Strassen's Algorithm for Tensor Contraction

Jianyu Huang, Devin A. Matthews, Robert A. van de Geijn

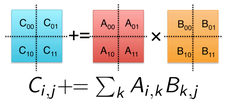
The University of Texas at Austin

Paper

Code

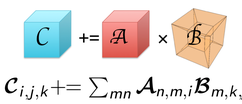
Acknowledgment

Matrix Multiplication



$C_{i,j} \mathrel{+}= \sum_k A_{i,k} B_{k,j}$

Tensor Contraction



$\mathcal{C}_{i,j,k} \mathrel{+}= \sum_{mn} \mathcal{A}_{n,m,i} \mathcal{B}_{m,k,j,n}$

Tensor contraction (TC) is an important computational kernel widely used in numerous applications. It is a multi-dimensional generalization of matrix multiplication (GEMM). While Strassen's algorithm for GEMM is well studied in theory and practice, extending it to accelerate TC has not been previously pursued. Thus, we believe this to be the first work to demonstrate how one can in practice speed up tensor contraction with Strassen's algorithm. By adopting a Block-Scatter-Matrix format, a novel matrix-centric tensor layout, we can conceptually view TC as GEMM for a general stride storage, with an implicit tensor-to-matrix transformation. This insight enables us to tailor a recent state-of-the-art implementation of Strassen's algorithm to a recent state-of-the-art TC, avoiding explicit transpositions (permutations) and extra workspace, and reducing the overhead of memory movement that is incurred. Performance benefits are demonstrated with a performance model as well as in practice on modern single core, multicore, and distributed memory parallel architectures, achieving up to 1.3× speedup. The resulting implementations can serve as a drop-in replacement for various applications with significant speedup.

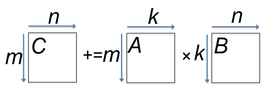
High-performance GEMM / Strassen's Algroithm

$M_0 := (A_{00}+A_{11})(B_{00}+B_{11});$
$M_1 := (A_{10}+A_{11})B_{00};$
$M_2 := A_{00}(B_{01}-B_{11});$
$M_3 := A_{11}(B_{10}-B_{00});$
$M_4 := (A_{00}+A_{01})B_{11};$
$M_5 := (A_{10}-A_{00})(B_{00}+B_{01});$
$M_6 := (A_{01}-A_{11})(B_{10}+B_{11});$
$C_{00} \mathrel{+}= M_0 + M_3 - M_4 + M_6$
$C_{01} \mathrel{+}= M_2 + M_4$
$C_{10} \mathrel{+}= M_1 + M_3$
$C_{11} \mathrel{+}= M_0 - M_1 + M_2 + M_5$

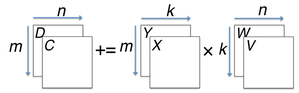
$M_0 := (A_{00}+A_{11})(B_{00}+B_{11});$ $C_{00} \mathrel{+}= M_0;$ $C_{11} \mathrel{+}= M_0;$
$M_1 := (A_{10}+A_{11})B_{00};$ $C_{10} \mathrel{+}= M_1;$ $C_{11} \mathrel{-}= M_1;$
$M_2 := A_{00}(B_{01}-B_{11});$ $C_{01} \mathrel{+}= M_2;$ $C_{11} \mathrel{+}= M_2;$
$M_3 := A_{11}(B_{10}-B_{00});$ $C_{00} \mathrel{+}= M_3;$ $C_{10} \mathrel{+}= M_3;$
$M_4 := (A_{00}+A_{01})B_{11};$ $C_{01} \mathrel{+}= M_4;$ $C_{00} \mathrel{-}= M_4;$
$M_5 := (A_{10}-A_{00})(B_{00}+B_{01});$ $C_{11} \mathrel{+}= M_5;$
$M_6 := (A_{01}-A_{11})(B_{10}+B_{11});$ $C_{00} \mathrel{+}= M_6;$

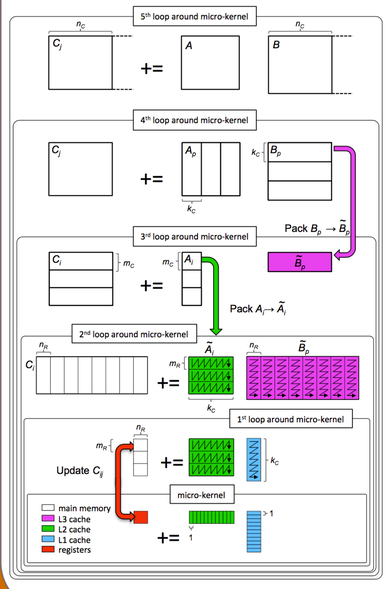
General Operation: $M := (X+\delta Y)(V+\varepsilon W);\ C \mathrel{+}= \gamma_0 M;\ D \mathrel{+}= \gamma_1 M;\ \gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\}.$

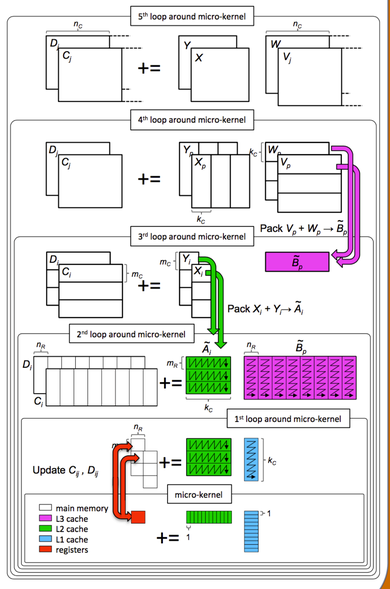
$C \mathrel{+}= AB;$



$M := (X+Y)(V+W);\ C \mathrel{+}= M;\ D \mathrel{+}= M;$







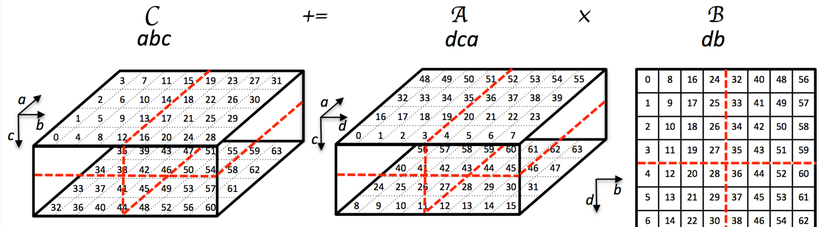
Tensors as Matrices: Block-Scatter-Matrix View

- Tensor: $\mathcal{A}_{dca}$, with $N_a = 4, N_c = 2, N_d = 8$. 8x2x4
  "d" dimension is stride-1, other dimensions have increasing strides (8, 16).
- Matrix: $A_{(ac)(d)}$, with $N_{I_m} = N_a \cdot N_c = 8, N_{P_k} = N_d = 8$. 8x8
  Column "ac" dimension has stride of "c" (8x2=16).
  Row "d" dimension has is stirde-1. (i.e. A is row-major.)
  Blocking lets us use the constant stride when possible, and a scattered algorithm otherwise.
- Matrix Multiplication: A, B, and C are eventually partitioned into small, fixed-sized blocks for packing or microkernel, such as 4x4 (picture below), 8x4, 6x8, etc.
- Tensor Contraction: The tensor blocks can be encoded into regular small matrix blocks whenever possible, so that no overhead is incurred.

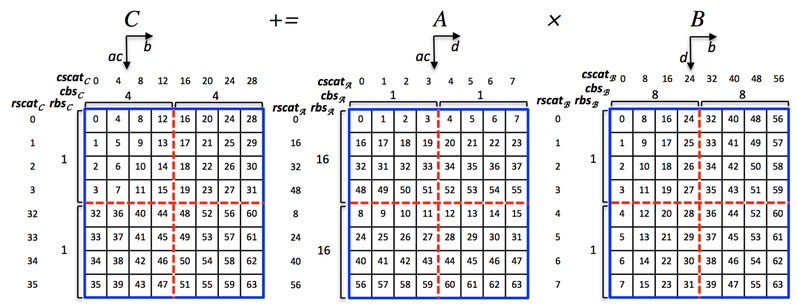
- $rscat_{\mathcal{A}}$, $cscat_{\mathcal{A}}$ store offset for each position in rows or columns.
  $\text{OFFSET}_{\mathcal{A}_{d,c,a}} = rscat_{A;(ac)} + cscat_{A;(d)}$
- $rbs_{\mathcal{A}}$, $cbs_{\mathcal{A}}$ store stride for each block or zero for irregular blocks.

The block scatter vectors help to utilize efficient SIMD vector load/store instructions for stride-1 index, or vector gather/scatter fetch instructions for stride-n index.

Strassen's Algorithm for Tensor Contraction



(a) Tensor contraction $\mathcal{C}_{a,b,c} \mathrel{+}= \mathcal{A}_{d,c,a} \cdot \mathcal{B}_{d,b}$ with $N_a = 4,\ N_b = N_d = 8$, and $N_c = 2$. The relative location of each data element in memory is given assuming a generalized column-major layout.



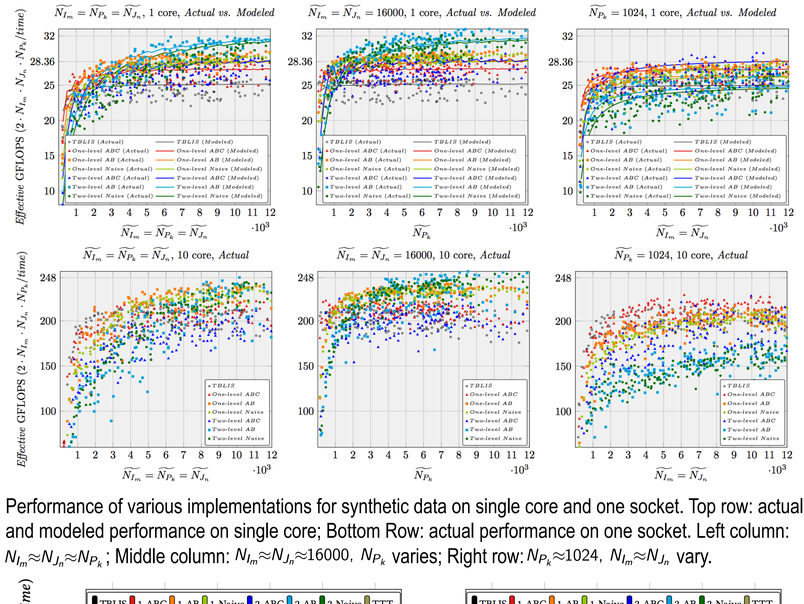
(b) Block scatter matrix view of **(a)**, where $\mathcal{A}_{d,c,a}$, $\mathcal{B}_{d,b}$, and $\mathcal{C}_{a,b,c}$ are mapped to matrices $A_{i,p}$, $B_{p,j}$, and $C_{i,j}$: $rscat_{\mathcal{T}}$ and $cscat_{\mathcal{T}}$ denote the scatter vectors; $rbs_{\mathcal{T}}$ and $cbs_{\mathcal{T}}$ denote the block scatter vectors. Element locations are given by the sum of the row and column scatter vector entries.

An example to illustrate Strassen's algorithm for tensor contraction. The red lines denotes Strassen's algorithm 2×2 partitions mapping from block scatter matrix view (bottom) to the original tensor (top). In this example the partitions are regular subtensors, but this is not required in general.

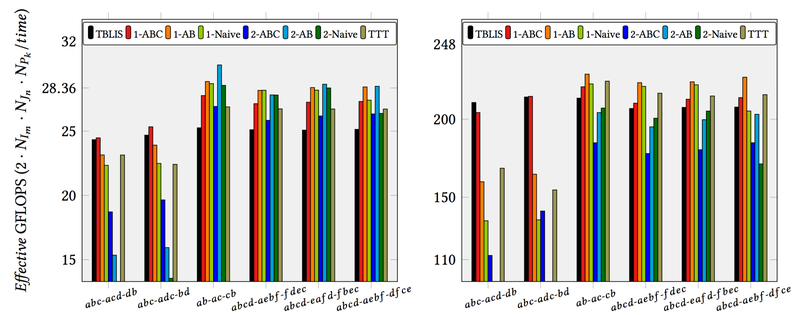
Implementation Variations

- Naïve Strassen TC: A traditional implementation with temporary buffers.
- AB Strassen TC: Integrate the addition of tensors into $\tilde{A}_i$ and $\tilde{B}_p$.
- ABC Strassen TC: AB Strassen TC. Additionally integrate the update of multiple submatrices of matrix representation of C in the micro-kernel.

Performance Experiments

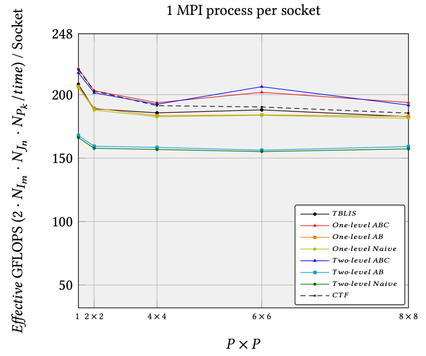


Performance of various implementations for synthetic data on single core and one socket. Top row: actual and modeled performance on single core; Bottom Row: actual performance on one socket. Left column: $N_{I_m} \approx N_{J_n} \approx N_{P_k}$; Middle column: $N_{I_m} \approx N_{J_n} \approx 16000$, N_{P_k} varies; Right row: $N_{P_k} \approx 1024$, $N_{I_m} \approx N_{J_n}$ vary.



Performance for representative user cases of benchmark from [4]. TC is identified by the index string, with the tensor index bundle of each tensor in the order $\mathcal{C}$-$\mathcal{A}$-$\mathcal{B}$, e.g. $\mathcal{C}_{abcd} \mathrel{+}= \mathcal{A}_{aebf} \mathcal{B}_{dfce}$ is denoted as *abcd-aebf-dfce*. Left: performance on single core. Right: performance on one socket.

$N_{I_m}(N_b \cdot N_j) = N_{P_k}(N_e \cdot N_m) = N_{J_n}(N_a \cdot N_i) \approx 16000 \cdot P$
on $P \times P$ MPI mesh
1 MPI process per socket



$\mathcal{Z} \mathrel{+}= \begin{pmatrix} \mathcal{W}_{e;0} & \cdots & \mathcal{W}_{e;K-1} \end{pmatrix} \begin{pmatrix} \mathcal{T}_{e;0} \\ \vdots \\ \mathcal{T}_{e;K-1} \end{pmatrix}$

Weak scalability performance result of the various implementations for a 4-D tensor contraction CCSD application on distributed $\mathcal{Z}_{abij} \mathrel{+}= \mathcal{W}_{bmej} \mathcal{T}_{aeim}$. CTF shows the performance of the Cyclops Tensor Framework (linked with Intel MKL).

[1] Jianyu Huang, Devin A. Matthews, and Robert A. van de Geijn. "Strassen's Algorithm for Tensor Contraction." *arXiv:1704.03092* (2017).
[2] Jianyu Huang, Tyler M. Smith, Greg H. Henry, and Robert A. van de Geijn. "Strassen's Algorithm Reloaded." In *SC'16*.
[3] Devin A. Matthews. "High-Performance Tensor Contraction without Transposition." Accepted in *SISC*.
[4] Paul Springer, and Paolo Bientinesi. "Design of a high-performance gemm-like tensor-tensor multiplication." *arXiv:1607.00145* (2016).
[5] Field G. Van Zee, and Robert A. van de Geijn. "BLIS: A framework for rapidly instantiating BLAS functionality." *TOMS* 41, no. 3 (2015): 14.