Beating cuBLAS: Automatically Generating Bespoke Matrix Multiplication Kernels Using GiMMiK

F.D. Witherden
Department of Aeronautics
Imperial College London
London, SW7 2AZ

B.D. Wozniak
Department of Computing
Imperial College London
London, SW7 2AZ

F.P. Russell
Department of Computing
Imperial College London
London, SW7 2AZ

P.E. Vincent
Department of Aeronautics
Imperial College London
London, SW7 2AZ

P.H.J. Kelly
Department of Computing
Imperial College London
London, SW7 2AZ

ABSTRACT

Matrix multiplication is a fundamental performance primitive ubiquitous in all areas of science and engineering. In this work we present GiMMiK: a generator of bespoke matrix multiplication kernels for block by panel type multiplications where the block matrix is constant. GiMMiK exploits *a priori* knowledge of this matrix to generate highly performant CUDA code for NVIDIA GPUs. The performance of GiMMiK kernels is particularly apparent when the matrix has some degree of sparsity. GiMMiK embeds matrix entries directly in the code and eliminates multiplies by zeros. Together with the ability of GiMMiK kernels to avoid poorly optimised cleanup code, GiMMiK is able to outperform cuBLAS on a variety of real-world problems. Speedups of 10 times are found on a K40c for a 294×1029 matrix with 99% sparsity. It is open source and released under a three clause BSD license.

1. INTRODUCTION

Matrix multiplications of the form

$$\mathbf{C} = \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C}, \tag{1}$$

lie at the heart of a variety of popular numerical methods. Over the past thirty years significant effort has been expended in developing optimised matrix multiplication kernels [1, 3]. Heretofore implementors have generally focused on the cases where (i) $\mathbf{A}$ and $\mathbf{B}$ are large, dense, and approximately square (GEMM), (ii) $\mathbf{A}$ is large, constant, and extremely sparse, and $\mathbf{B}$ is large and dense (SpMM); with there being comparatively less work on the case of $\mathbf{A}$ being moderately sized, constant, and potentially sparse. Existing libraries often fall short here either due to suboptimal tiling strategies for small matrices or due to the high overheads associated with generic representations such as CSR and ELLPACK.

Our focus in this work is on automatically generating bespoke block by panel type matrix multiplication kernels in the case of a constant, and potentially sparse, $\mathbf{A}$ matrix.

2. MOTIVATION

PyFR [4] is a BSD licensed Python framework for solving the compressible Navier-Stokes equations on mixed unstructured grids using the high-order flux reconstruction (FR) approach. PyFR has been designed from the ground up to target NVIDIA GPUs. Within an FR time-step the majority of operations involve manipulating polynomials inside of an element. It is possible to cast these manipulations as block by panel type matrix multiplications where $\mathbf{A}$ takes the form of a *constant operator matrix*. The size of these operator matrices depends on the operation, the type of the element, and the polynomial order. In some instances, such as for tensor product, elements $\mathbf{A}$ can also be sparse. Approximately 200 distinct operator matrices appear in practice with dimensions from 3×6 to 343×1029 with sparsity factors from 0% in the completely dense case to 99.2% in the sparsest case.

On NVIDIA GPUs the reference BLAS-like library for performing dense matrix multiplications is cuBLAS [2]. Developed by NVIDIA and part of the CUDA runtime cuBLAS is highly optimised. Given that the majority of operations within an FR time step are matrix multiplications then runtime of PyFR is hence dominated by calls to cuBLAS. Despite the presence of sparsity in $\mathbf{A}$ it has been found that in the majority of cases that straight cuBLAS outperforms cuSPARSE. It has therefore been difficult to reliably exploit this sparsity without writing the specific matrix multiplication kernels by hand.

3. GIMMIK

In order to improve the performance of PyFR it is necessary to beat cuBLAS. In this work, we present GiMMiK, a Python library for automatically generating bespoke matrix multiplication kernels for NVIDIA GPUs in the case where $\mathbf{A}$ is known *a priori*. This approach makes it possible to both embed the coefficients of $\mathbf{A}$ into the kernel itself while also

```
__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;

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

        c_local[0*cstride] = 0.5908*subterm_0;
        c_local[1*cstride] = 0.6345*subterm_1;
        c_local[2*cstride] = 0.9594*subterm_0
                           + 0.7119*subterm_2;
    }
}
```

Figure 1: GiMMiK generated CUDA kernel for performing $\mathbf{C} = \mathbf{AB}$ where $\mathbf{A}$ is given by Eq. (2).

eliminating unnecessary multiplies by zeros. In GiMMiK each CUDA thread performs a complete matrix-vector product. For small dense $\mathbf{A}$ matrices with irregular shapes this can result in significantly less overhead compared to cuBLAS. Further as GiMMiK is a Python library it is possible for the kernel generation and compilation to occur at runtime.

GiMMiK is open source and released under a three clause BSD license. It is available from [5].

4. EXAMPLE

Given the constant operator matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0.5908 \\ 0.6345 & 0 & 0 \\ 0 & 0.7119 & 0.9594 \end{bmatrix}, \tag{2}$$

the GiMMiK generated CUDA kernel for performing $\mathbf{C} = \mathbf{AB}$ is shown in Figure 1 where we note that GiMMiK both embeds the elements of the $\mathbf{A}$ matrix into the kernel and eliminates any multiplies by zeros..

5. PERFORMANCE

The speedup of GiMMiK compared with cuBLAS can be seen in Figure 2. On both high-end and consumer-grade hardware GiMMiK can be seen to consistently yield a speedup when compared against cuBLAS. Speed improvements of up to 10 times are reported on a K40c for a 294×1029 matrix with 99% sparsity. Further, improvements of up to 63 times are observed on the consumer grade GTX 780 TI card on account of the restricted double precision capabilities. Only for the largest dense matrices are regressions observed. This is attributed to excessive register spillage caused by the large number of constants embedded into the kernel.

6. CONCLUSIONS AND FUTURE WORK

In this work, we have shown how it is possible to outperform cuBLAS for a variety of dense and sparse block by panel type multiplications. This is accomplished using GiMMiK, a tool for automatically generating bespoke CUDA matrix multiplication kernels.

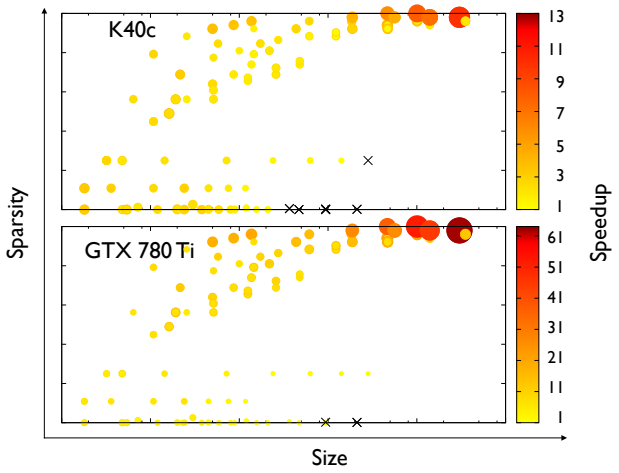

Figure 2: Speedup for GiMMiK over cuBLAS for the operator matrices associated with PyFR. The total number of elements in each $\mathbf{A}$ matrix is shown along the x-axis and plotted on a log scale. The fraction of these elements which are zero is shown on the y-axis. Speedup is indicated by the colour and size of the point with crosses indicating a regression compared with cuBLAS.

In the future, we plan on extending GiMMiK so that it can also target regular CPUs via the generation of OpenMP annotated C source code. Register tiling will also be investigated for some of the larger matrices as a means to reduce register spillage and hence improve memory bandwidth utilisation.

7. ACKNOWLEDGMENTS

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.

8. REFERENCES

[1] K. Goto and R. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3):12:1–12:25, May 2008.

[2] NVIDIA. *CUBLAS Library User Guide*. NVIDIA, v5.0 edition, 2012.

[3] R. C. Whaley and J. J. Dongarra. Automatically Tuned Linear Algebra Software. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, pages 1–27, Washington, DC, USA, 1998. IEEE Computer Society.

[4] F. D. Witherden, A. M. Farrington, and P. 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):3028–3040, 2014.

[5] B. D. Wozniak, F. D. Witherden, F. P. Russell, P. E. Vincent, and P. H. J. Kelly. GiMMiK. https://github.com/vincentlab/GiMMiK.