Strassen's Algorithm for Tensor Contraction

Jianyu Huang
jianyu@cs.utexas.edu
Department of Computer Science
Institute for Computational Engineering and Sciences
The University of Texas at Austin

Devin A. Matthews
dmatthews@utexas.edu
Institute for Computational Engineering and Sciences
The University of Texas at Austin

Robert A. van de Geijn
rvdg@cs.utexas.edu
Department of Computer Science
Institute for Computational Engineering and Sciences
The University of Texas at Austin

1 INTRODUCTION

Standing on the shoulders of giants. This work builds upon a number of recent developments: The GotoBLAS algorithm for matrix multiplication (GEMM) [5] that underlies the currently fastest implementations of GEMM for CPUs; The refactoring of the GotoBLAS algorithm as part of the BLAS-like Library Instantiation Software (BLIS) [15, 16], which exposes primitives for implementing BLAS-like operations; The systematic parallelization of the loops that BLIS exposes so that high-performance can be flexibly attained on multicore and many-core architectures [12]; The casting of tensor contraction (TC) in terms of the BLIS primitives [10, 13] without requiring the transposition (permutation) used by traditional implementations; The practical high-performance implementation of the classical Strassen's algorithm (Strassen) [8] in terms of variants of the BLIS primitives; and the extension of this implementation [7] to a family of Strassen-like Fast Matrix Multiplication (FMM) algorithms [1]. Together, these results facilitate what we believe to be the first extension of Strassen's algorithm to TC.

Contributions. This work presents the first efficient implementation of Strassen's algorithm for tensor contraction, a significant problem with numerous applications.

- It describes how to extend Strassen's algorithm to TC without the explicit transposition of data that inherently incurs significant memory movement and workspace overhead.
- It provides a performance model for the cost of the resulting family of algorithms.
- It details the practical implementation of these algorithms, including how to exploit variants of the primitives that underlie BLIS and a data layout to memory for the tensors.
- It demonstrates practical speedup on modern single core and multicore CPUs.
- It illustrates how the local Strassen's TC algorithm improves performance of a simple distributed memory tensor contraction.

Together, these results unlock a new frontier for the research and application of Strassen's algorithm.

Related work. To the best of our knowledge, this work represents the first implementation of Strassen's algorithm for tensor contraction. In the context of Strassen for matrices, there have been a variety of practical implementations [1, 2, 4], including the closely related implementation of Strassen using the BLIS framework [8] which we extend.

For tensor contraction, recent work on high-performance tensor contraction [10, 13] serves as the motivation and basis for our present work, while other research has focused on algorithms using tensor slicing [3, 11] or on improving the efficiency of the so-called ttdt algorithm for tensor contraction [6, 9, 14], where input tensors $\mathcal{A}$ and $\mathcal{B}$ are Transposed (permuted) and then used in a standard gemm algorithm, with the output then being Transposed and accumulated onto the tensor $\mathcal{C}$. ttdt could be used to construct a Strassen algorithm for TC by transposing subtensors into submatrices and vice versa and using a matrix implementation of Strassen instead of gemm. However, we will show that this algorithm is essentially the same as our **Naive Strassen** algorithm, which is often less efficient than the other algorithms that we have implemented.

The gett algorithm [13] is a high-performance tensor contraction implementation similar in many ways to the BLIS-based implementation by Matthews [10]. As in our current implementation, formation of linear combinations of input subtensors of $\mathcal{A}$ and $\mathcal{B}$ and output to multiple subtensors of $\mathcal{C}$ could be fused with the internal tensor transposition and micro-kernel steps of gett. However, the implementation would be restricted to regular subtensors rather than more general submatrices, which could have possible negative performance implications (e.g. false sharing).

2 SUMMARY AND CONCLUSIONS

As exemplified by Figure 1, performance benefits are demonstrated with a performance model as well as in practice, achieving up to 1.3× speedup (**AB Strassen** for large problem sizes). We presented what we believe to be the first work to demonstrate how to leverage Strassen's algorithm for tensor contraction, and showed practical performance speedup on single core, multicore, and distributed memory implementations. Using a block scatter matrix layout enables us to partition the matrix view of the tensor, instead of the tensor itself, with automatic (implicit) tensor-to-matrix transformation, and the flexibility to facilitate Strassen's 2D matrix partition to multi-dimensional tensor spaces. Fusing the matrix summation that must be performed for Strassen and the transposition that must be conducted for tensor

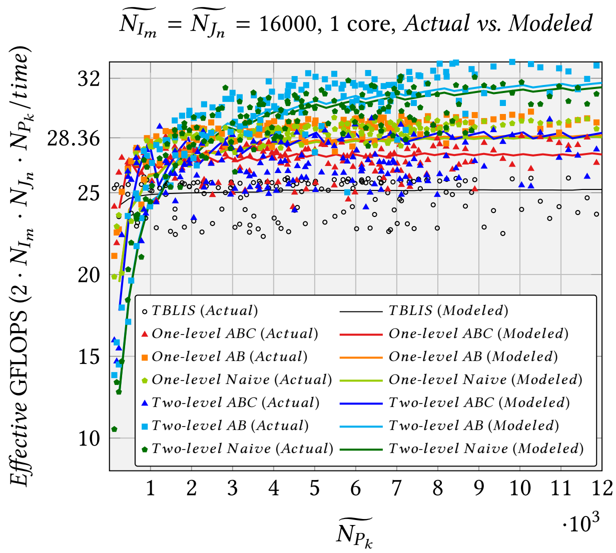

Figure 1: *Actual and modeled performance of various implementations on single core:* $N_{I_m} \approx N_{J_n} \approx 16000$, N_{P_k} *varies.*

contraction with the packing and micro-kernel operations inside high-performance implementation of GEMM avoids extra workspace requirements, and reduces the cost of additional memory movement. We provided a performance model which can accurately predict the speedup of the resulting family of algorithms for different tensor shapes, sizes, and permutations. We evaluated our families of implementations for various tensor sizes and shapes on synthetic and real-world datasets, both observing significant speedups comparing to the baseline (TBLIS) and naive implementations (**Naive Strassen**), particularly for smaller problem sizes ($N_{I_m}, N_{J_n}, N_{P_k} \approx 2k_C, 4k_C$), and irregular shape ($N_{P_k}$ is much smaller comparing to N_{I_m}, N_{J_n}) with **ABC Strassen**. Together, this work demonstrates Strassen's algorithm can be applied for tensor contraction with practical performance benefit.

ACKNOWLEDGMENTS

This work was sponsored in part by the National Science Foundation under grant number ACI-1550493, CCF-1714091, by Intel Corporation through an Intel Parallel Computing Center grant, and by a gift from Qualcomm. Access to the Maverick supercomputers administered by TACC is gratefully acknowledged. Devin Matthews is an Arnold O. Beckman Postdoctoral Fellow. We thank Martin Schatz for his help with distributed memory implementations, and the rest of the SHPC team (`http://shpc.ices.utexas.edu`) for their supports.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Austin R. Benson and Grey Ballard. 2015. A Framework for Practical Parallel Fast Matrix Multiplication. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2015)*. ACM, 42–53.

[2] Paolo D'Alberto, Marco Bodrato, and Alexandru Nicolau. 2011. Exploiting Parallelism in Matrix-computation Kernels for Symmetric Multiprocessor Systems: Matrix-multiplication and Matrix-addition Algorithm Optimizations by Software Pipelining and Threads Allocation. *ACM Trans. Math. Softw.* 38, 1, Article 2 (December 2011), 30 pages.

[3] E. Di Napoli, D. Fabregat-Traver, G. Quintana-Orti, and P. Bientinesi. 2014. Towards an efficient use of the BLAS library for multilinear tensor contractions. *Appl. Math. Comput.* 235 (2014), 454–468.

[4] C.C. Douglas, M. Heroux, G. Slishman, and R.M. Smith. 1994. GEMMW - A Portable Level 3 BLAS Winograd Variant of Strassen's Matrix-Matrix Multiplication Algorithm. *J. Computational Physics* (1994), 1–10.

[5] Kazushige Goto and Robert A. van de Geijn. 2008. Anatomy of a High-Performance Matrix Multiplication. *ACM Trans. Math. Soft.* 34, 3 (May 2008), 12.

[6] A. Hartono, Q. Lu, T. Henretty, S. Krishnamoorthy, H. Zhang, G. Baumgartner, D. E. Bernholdt, M. Nooijen, R. Pitzer, J. Ramanujam, and P. Sadayappan. 2009. Performance optimization of tensor contraction expressions for many-body methods in quantum chemistry. *J. Phys. Chem. A* 113, 45 (2009), 12715–12723.

[7] Jianyu Huang, Leslie Rice, Devin A. Matthews, and Robert A. van de Geijn. 2017. Generating Families of Practical Fast Matrix Multiplication Algorithms. In *31th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2017)*.

[8] Jianyu Huang, Tyler M. Smith, Greg M. Henry, and Robert A. van de Geijn. 2016. Strassen's Algorithm Reloaded. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 16)*. IEEE Press, Article 59, 12 pages.

[9] Dmitry I. Lyakh. 2015. An efficient tensor transpose algorithm for multicore CPU, Intel Xeon Phi, and NVidia Tesla GPU. *Comput. Phys. Commun.* 189 (2015), 84–91.

[10] Devin A Matthews. 2017. High-performance tensor contraction without Transposition. *SISC* (2017). Accepted.

[11] E. Peise, D. Fabregat-Traver, and P. Bientinesi. 2014. On the performance prediction of BLAS-based tensor contractions. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, S. A. Jarvis, S. A. Wright, and S. D. Hammond (Eds.). Number 8966 in Lecture Notes in Computer Science. Springer International Publishing, 193–212.

[12] 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 and Distributed Processing Symposium (IPDPS 2014)*.

[13] Paul Springer and Paolo Bientinesi. 2016. Design of a high-performance GEMM-like Tensor-Tensor Multiplication. *CoRR* abs/1607.00145 (2016).

[14] Paul Springer, Jeff R. Hammond, and Paolo Bientinesi. 2016. TTC: A high-performance Compiler for Tensor Transpositions. *CoRR* abs/1603.02297 (2016).

[15] Field G. Van Zee, Tyler Smith, Francisco D. Igual, Mikhail Smelyanskiy, Xianyi Zhang, Michael Kistler, Vernon Austel, John Gunnels, Tze Meng Low, Bryan Marker, Lee Killough, and Robert A. van de Geijn. 2016. The BLIS Framework: Experiments in Portability. *ACM Trans. Math. Software* 42, 2 (June 2016), 12:1–12:19.

[16] Field G. Van Zee and Robert A. van de Geijn. 2015. BLIS: A Framework for Rapidly Instantiating BLAS Functionality. *ACM Trans. Math. Soft.* 41, 3, Article 14 (June 2015), 33 pages.