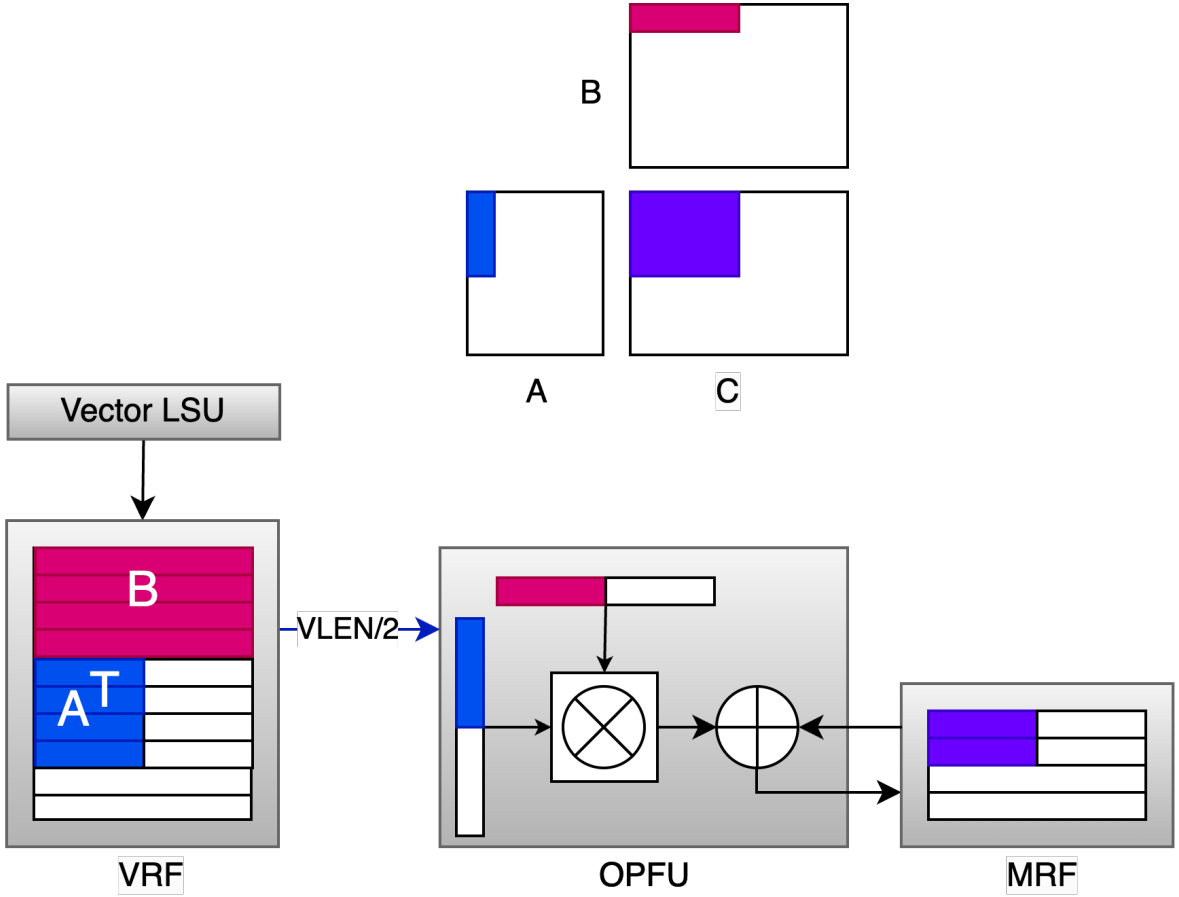
We see that we can significantly reduce memory bandwidth requirements for large matrices



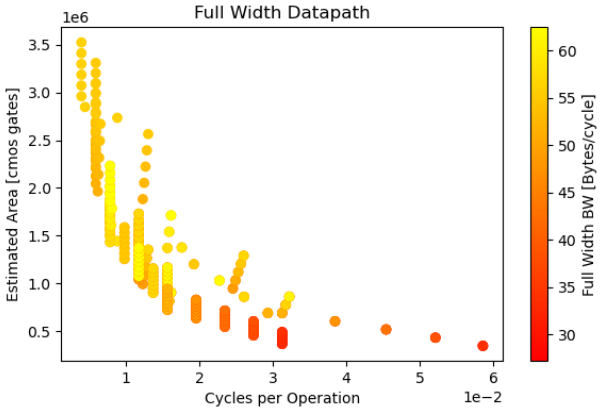
Half-Width Datapath

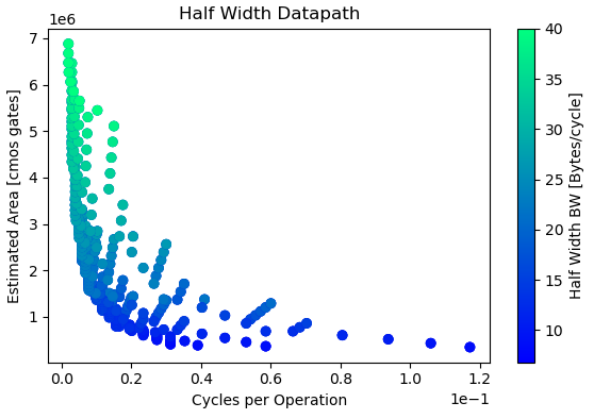
Narrowing the datapath width by half:

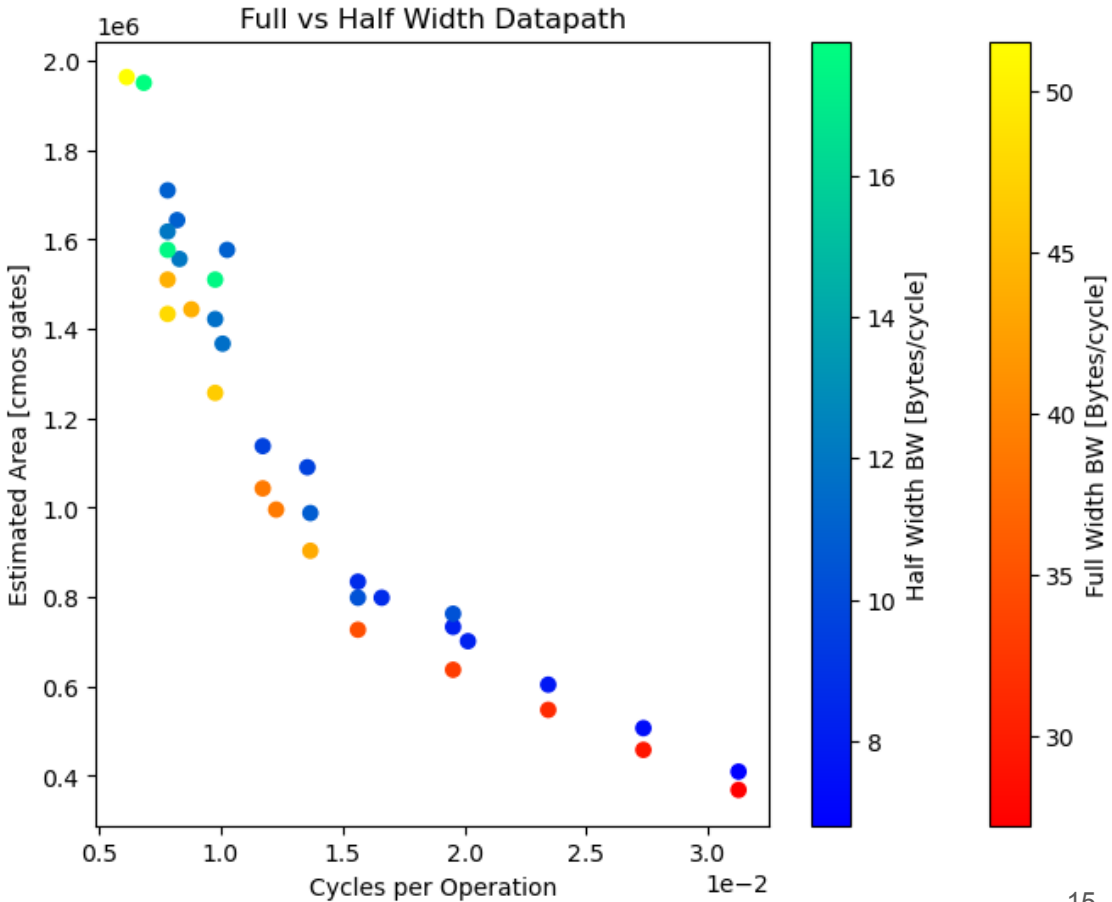
- Reduces the functional unit throughput by a factor of four, and
- Increases the latency by a factor of four, but
- Alleviates memory and VRF bandwidth requirements



Half Width Data Path







Half Width Data Path

Memory bandwidth can be reduced by reducing the width of the datapath relative to the vector length

This trades off reduced OPS for reduced memory bandwidth