GEMM-like Tensor-Tensor Contraction

Paul Springer
AICES, RWTH Aachen University
Aachen, Germany
springer@aices.rwth-aachen.de

Paolo Bientinesi
AICES, RWTH Aachen University
Aachen, Germany
pauldj@aices.rwth-aachen.de

1 INTRODUCTION

Tensor contractions (TCs)—a generalization of matrix-matrix multiplications (GEMM) to multiple dimensions—arise in a wide range of scientific computations such as machine learning [1], spectral element methods [10], quantum chemistry calculations [2], multidimensional Fourier transforms [4] and climate simulations [3]. Despite the close connection between TCs and matrix-matrix products, the performance of the former is in general vastly inferior to that of an optimized GEMM. To close such a gap, we propose a novel approach: **GE**MM-like **T**ensor-**T**ensor (GETT) contraction [8].

GETT captures the essential features of a high-performance GEMM. In line with a highly optimized GEMM, and in contrast to previous approaches to TCs, our high-performance implementation of the GETT approach is fully vectorized, exploits the CPU's cache hierarchy, avoids explicit preprocessing steps, and operates on arbitrary dimensional subtensors while preserving stride-one memory accesses. Another key design principle is the fact that GETT utilizes tensor transpositions [9] to pack subtensors into the caches, yielding highly efficient packing routines.

A central challenge to high-performance tensor contractions—in contrast to matrix-matrix multiplications—is due to the increased dimensionality of tensors, often resulting in suboptimal memory accesses; as a consequence, the performance of many tensor contractions can be limited by the system's memory-bandwidth (i.e., bandwidth-bound) as opposed to its floating-point units (i.e., compute-bound). Hence, developing a systematic way to exploit the system's caches is critical to increase the performance of TCs and push them from the bandwidth-bound regime to the compute-bound regime.

Previous approaches to TCs, such as **L**oop-**o**ver-**G**EMM (LoG) or **T**ranspose-**T**ranspose-**G**EMM-**T**ranspose (TTGT) rely on highly optimized matrix-matrix multiplications to attain high performance. However, both approaches suffer from inherent disadvantages: Foremost, both LoG as well as TTGT exhibit suboptimal I/O cost, making them especially less effective in the bandwidth-bound regime. While TTGT requires additional auxiliary memory, increasing the memory footprint of the tensor contraction by up to 2×, LoG is not applicable to all tensor contractions without relying on a strided (suboptimal) GEMM implementation.

Simultaneously to our research, Matthews [6] recently introduced another "native" tensor contraction algorithm (TBLIS) that also adopts a GEMM-like design, attaining the same I/O cost as an equivalent-sized matrix-matrix product. TBLIS—in contrast to GETT—does not offer multi-dimensional packing routines, which can result in strided memory accesses. However, TBLIS is already available as a C++ library that avoids the code-generation process currently required by GETT.

Super Computing '17, November 2017, Denver, CO, USA
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 GEMM-LIKE TENSOR CONTRACTION

Let $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ be tensors (i.e., multi-dimensional arrays), a tensor contraction is:

$$\mathcal{C}_{I_\mathcal{C}} \leftarrow \alpha \times \mathcal{A}_{I_\mathcal{A}} \times \mathcal{B}_{I_\mathcal{B}} + \beta \times \mathcal{C}_{I_\mathcal{C}}, \qquad (1)$$

where $I_\mathcal{A}$, $I_\mathcal{B}$, and $I_\mathcal{C}$ denote symbolic index sets. We further define three important index sets: $I_m := I_\mathcal{A} \cap I_\mathcal{C}$, $I_n := I_\mathcal{B} \cap I_\mathcal{C}$, and $I_k := I_\mathcal{A} \cap I_\mathcal{B}$, respectively denoting the *free indices* of $\mathcal{A}$, the *free indices* $\mathcal{B}$, and the contracted indices. Moreover, we adopt the *Einstein notation*, indicating that all indices of I_k are implicitly summed (i.e., contracted).

Adhering to this definition, we can reformulate Equation (1) such that its similarity to a matrix-matrix multiplication becomes apparent:

$$\mathcal{C}_{\Pi^\mathcal{C}(I_m \cup I_n)} \leftarrow \alpha \times \mathcal{A}_{\Pi^\mathcal{A}(I_m \cup I_k)} \times \mathcal{B}_{\Pi^\mathcal{B}(I_n \cup I_k)} + \beta \times \mathcal{C}_{\Pi^\mathcal{C}(I_m \cup I_n)}, \qquad (2)$$

where $\Pi^\mathcal{A}, \Pi^\mathcal{B}$, and $\Pi^\mathcal{C}$ denote arbitrary permutations of the indices respectively belonging to $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$. The only difference between tensor contractions and matrix-matrix multiplication is that the former allows $|I_m|, |I_n|, |I_k| \geq 1$, while the latter requires that $|I_m| = |I_n| = |I_k| = 1$.

```
// N-Loop
for n = 1 : n_C : S_In                                          ①
   // K-Loop (contracted)
   for k = 1 : k_C : S_Ik                                       ②
      B^ = identify_subtensor(B, n, k)
      // pack B^ into B~
      B~ = packB(B^)                                            ③
      // M-Loop
      for m = 1 : m_C : S_Im                                    ④
         A^ = identify_subtensor(A, m, k)
         // pack A^ into A~
         A~ = packA(A^)                                         ⑤
         C^ = identify_subtensor(C, m, n)
         // matrix-matrix product: C^ ← αA~ × B~ + βC^
         macroKernel(A~, B~, C^, α, β)                          ⑥
```

Listing 1: GETT design.

Blocking & Packing. Listing 1 outlines the GEMM-like design of GETT, which is inspired by BLIS [11]. GETT begins to block along the indices of I_n (①) and I_k (②), at this stage the size of the subtensor $\widehat{\mathcal{B}}$ of $\mathcal{B}$ is constrained and can be packed into a local buffer $\widetilde{\mathcal{B}}$ via tensor transpositions (③). The next steps involve blocking along the indices of I_m (④) and packing the subtensor $\widehat{\mathcal{A}}$ of $\mathcal{A}$ into a local buffer $\widetilde{\mathcal{A}}$. At this point the data is suitable prepared for the *macro-kernel*—a specialized form of an in-cache matrix-matrix product (⑥). This design resembles that of a high-performance GEMM while the additional complexity due to the higher dimensionality of the tensors has been moved into the packing routines.

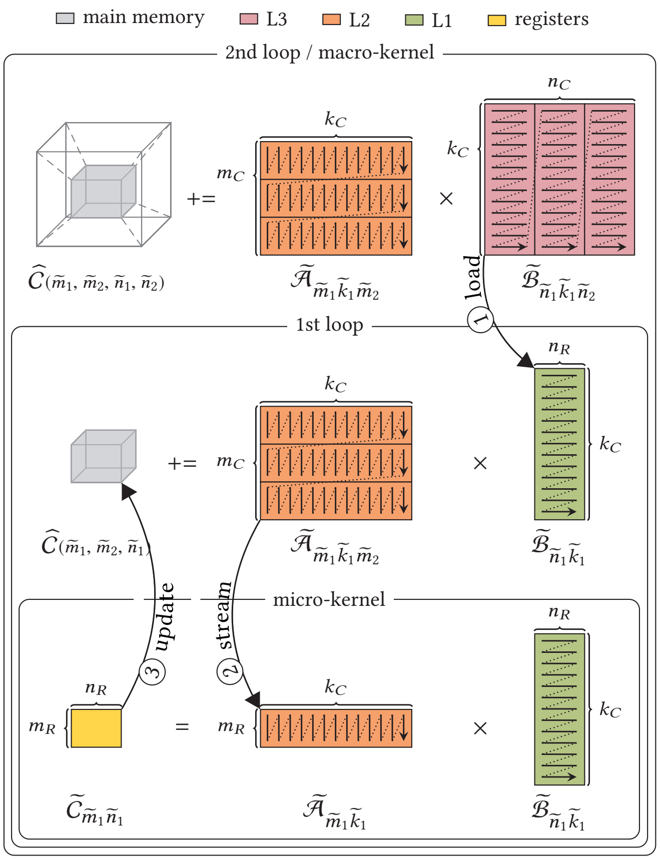

Figure 1: Macro-kernel & micro-kernel.

Macro-Kernel. The sizes of the packed subtensors are chosen such that $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$ respectively reside in the L2- and L3-caches during the invocation to the macro-kernel (see Figure 1); the macro-kernel exposes two additional loops around a *micro-kernel* that computes a fat outer-product of two appropriate slivers from $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$ that fit into the L1-cache; the sizes of m_R and n_R are chosen such that the corresponding tile of $\widetilde{C}$ is kept in registers.

Parallelism. GETT exposes a total of five loops around the micro-kernel that can be parallelized in a nested fashion [7]; a performance heuristic is used to select an appropriate parallelization strategy that maximizes load-balancing while staying within the cache limitations. Special care has to be taking when the loop associated to I_k (i.e., 4th loop around the micro-kernel) ought to be parallelized due to the simultaneous updates to the same portion of C. Thus, the 4th loop is only parallelized if the available parallelism along the I_m and I_n dimensions is too few (i.e., C is very small); in that case GETT manages multiple copies of C and reduces these copies after the macro-kernel has completed.

Performance. Figure 2 compares the performance of GETT to that of TBLIS [6] and Eigen [5] across a benchmark of 1013 random tensor contractions[1]: In a multi-threaded setting, GETT respectively attains an average speedup of 1.6× and 16.2× over these

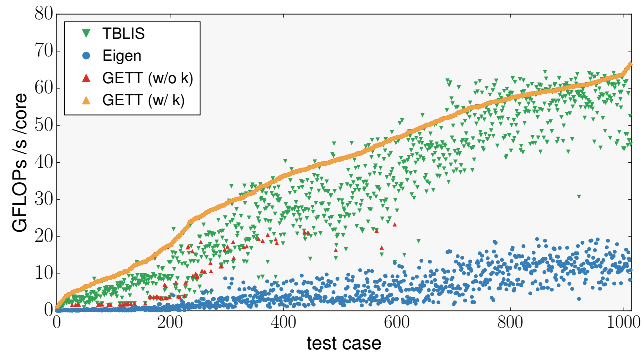

Figure 2: Performance for TCs with varying arithmetic intensity (sorted w.r.t. GETT). Host: 2× Intel *Xeon E5-2680 v3* using 24 threads.

frameworks. These speedups highlight GETT's excellent performance. Moreover, comparing the orange triangles (with parallel-k) with the red triangles (without parallel-k) indicates that some tensor contraction benefit from GETT's ability to parallelize the loop associated to I_k.

3 CONCLUSION

We presented GETT, a novel GEMM-like approach for tensor contractions. GETT—similarly to a high-performance GEMM—is fully vectorized, blocks for all levels of the cache hierarchy as well as for registers, and parallelizes all five loops around its micro-kernel individually to yield good load-balancing. Moreover, it utilizes highly optimized tensor transpositions for its packing routines to attain high bandwidth. These desirable properties reflect positively on GETT's performance.

REFERENCES

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. (2015). https://www.tensorflow.org
[2] Rodney J. Bartlett and Monika Musiał. 2007. Coupled-cluster theory in quantum chemistry. *Reviews in Modern Physics* 79, 1 (2007), 291–352.
[3] John Drake, Ian Foster, John Michalakes, Brian Toonen, and Patrick Worley. 1995. Design and performance of a scalable parallel community climate model. *Parallel Comput.* 21, 10 (1995), 1571–1591.
[4] Matteo Frigo and Steven G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (Feb 2005), 216–231.
[5] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3.3.4. http://eigen.tuxfamily.org. (2010).
[6] Devin A. Matthews. 2016. High-Performance Tensor Contraction without BLAS. *CoRR* abs/1607.00291 (2016). http://arxiv.org/abs/1607.00291
[7] Tyler M. Smith, Robert A. van de Geijn, Mikhail Smelyanskiy, Jeff R. Hammond, and Field G. Van Zee. 2014. Anatomy of High-Performance Many-Threaded Matrix Multiplication. In *28th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2014)*.
[8] Paul Springer and Paolo Bientinesi. 2016. *Design of a High-Performance GEMM-like Tensor-Tensor Multiplication.* Technical Report. arXiv:1607.00145 Under review for ACM Transactions on Mathematical Software.
[9] Paul Springer, Tong Su, and Paolo Bientinesi. 2017. HPTT: A High-Performance Tensor Transposition C++ Library. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY 2017)*. ACM, 56–62.
[10] Henry M Tufo and Paul F Fischer. 1999. Terascale spectral element algorithms and implementations. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*. ACM, 68.
[11] Field G. Van Zee and Robert A. van de Geijn. 2015. BLIS: A Framework for Rapidly Instantiating BLAS Functionality. *ACM Trans. Math. Software* (2015).

[1] The full set of TCs can be found at hpac.rwth-aachen.de/springer/randomTCs.dat.