Imperial College London

EPSRC
Engineering and Physical Sciences Research Council

GiMMiK: Generating Bespoke Matrix Multiplication Kernels

F.D. Witherden, **B.D. Wozniak, F.P Russel, P.E. Vincent, P.H.J Kelly**

Department of Aeronautics & Department of Computer Science
Imperial College London

Motivation

- Computational fluid dynamics (CFD) is the bedrock of several high-tech industries.
- Desire amongst practitioners to perform **unsteady**, **scale resolving** simulations within the vicinity of **complex geometries**.
- Not currently viable with current generation CFD.

Motivation

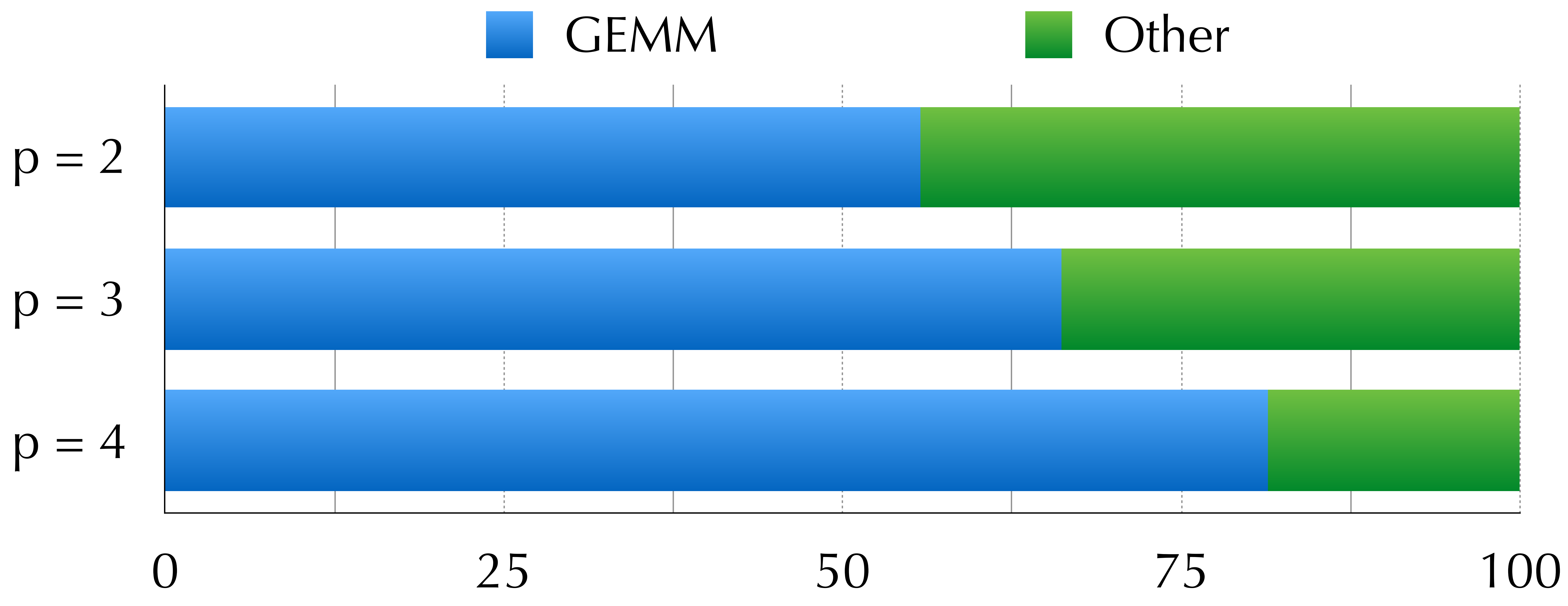
- Our solution **PyFR**—the Py being for **Python**.
- Runs on clusters of NVIDIA GPUs.
- Uses the **flux reconstruction** (FR) approach to solver the compressible **Navier-Stokes** equations on mixed unstructured grids in **2D/3D**.

Motivation

- FR has a variety of desirable numerical properties:
  - completely **explicit**;
  - **halo type** exchanges between elements;
  - majority of operations can be cast as large **matrix-matrix multiplications**.

Motivation

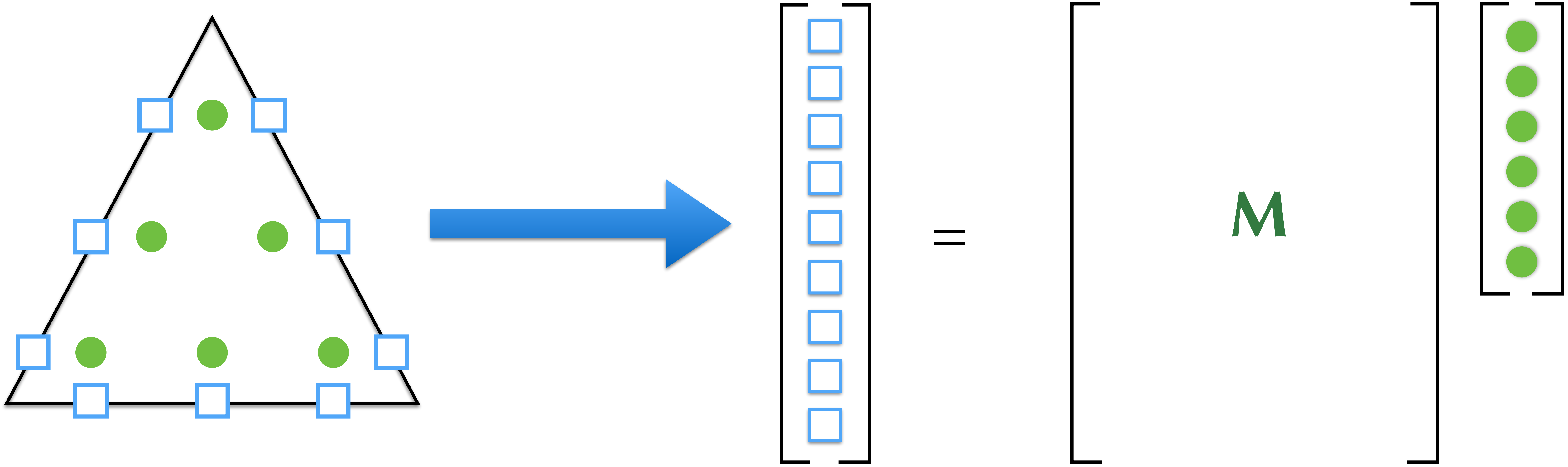
- Runtime of PyFR is hence dominated by calls to **GEMM**.



- To speed-up PyFR we therefore need to **beat cuBLAS**!

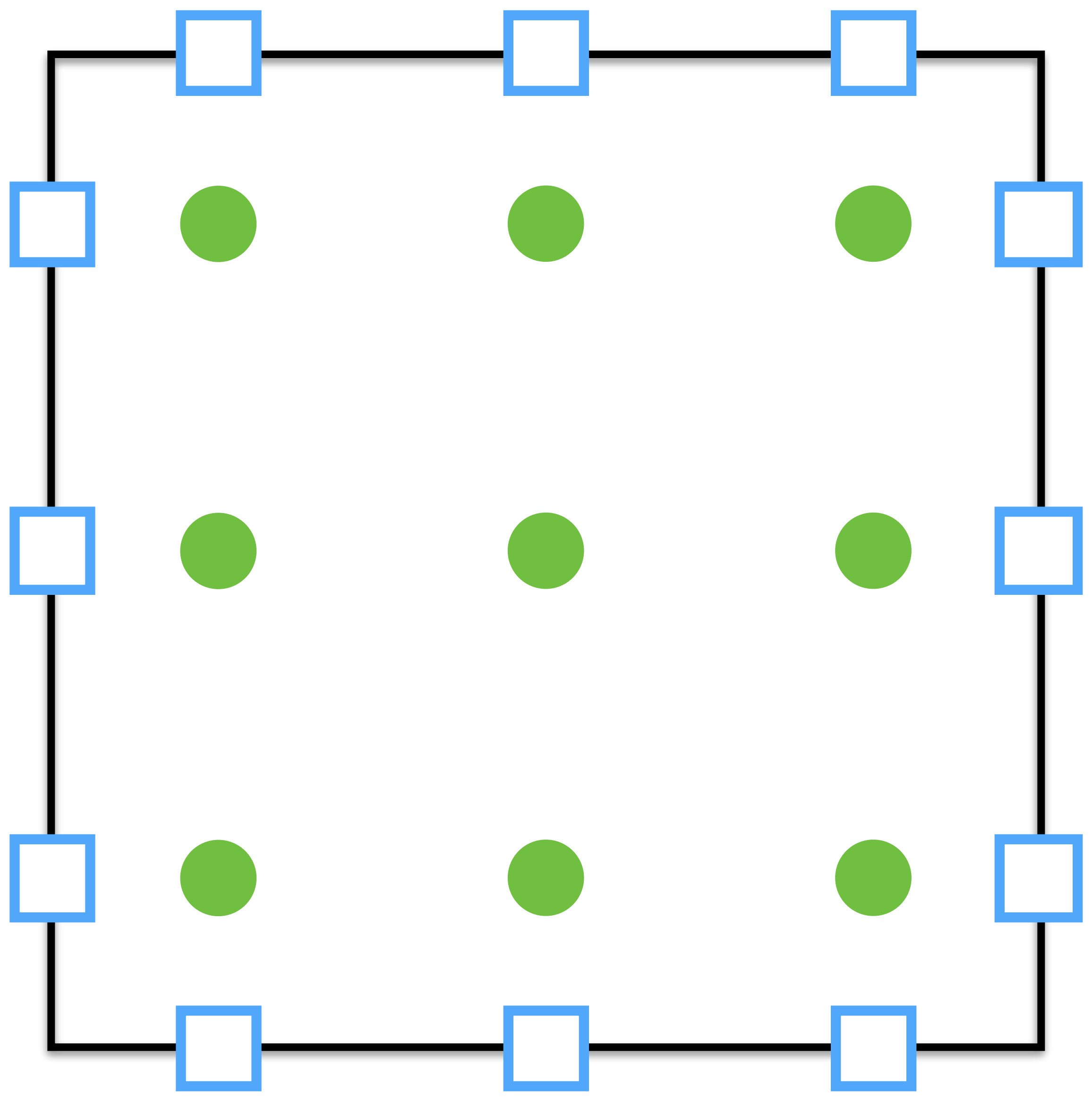
Motivation

- Have data at ● and want to interpolate to □ .



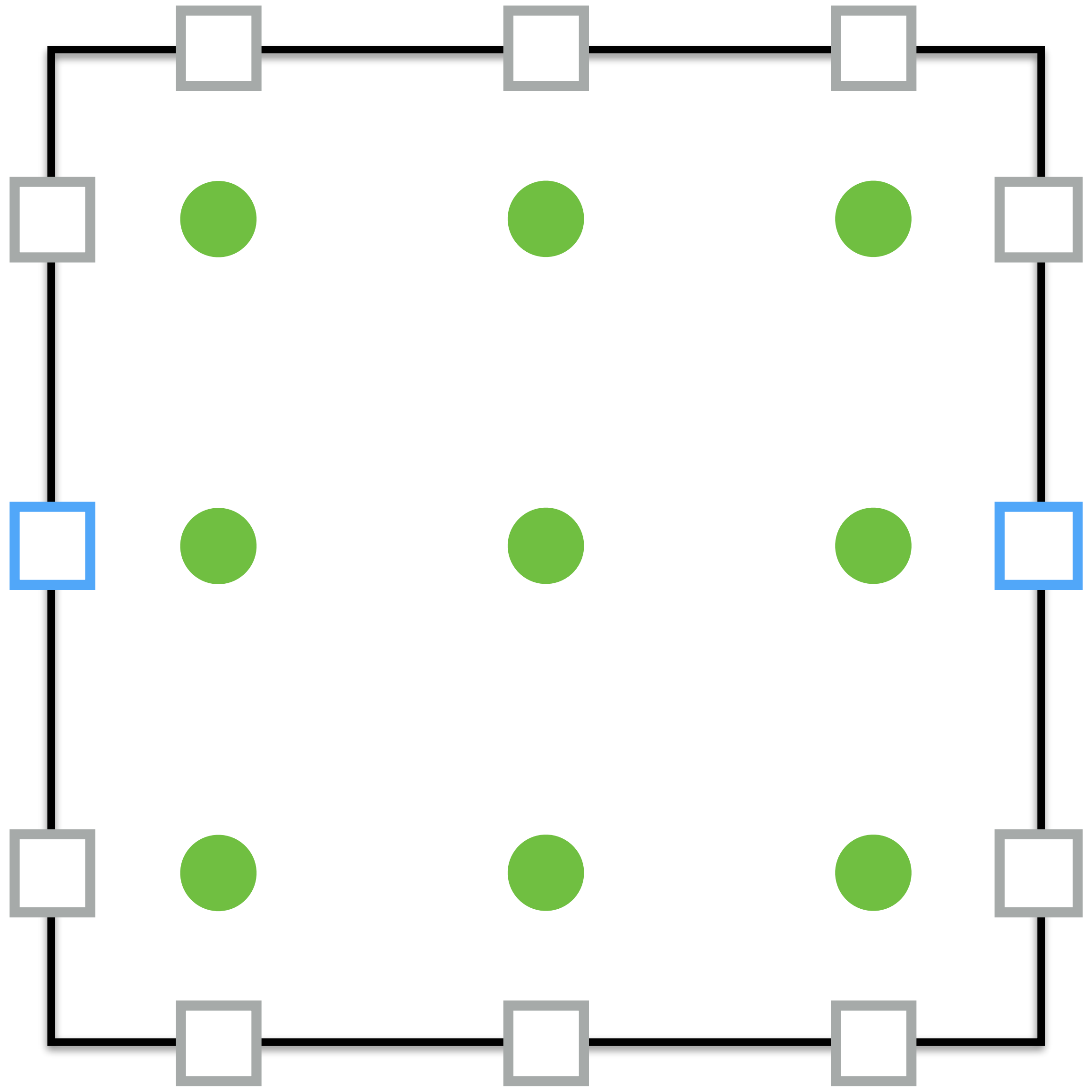
Motivation

- In a **tensor product** element points can **align**.



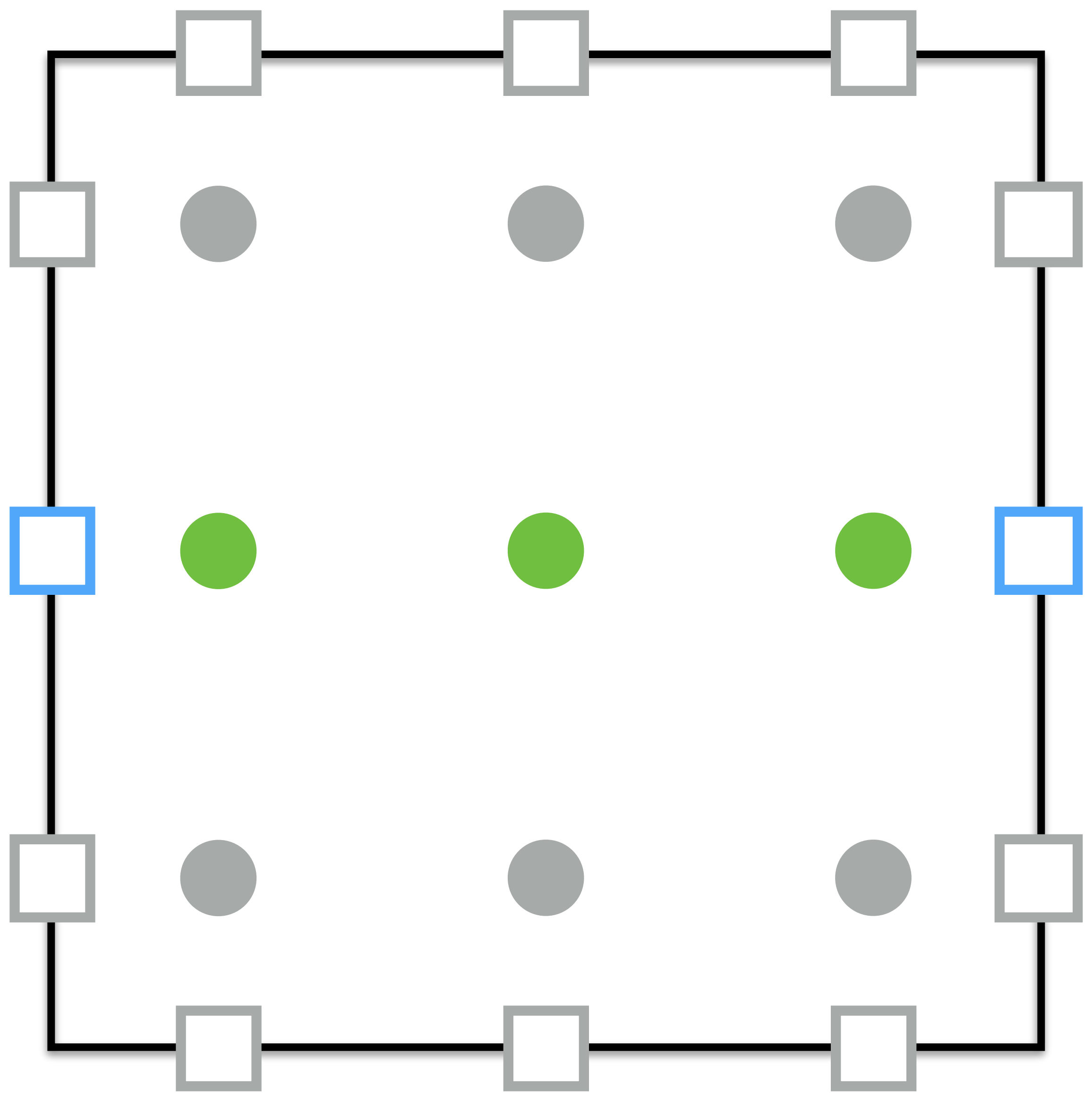
Motivation

- Consider the two highlighted blue points.
- These **line up** with the three interior points.



Motivation

- Hence, the entires in **M** for these two points only depend on **some** of the interior points.
- This introduces **sparsity** into **M**.



Putting the **G** in GEMM

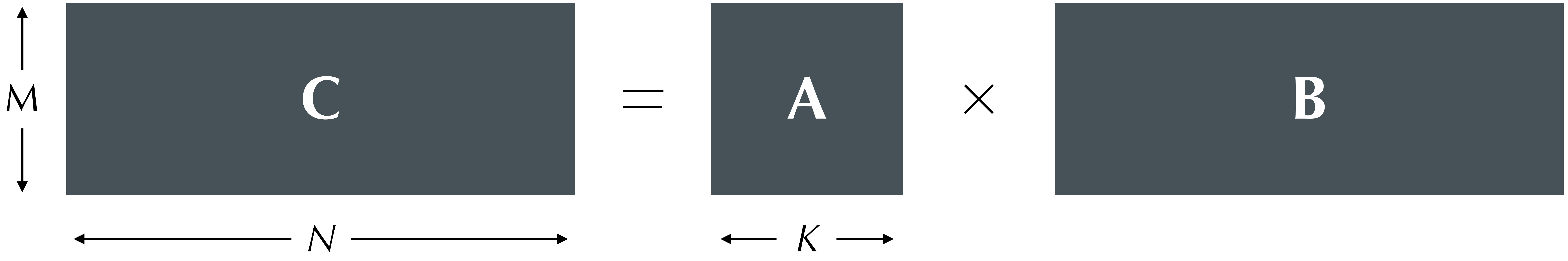
- The G in GEMM stands for **general**.

$$\mathbf{C} = \mathbf{A} \times \mathbf{B}$$

- But in the case of FR we know things BLAS doesn't.

What We Know: Shape

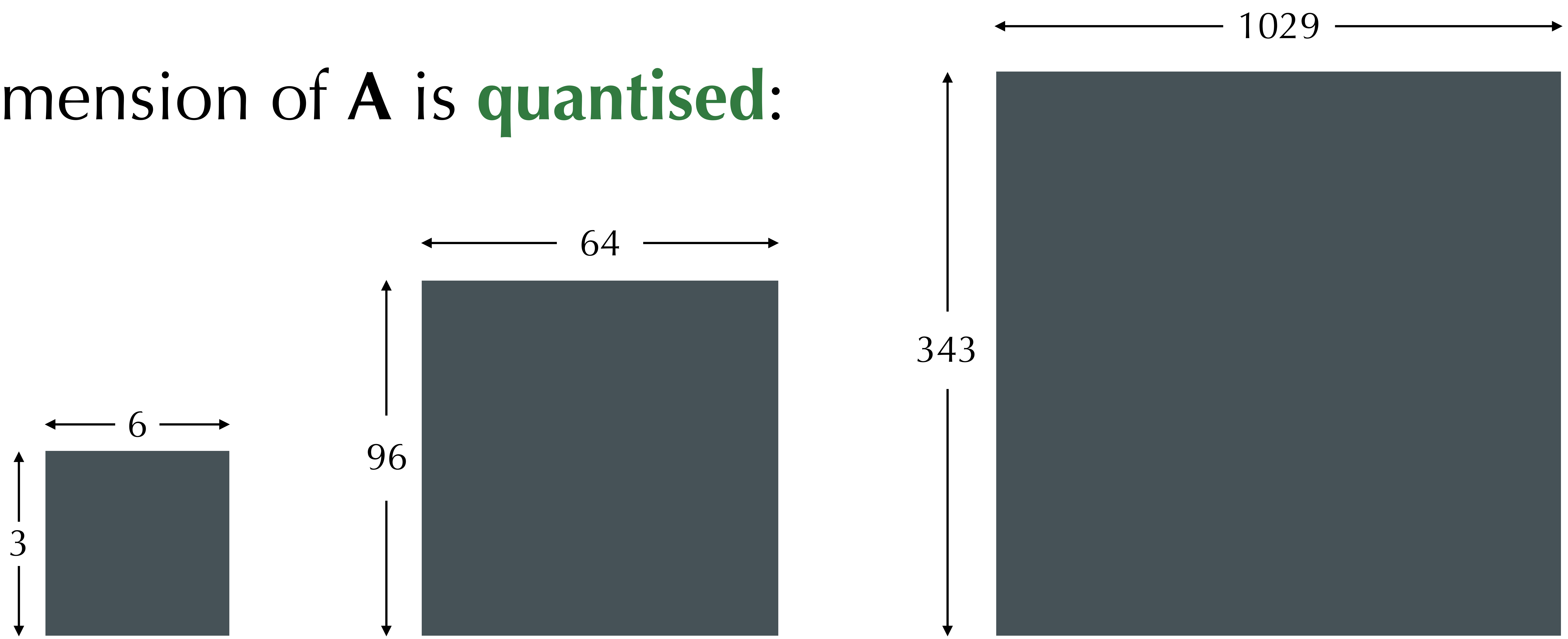
- Multiplications are of the **block-by-panel** variety:



- where $N \sim 10^5$ and $N \gg (M, K)$.

What We Know: Size

- Dimension of **A** is **quantised**:



- Around ~100 different sizes occur in practise.

What We Know: Values

- Entries of **A** are **constant**:

5	2	0	1	0	7	6	2	5	8	0	3	9	0	0	2	5	3
0	1	0	5	7	0	6	0	1	8	4	0	5	3	9	2	1	8
0	0	0	0	8	4	3	9	0	4	3	0	0	0	9	0	1	4
4	4	5	8	7	1	4	6	3	0	0	0	0	7	9	2	1	8
3	5	1	2	0	7	4	6	0	9	3	5	0	4	1	2	6	1
9	0	5	0	2	9	5	8	7	1	4	0	0	0	1	2	6	2
4	3	6	5	0	0	2	0	0	3	0	0	2	8	7	4	6	9
4	0	0	5	7	7	0	9	0	8	0	2	5	3	0	2	1	8
9	0	0	8	4	0	2	6	7	3	0	0	0	8	7	4	6	3
7	0	9	0	8	7	6	2	0	8	0	0	0	1	4	0	5	4
3	5	0	2	0	0	0	6	9	1	0	4	2	5	3	4	6	9
0	8	9	8	8	5	2	7	4	2	0	0	0	9	0	8	1	4

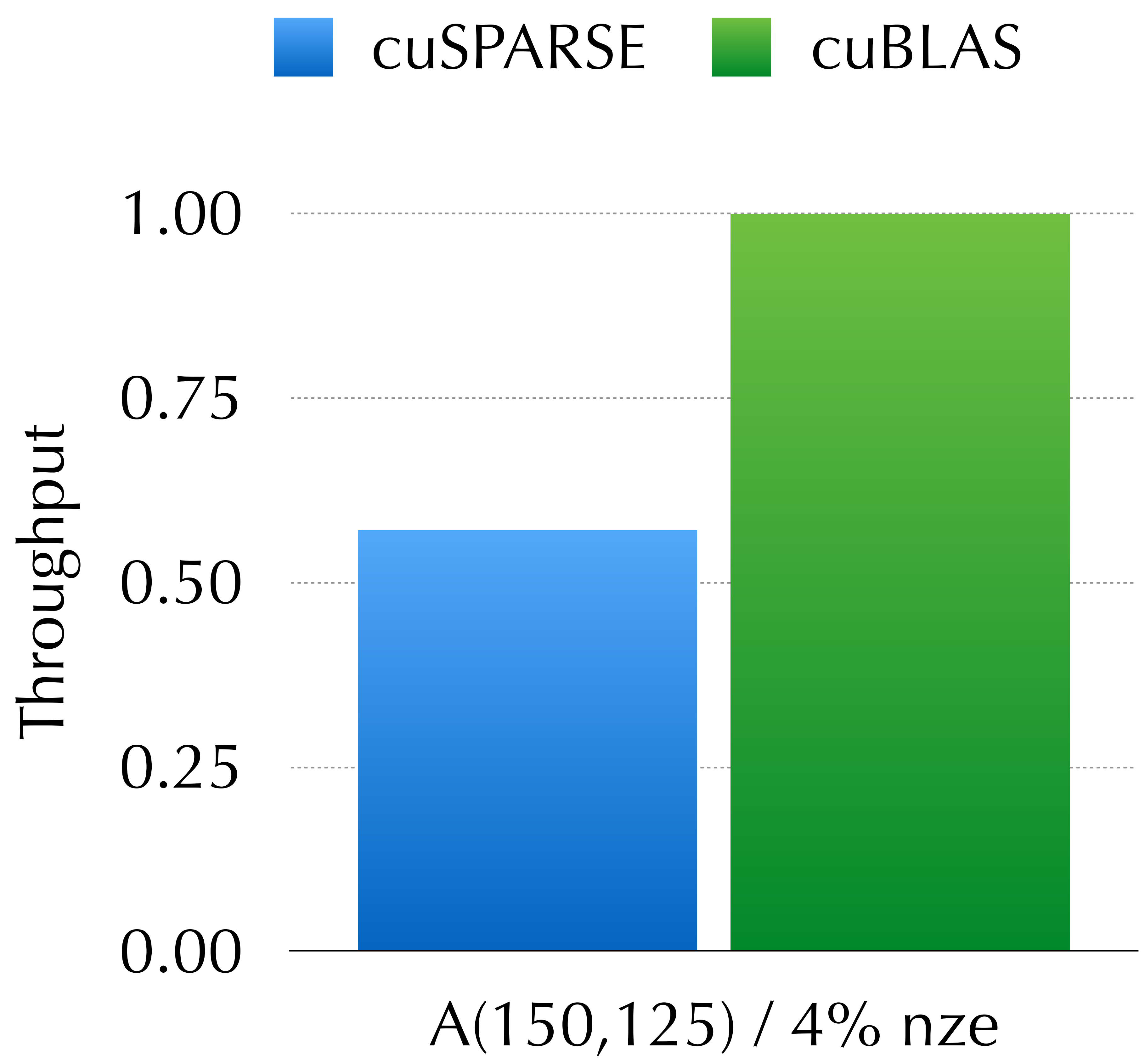
What We Know: Sparsity

- **A** can sometimes exhibit **sparsity**:

5	2		1		7	6	2	5	8		3	9			2	5	3
	1		5	7		6		1	8	4		5	3	9	2	1	8
				8	4	3	9		4	3				9		1	4
4	4	5	8	7	1	4	6	3					7	9	2	1	8
3	5	1	2		7	4	6		9	3	5		4	1	2	6	1
9		5		2	9	5	8	7	1	4				1	2	6	2
4	3	6	5			2			3			2	8	7	4	6	9
4			5	7	7		9		8		2	5	3		2	1	8
9			8	4		2	6	7	3				8	7	4	6	3
7		9		8	7	6	2		8				1	4		5	4
3	5		2				6	9	1		4	2	5	3	4	6	9
	8	9	8	8	5	2	7	4	2				9		8	1	4

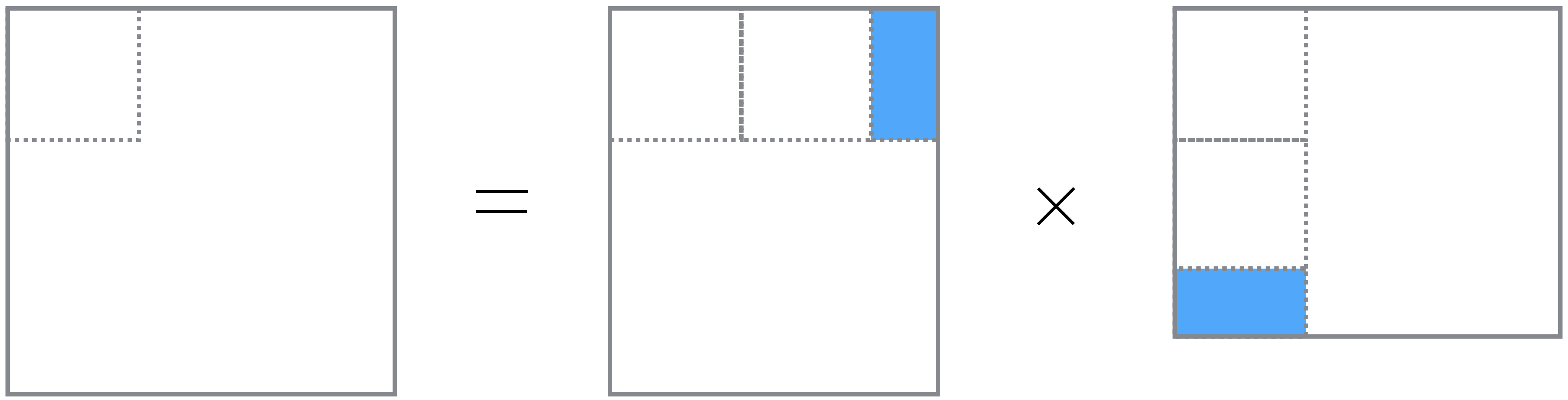
Interlude on cuSPARSE

- cuSPARSE provides **cusparseDcsrmm**.
- However, it consistently **under performs** straight cuBLAS.



Knowledge Exploitation

- Leveraging **size** we can avoid **inefficient cleanup** code.



- Leveraging **values** we can save loads from memory;
- …and exploit any **sparsity** to reduce FLOPs.

Generating Kernels

- Given an **A** generate at **runtime** a kernel for performing:

$$\mathbf{C} := \alpha\mathbf{AB} + \beta\mathbf{C}$$

- Readily accomplished using **Python** and **PyCUDA**
- We call our solution for this **GiMMiK**;
    - **G**enerator of **M**atrix **M**ultiplication **K**ernels.

GiMMiK In Action

- As an example take **A** as:

```
0.0         0.0         0.59097691
0.63448574  0.0         0.0
0.0         0.71191878  0.95941663
```

- and $\alpha = 1$ and $\beta = 0$.

GiMMiK In Action

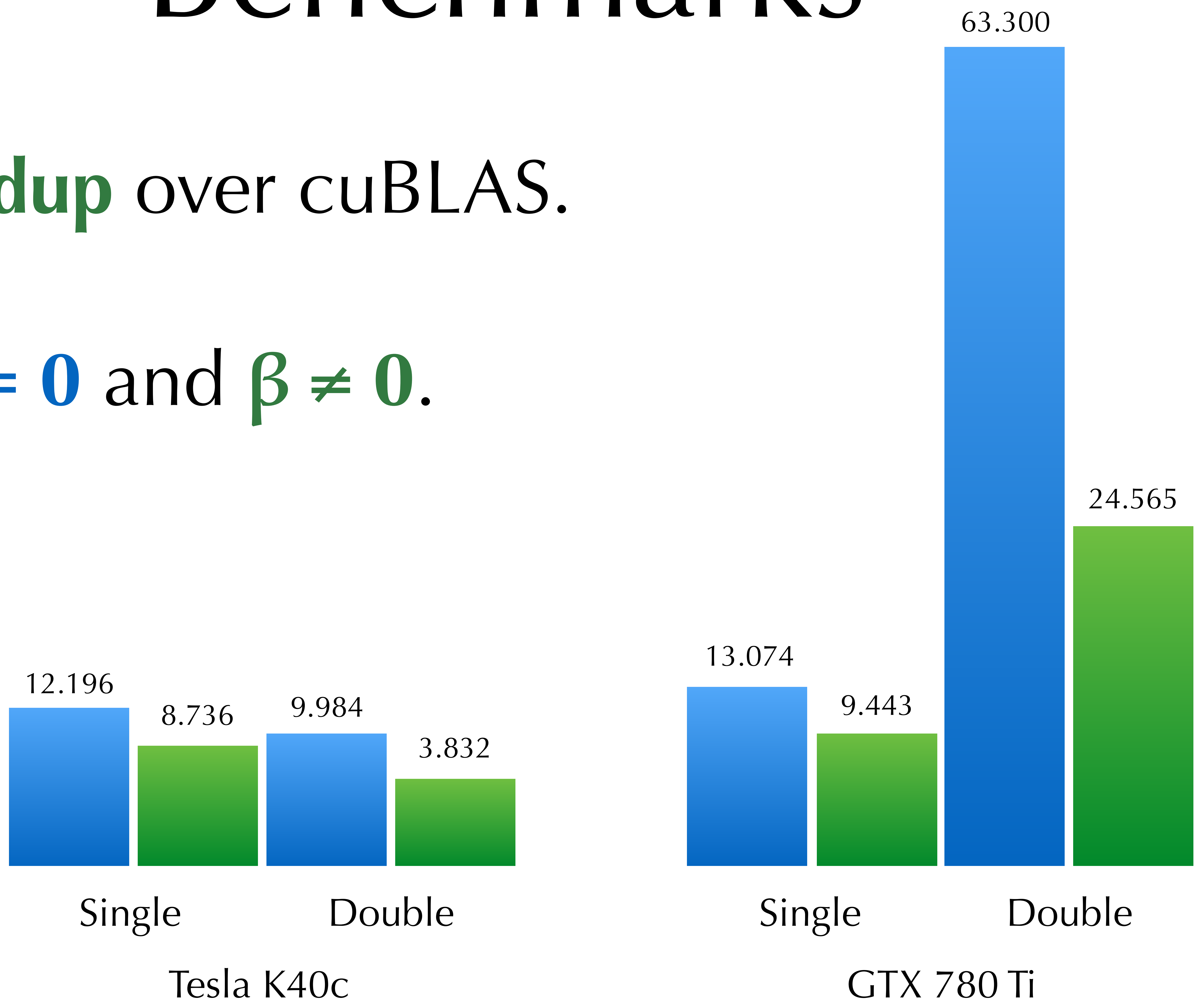
```
__global__ void
gimmik_mm(const double* __restrict__ b,
          double* __restrict__ c,
          const int width,
          const int bstride,
          const int cstride)
{
    int index = blockDim.x * blockIdx.x + threadIdx.x;
    if (index < width)
    {
        const double *b_local = b + index;
        double *c_local = c + index;

        const double subterm_0 = b_local[2 * bstride];
        const double subterm_1 = b_local[0 * bstride];
        const double subterm_2 = b_local[1 * bstride];

        c_local[0 * cstride] = 0.5909769053580467 * subterm_0;
        c_local[1 * cstride] = 0.6344857400767476 * subterm_1;
        c_local[2 * cstride] = 0.9594166286064713 * subterm_0
                             + 0.7119187815275971 * subterm_2;
    }
}
```

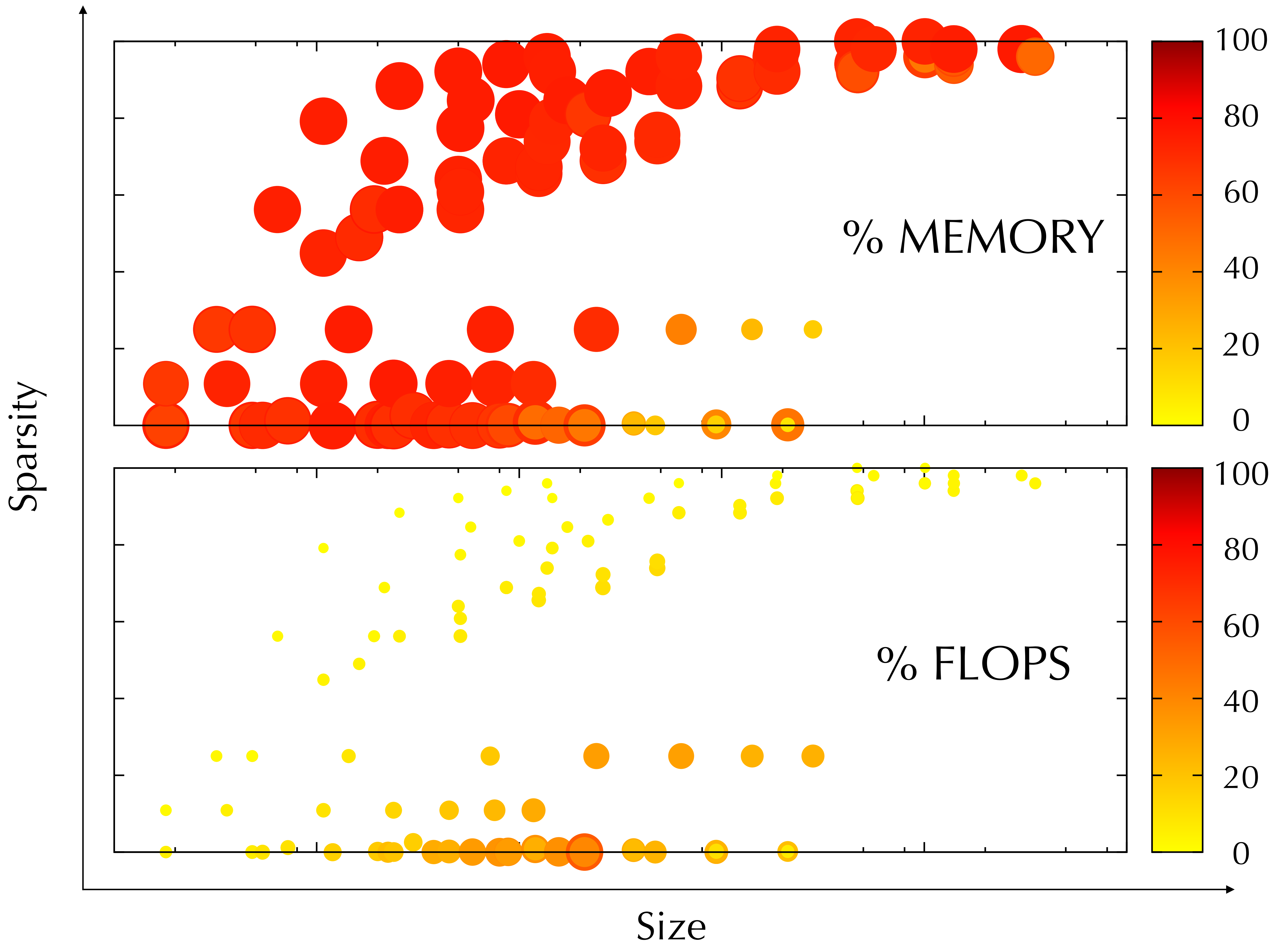
Benchmarks

- Average **speedup** over cuBLAS.
- Two cases $\boldsymbol{\beta = 0}$ and $\boldsymbol{\beta \neq 0}$.



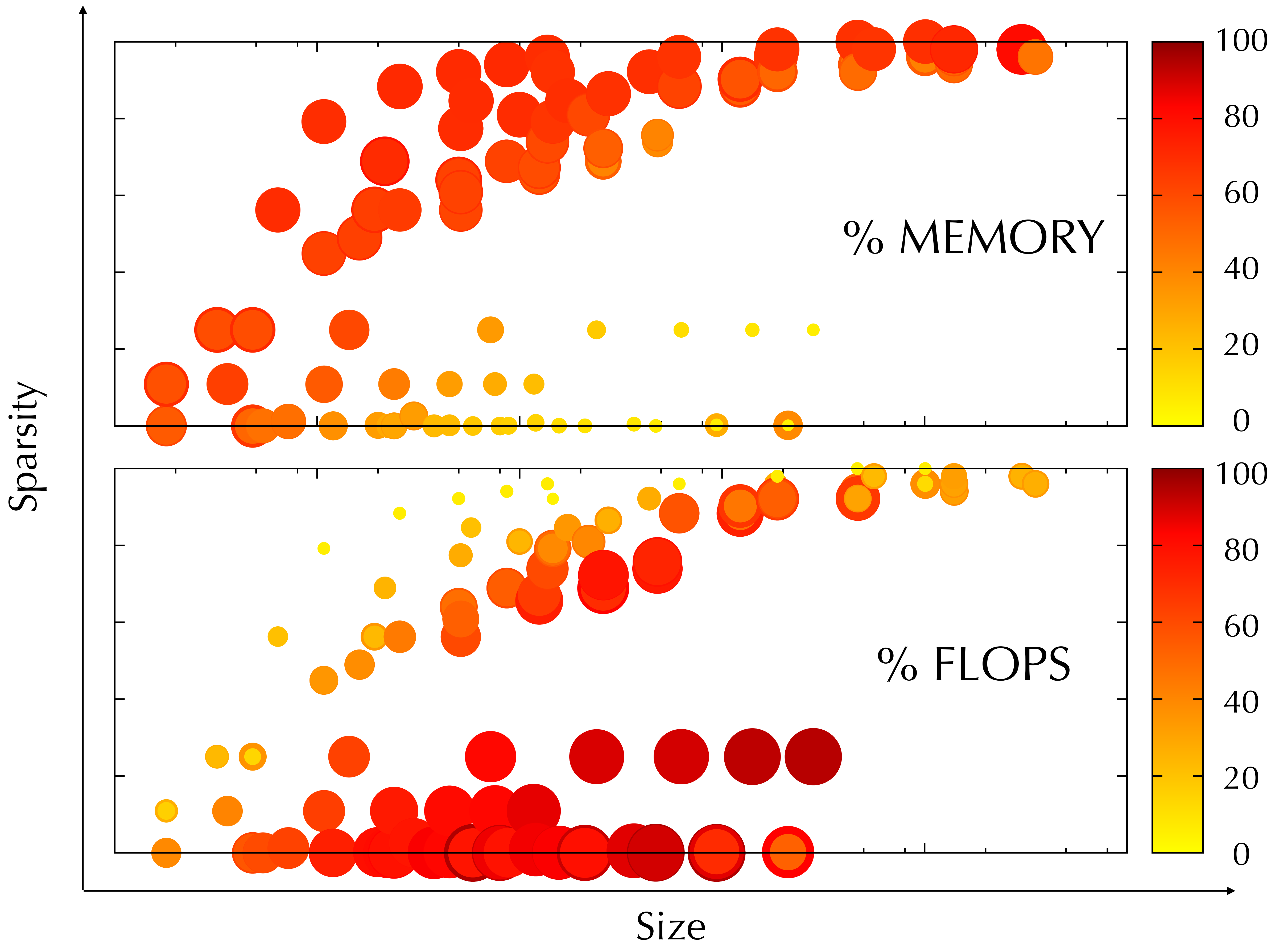
Performance Analysis: K40c

- Most sparse kernels are **bandwidth bound**.
- But **40% of peak** possible for denser cases.

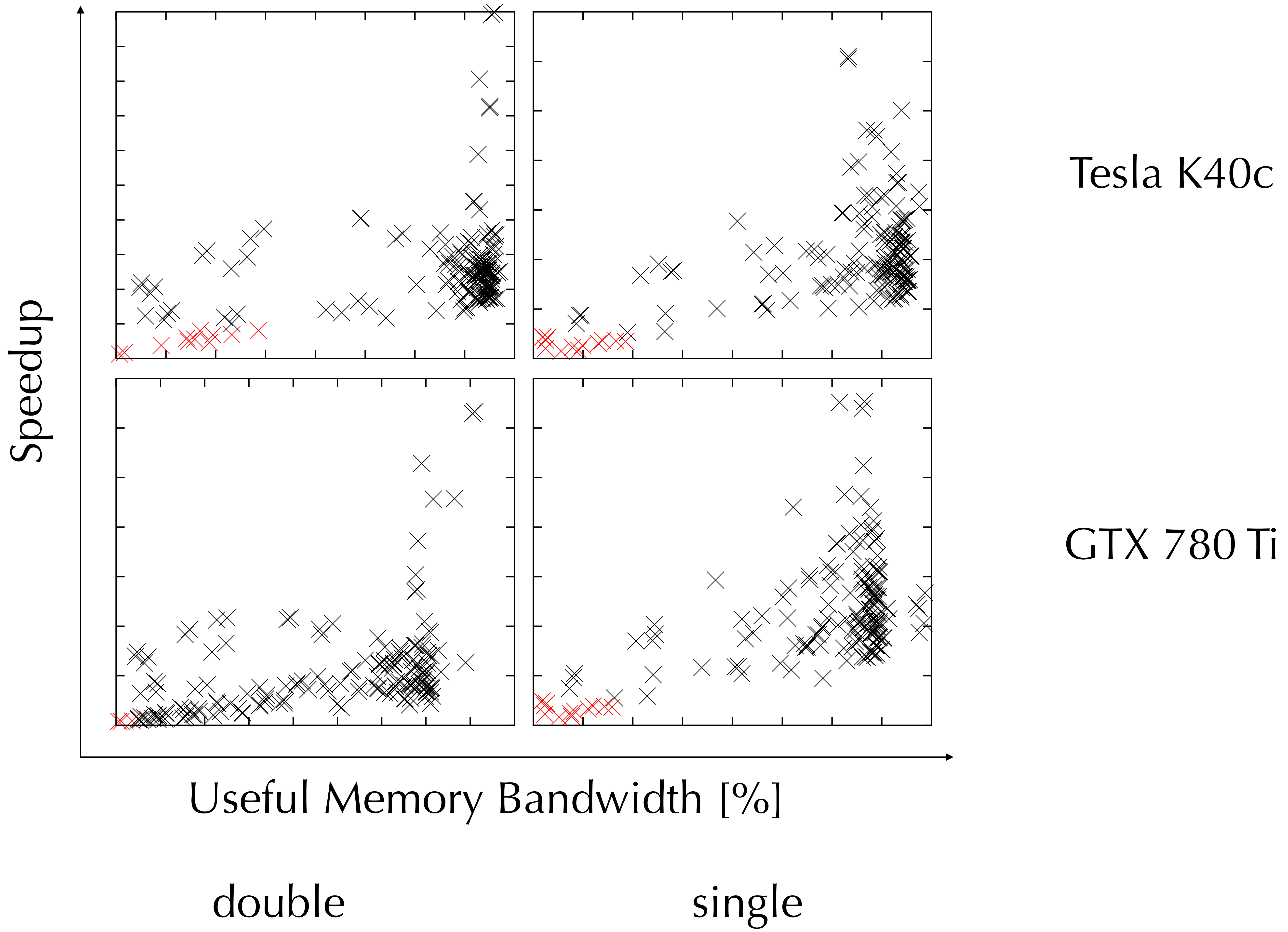


Performance Analysis: GTX 780 Ti

- Speedup for dense matrices limited by **FLOPs**.

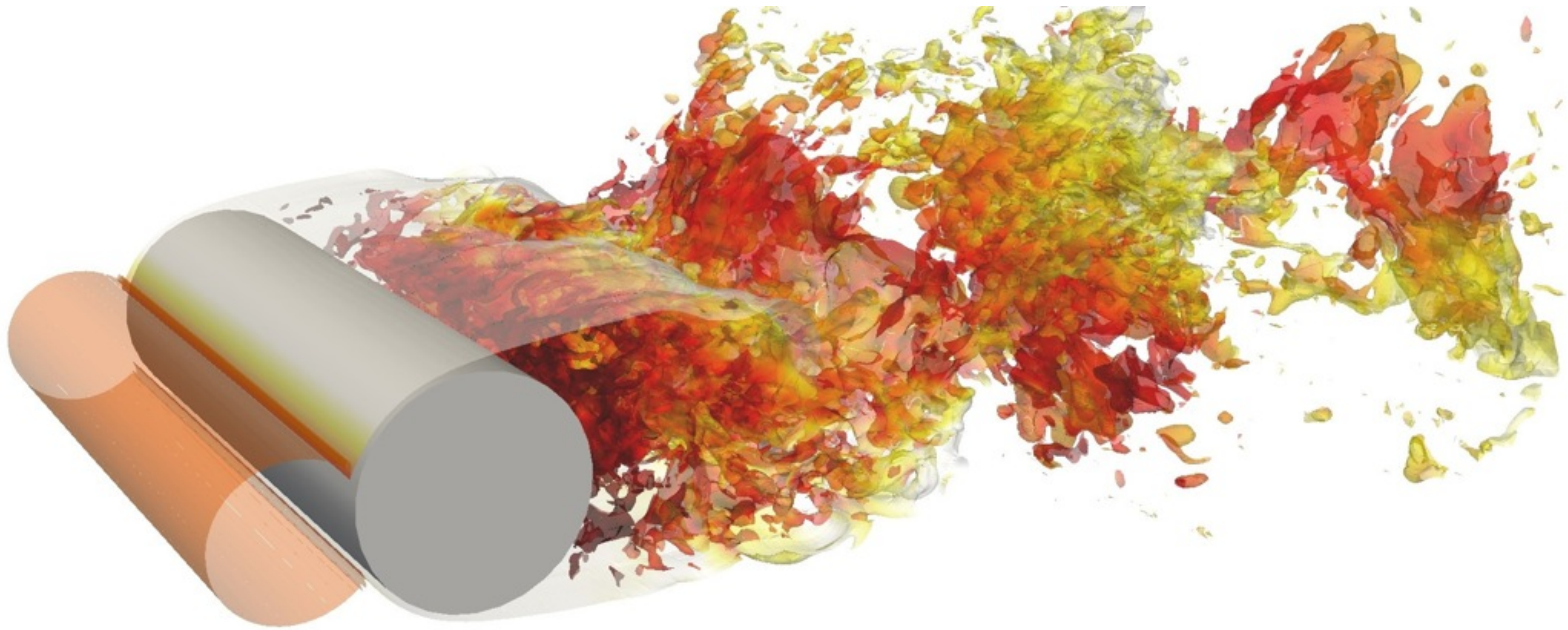


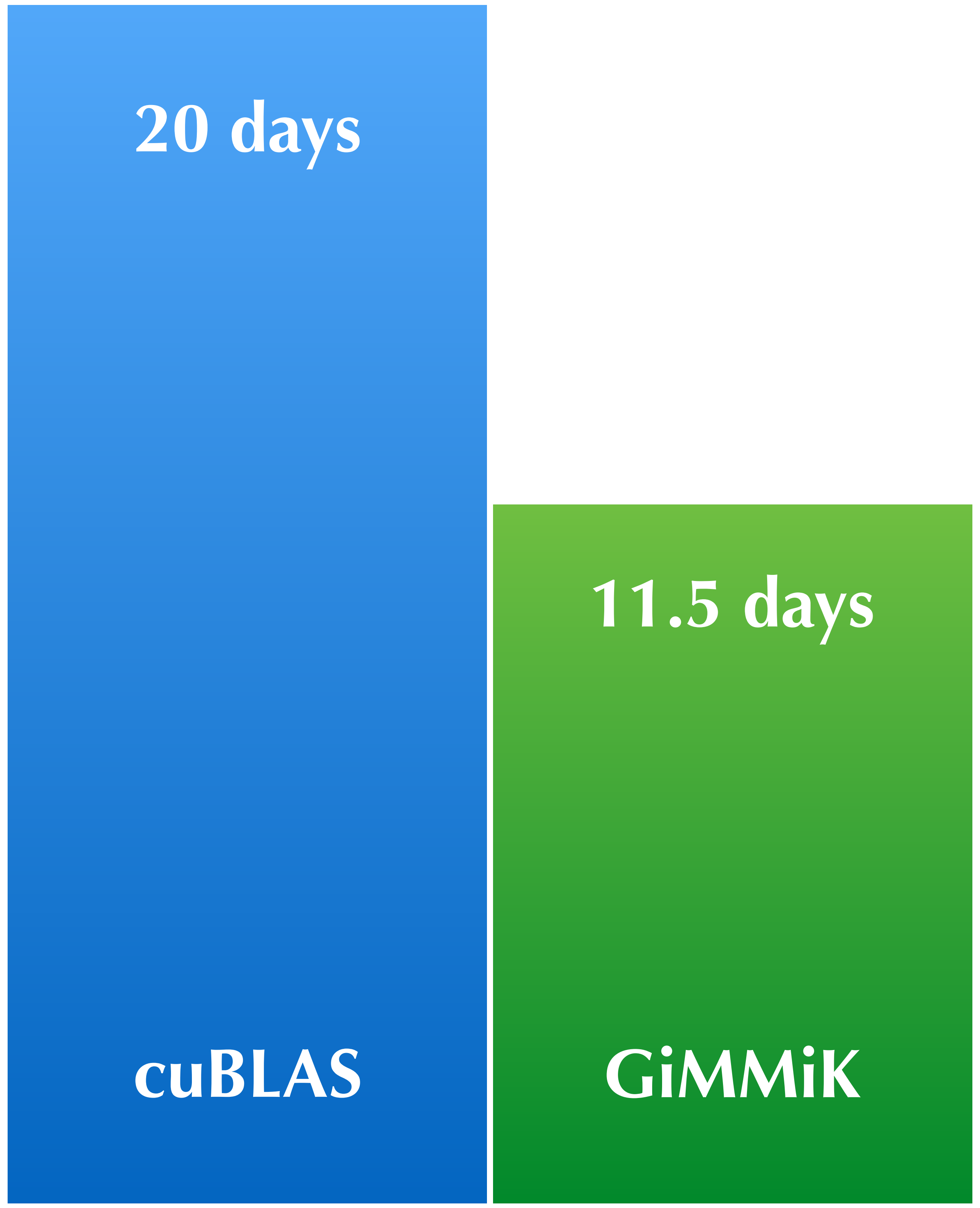
Profiling: Register Pressure



Speedup in PyFR

- Runtime for an benchmark flow problem.





Takeaway Messages

- GiMMiK can outperform cuBLAS when **A** is:
  - **small**—on account of **reduced overheads**;
  - or **relatively sparse**;
  - especially true for **fp64** on **consumer-grade hardware**.

Further Information

- Journal paper under review in Comput. Phys. Commun.

GiMMiK - Generating Bespoke Matrix Multiplication Kernels for Various Hardware Accelerators; Applications in High-Order Computational Fluid Dynamics

Bartosz D. Wozniak[a], Freddie D. Witherden[b], Peter E. Vincent[b], Paul H. J. Kelly[a]

[a] *Department of Computing, Imperial College London*
[b] *Department of Aeronautics, Imperial College London*

Abstract

Matrix multiplication is a fundamental linear algebra routine ubiquitous in all areas of science and engineering. Highly optimised BLAS libraries (cuBLAS and clBLAS on GPUs) are the most popular choices for an implementation of the General Matrix Multiply (GEMM) in software. However, performance of library GEMM is poor for small matrix sizes. In this paper we consider a *block-by-panel* type of matrix multiplication, where the block matrix is typically small (e.g. dimensions of 96×64), motivated by an application in PyFR - the most recent implementation of Flux

Summary

- You **can** beat BLAS.

- Funded and supported by

EPSRC

Engineering and Physical Sciences Research Council

NVIDIA®

- Any questions?

- E-mail: **freddie.witherden08@imperial.ac.uk**