GEMM-like Tensor-Tensor Contraction (GETT)

Paul Springer and Paolo Bientinesi

Introduction

Tensor contractions (TC) are a performance critical component in numerous scientific computations. Despite the close connection between matrix-matrix products (GEMM) and TCs, the performance of the latter 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 multiplication (GETT) [1].

Tensor Contraction

- Tensor contractions are the generalization of matrix-matrix multiplications to higher dimensions.

Let $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ be multi-dimensional tensors, a tensor contraction is:

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

where $I_\mathcal{A}$, $I_\mathcal{B}$, and $I_\mathcal{C}$ denote symbolic index sets. We further define three important index sets:

- Free indices of $\mathcal{A}$, $I_m := I_\mathcal{A} \cap I_\mathcal{C}$
- Free indices of $\mathcal{B}$, $I_n := I_\mathcal{B} \cap I_\mathcal{C}$
- Contracted indices, $I_k := I_\mathcal{A} \cap I_\mathcal{B}$.

All indices of I_k are implicitly summed.

Example: Matrix-Matrix Multiplication.
$\mathcal{C}_{mn} \leftarrow \mathcal{A}_{mk} \times \mathcal{B}_{kn}$
with $I_m = \{m\}$, $I_n = \{n\}$, and $I_k = \{k\}$.

Example: Tensor Contraction.
$\mathcal{C}_{m_1n_1m_2n_2} \leftarrow \mathcal{A}_{m_2k_1m_1k_2} \times \mathcal{B}_{n_1n_2k_2k_1}$
with $I_m = \{m_1, m_2\}$, $I_n = \{n_1, n_2\}$, and $I_k = \{k_1, k_2\}$.

Prior Approaches

- **T**ranspose-**T**ranspose-**G**EMM-**T**ranspose (TTGT)
  - Flatten tensors into matrices
  - Contract via `GEMM`
  - Fold result back into a tensor
  - Adopters: Cyclops Tensor Framework, Tensor Toolbox, LibTensor
- **L**oop-**o**ver-**G**EMM (LoG)
  - Identify 2D slices of the tensors
  - Contract all slices via `GEMM`

Both approaches exhibit significant disadvantages:

- **T**ranspose-**T**ranspose-**G**EMM-**T**ranspose
  - Suboptimal I/O cost
  - Flattening accounts for pure overhead
  - Auxiliary memory for flattened tensors
- **L**oop-**o**ver-**G**EMM (LoG)
  - Suboptimal I/O cost
  - May require strided `GEMM`

Classification

Tensor contractions can be mapped to BLAS kernels:

- `GEMM`: $I_m \neq \emptyset$ **and** $I_n \neq \emptyset$ **and** $I_k \neq \emptyset$
- `GEMV`: ($I_m = \emptyset$ **or** $I_n = \emptyset$) **and** $I_k \neq \emptyset$
- `GER`: $I_m \neq \emptyset$ **and** $I_n \neq \emptyset$ **and** $I_k = \emptyset$
- `AXPY`: ($I_m = \emptyset$ **or** $I_n = \emptyset$) **and** $I_k = \emptyset$
- `DOT`: else.

GETT's Key Features

GETT constitutes a *native* tensor contraction algorithm that does not rely on an available GEMM implementation. The key features are:

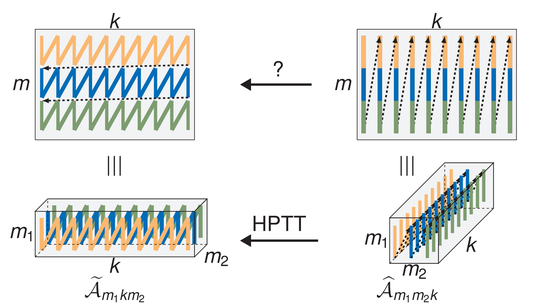
- GEMM-like design
- Multi-dimensional packing routines
  - Packing via tensor transpositions [2]
  - Preserve stride-1 index
  - Explicitly vectorized

Design

- Similar to a high-performance GEMM
- Complexity offloaded into packing routines

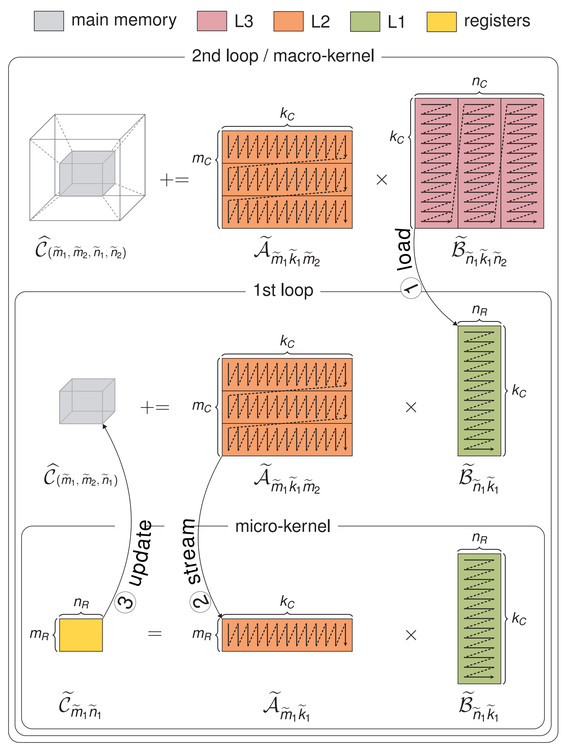
```
// N-Loop
for n = 1 : nC : Sln                                   ①
   // K-Loop (contracted)
   for k = 1 : kC : Slk                                ②
      B̂ = identify_subtensor(B, n, k)
      // pack B̂ into B̃
      B̃ = packB(B̂)                                     ③
      // M-Loop
      for m = 1 : mC : Slm                             ④
         Â = identify_subtensor(A, m, k)
         // pack Â into Ã
         Ã = packA(Â)                                  ⑤
         Ĉ = identify_subtensor(C, m, n)
         // matrix-matrix product: Ĉ ← αÃ × B̃ + βĈ
         macroKernel(Ã, B̃, Ĉ, α, β)                    ⑥
```

Packing Routines



Macro- & Micro-Kernel

- Blocking for caches
- Blocking for registers
- Explicitly vectorized



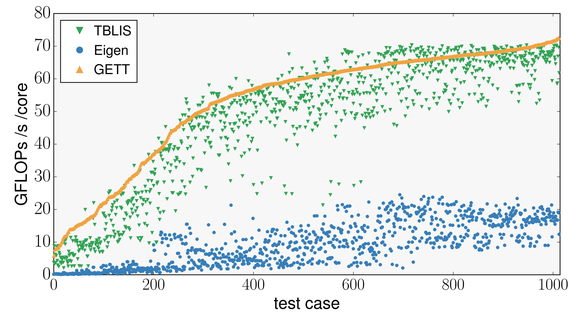
Shared-Memory Parallelism

- Parallelism is similar to that of BLIS [4]
  - All loops can be parallelized individually
- GETT also parallelizes along the contracted dimensions
  - Exposes additional parallelism if $\mathcal{C}$ is very small
- A promising *parallelism strategy* is chosen according to a cost model:
  - Maximize load balancing
  - Stay within the cache limitations

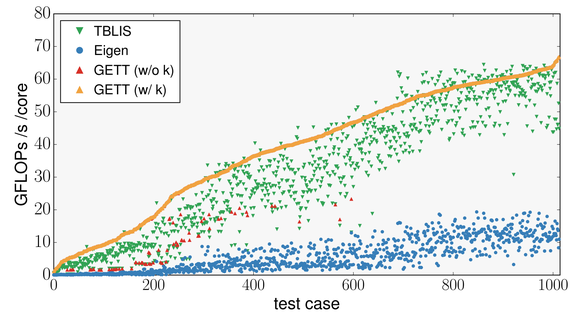
Future Work

- Implement GETT as part of a C++ library

Performance



(a) 1 core



(b) 24 cores

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

	Single-Threaded	Multi-Threaded
TBLIS [3]	1.3×	1.6×
Eigen [5]	13.2×	16.2×
Direct [1]	-	52.3×

Table: GETT's average speedups.

References

[1] P. Springer and P. Bientinesi.
Design of a high-performance GEMM-like Tensor-Tensor Multiplication. 2016.
`http://arxiv.org/abs/1607.00145`.

[2] P. Springer, T. Su, and P. Bientinesi.
HPTT: A High-Performance Tensor Transposition C++ Library. *ACM Array Workshop*, 2017. DOI: 10.1145/3091966.3091968.

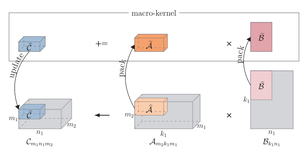
[3] Devin A. Matthews
High-Performance Tensor Contraction without BLAS. 2016.
`http://arxiv.org/abs/1607.00291`.

[4] Tyler M. Smith et al.
Anatomy of High-Performance Many-Threaded Matrix Multiplication. IPDPS'14. DOI: 10.1109/IPDPS.2014.110

[5] Gaël Guennebaud and Benoît Jacob et al.
Eigen v3.3.4. `http://eigen.tuxfamily.org`

Paul Springer
Telefon: +49 241 80 99 141
Schinkelstr. 2, 52062 Aachen, GERMANY
E-Mail: springer@aices.rwth-aachen.de

GitHub



HPAC High Performance and Automatic Computing

RWTH AACHEN UNIVERSITY