Beating cuBLAS: Automatically Generating Bespoke Matrix Multiplication Kernels Using GiMMiK

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

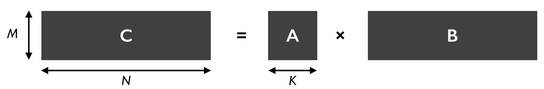
Departments of Aeronautics & Computing Imperial College London, SW7 2AZ, UK

Engineering and Physical Sciences Research Council

Introduction

PyFR [1] is a BSD licensed Python framework for solving the unsteady compressible Navier-Stokes equations on mixed unstructured grids using the high-order flux reconstruction (FR) approach. PyFR supports all of the standard finite element types in two and three dimensions and can run on a range of hardware platforms—including NVIDIA GPUs.

The majority of operations within an FR step can be abstracted as block-by-panel type matrix multiplications. In such multiplications **A** is a small block of dimension $M \times K$ and **B** is a long panel of dimension $K \times N$ with $N \gg (K, M)$ [2,3]. In FR the **A** matrix is both constant and potentially sparse.

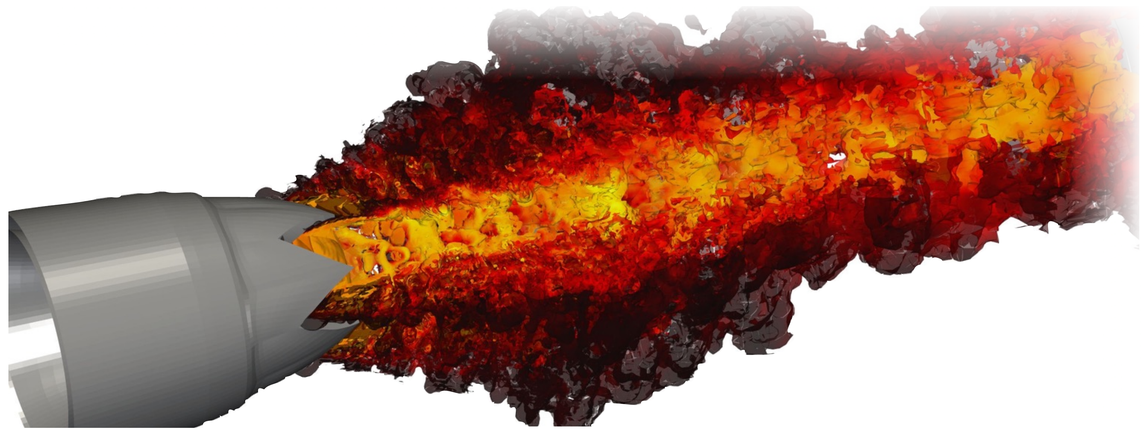


Approximately 200 distinct operator matrices occur in practice with dimensions from 3×6 to 343×1029 and sparsity factors from 0% to 99%.

The runtime of PyFR is hence dominated by calls to BLAS. Specifically, on NVIDIA GPUs, cuBLAS. Hence, in order to substantially improve the performance of PyFR it is necessary to beat cuBLAS for this particular class of matrices.

Our solution to this is called GiMMiK and employs runtime code generation to produce bespoke matrix multiplication kernels specialised to the entries of a given A matrix.

An example of a simulation performed using PyFR and GiMMiK of flow out of a serrated nozzle can be seen on the right. The simulation was performed at Reynolds number 15,000 on 256 NVIDIA K20X GPUs using 200,000 hexahedral elements with fourth order solution polynomials.



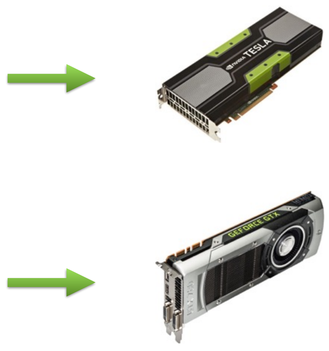
GiMMiK

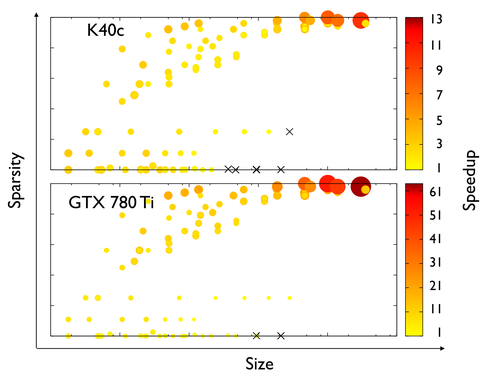
$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.5908 \\ 0.6345 & 0.0 & 0.0 \\ 0.0 & 0.7119 & 0.9594 \end{bmatrix}$$

```
__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;
    }
}
```





Speedup

Plots analysing the speedup of GiMMiK relative to cuBLAS for the operator matrices in PyFR can be seen on the left. The size of a matrix is given by MK and is shown on a log scale. Sparsity denotes the fraction of zero elements. The speedup is indicated by both the colour and size of the point. Crosses indicate a regression compared to cuBLAS.

Get GiMMiK

GiMMiK is open source and released under a three clause BSD license. Version 1.0.0 can be obtained from GitHub at:

https://github.com/vincentlab/GiMMiK

Analysis

Plots analysing the double precision performance profile of GiMMiK in terms of fraction of peak memory bandwidth and fraction of peak FLOP/s can be seen on the right. The fraction of peak obtained is indicated by both the colour and size of the point.

On the K40c the majority of matrices appear to be bandwidth bound. For the GTX 780 Ti the denser matrices can be observed to achieve near peak double precision FLOP/s. As the degree of sparsity is increased the kernels—as with the K40c—become bandwidth bound. The high fraction of peak bandwidth achieved by GiMMiK indicates that many of the kernels are close to the absolute optimal.

Acknowledgements

The authors would like to thank the Engineering and Physical Sciences Research Council for their support though grants EP/K027379/1, EP/L000407/1, EP/I00677X/1 and EP/I006613/1.

References

[1] Witherden, Freddie D., Antony M. Farrington, and Peter E. Vincent. PyFR: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. Computer Physics Communications 185.11 (2014): 3028-3040.

[2] Whaley, R. Clint, and Jack J. Dongarra. Automatically tuned linear algebra software. Proceedings of the 1998 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 1998.

[3] Goto, Kazushige, and Robert A. Geijn. Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS) 34.3 (2008): 12.