The Landscape of High-Performance Tensor Contractions

Paul Springer and Paolo Bientinesi

Aachen Institute for Advanced Study in Computational Engineering Science

Atlanta, Feb. 24th 2017
Introduction

- A tensor is a multidimensional array:
  - 0-order tensor: scalar $\alpha$
A tensor is a multidimensional array:
- 0-order tensor: scalar $\alpha$
- 1-order tensor: vector $A_{i_1}$
A tensors is a multidimensional array:

- 0-order tensor: scalar $\alpha$
- 1-order tensor: vector $A_{i_1}$
- 2-order tensor: matrix $A_{i_1,i_2}$
A tensors is a multidimensional array:
- 0-order tensor: scalar $\alpha$
- 1-order tensor: vector $\mathbf{A}_{i_1}$
- 2-order tensor: matrix $\mathbf{A}_{i_1,i_2}$
- $n$-order tensor: $\mathbf{A}_{i_1,i_2,\ldots,i_n}$

Tensor contractions can be thought of as generalized GEMMs

Three approaches to tensor contractions:
- Nested loops
- Loops over GEMM (LoG)
- Transpose-Transpose-GEMM-Transpose (TTGT)

We propose a novel approach: GETT

Akin to a high-performance GEMM implementation

---

Introduction

- A tensor is a multidimensional array:
  - 0-order tensor: scalar $\alpha$
  - 1-order tensor: vector $A_{i_1}$
  - 2-order tensor: matrix $A_{i_1,i_2}$
  - $n$-order tensor: $A_{i_1,i_2,...,i_n}$

- Tensor contractions can be thought of as generalized GEMMs

---

A tensor is a multidimensional array:

- 0-order tensor: scalar $\alpha$
- 1-order tensor: vector $A_{i_1}$
- 2-order tensor: matrix $A_{i_1, i_2}$
- $n$-order tensor: $A_{i_1, i_2, \ldots, i_n}$

Tensor contractions can be thought of as generalized GEMMs.

Three approaches to tensor contractions:

- Nested loops
- Loops over GEMM (LoG)
- Transpose-Transpose-GEMM-Transpose (TTGT)

---

A tensors is a multidimensional array:
- 0-order tensor: scalar $\alpha$
- 1-order tensor: vector $A_{i_1}$
- 2-order tensor: matrix $A_{i_1,i_2}$
- $n$-order tensor: $A_{i_1,i_2,...,i_n}$

Tensor contractions can be thought of as generalized GEMMs

Three approaches to tensor contractions:
- Nested loops
- Loops over GEMM (LoG)
- Transpose-Transpose-GEMM-Transpose (TTGT)

We propose a novel approach: GETT\(^1\)
- Akin to a high-performance GEMM implementation

Approaches to Tensor Contractions:
- Loops over GEMM (LoG)
- Transpose-Transpose-GEMM-Transpose (TTGT)
- GEMM-like Tensor-Tensor Multiply (GETT)

Tensor Contraction Code Generator

Performance Evaluation

Source code available at: https://github.com/HPAC/tccg
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

\[ C_{m_1,n_1} \leftarrow \sum_{k_1} A_{m_1,k_1} B_{k_1,n_1} \]
Loop over GEMM (LoG)

Conceptual Idea
Identify 2D subtensors and contract them via GEMM

\[ C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1} \]
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

\[ C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1} \]

\[
\text{gemm}(M_1, N_1, K_1, A[:, :], B[:, :], C[:, :])
\]
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

\[ C_{m_1,n_1} \leftarrow A_{m_1,k_1}B_{k_1,n_1} \]
\[ C_{m_1,m_2,n_1} \leftarrow A_{m_1,m_2,k_1}B_{k_1,n_1} \]
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

\[ C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1} \]
\[ C_{(m_1,m_2),n_1} \leftarrow A_{(m_1,m_2),k_1} B_{k_1,n_1} \]

\[
gemm(M_1 \times M_2, N_1, K_1, A[:,:), B[:,:), C[:,:])
\]
Loop over GEMM (LoG)

**Conceptual Idea**

Identify 2D subtensors and contract them via GEMM

- \( C_{m_1, n_1} \leftarrow A_{m_1, k_1} B_{k_1, n_1} \)
- \( C_{(m_1, m_2), n_1} \leftarrow A_{(m_1, m_2), k_1} B_{k_1, n_1} \)
- \( C_{m_1, n_1, n_2, m_2} \leftarrow A_{m_1, m_2, k_1} B_{k_1, n_2, n_1} \)

for \( m_2 = 0; m_2 < M_2; m_2++ \)
  for \( n_1 = 0; n_1 < N_1; n_1++ \)
    \[ \text{gemm}(M_1, N_2, K_1, A[:, m_2, :], B[:, :, n_1], C[:, n_1 :, m_2]) \]
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

\[
\begin{align*}
C_{m_1,n_1} & \leftarrow A_{m_1,k_1}B_{k_1,n_1} \\
C_{(m_1,m_2),n_1} & \leftarrow A_{(m_1,m_2),k_1}B_{k_1,n_1} \\
C_{m_1,n_1,n_2,m_2} & \leftarrow A_{m_1,m_2,k_1}B_{k_1,n_2,n_1}
\end{align*}
\]

\[
\text{for } (m_2 = 0; m_2 < M_2; m_2++ ) \\
\quad \text{for } (n_2 = 0; n_2 < N_2; n_2++ ) \\
\quad \text{gemm}(M_1, N_1, K_1, A[:,m_2,:], B[:,n_2,:], C[:,n_2,m_2])
\]
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

\[ C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1} \]
\[ C_{(m_1,m_2),n_1} \leftarrow A_{(m_1,m_2),k_1} B_{k_1,n_1} \]
\[ C_{m_1,n_1,n_2,m_2} \leftarrow A_{m_1,m_2,k_1} B_{k_1,n_2,n_1} \]

\[
\text{for ( } m_2 = 0; m_2 < M_2; m_2++ \text{ )}
\quad \text{gemm\_batch}(M_1, N_1, K_1, A[::,m_2,:], B[::,n_2,:], C[::,n_2,m_2], N_2)
\]
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- \( C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1} \)
- \( C_{(m_1,m_2),n_1} \leftarrow A_{(m_1,m_2),k_1} B_{k_1,n_1} \)
- \( C_{m_1,n_1,n_2,m_2} \leftarrow A_{m_1,m_2,k_1} B_{k_1,n_2,n_1} \)

```latex
\textbf{for} ( \ n_2 = 0; \ n_2 < N_2; \ n_2++ \ )
\textbf{gemm\_batch} (M_1, \ N_1, \ K_1, \ A[::,m_2,:], \ B[::,n_2,:], \ C[::,n_2,m_2], \ M_2) 
```
Loop over GEMM (LoG)

Conceptual Idea
Identify 2D subtensors and contract them via GEMM

- $C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1}$
- $C(m_1,m_2),n_1 \leftarrow A(m_1,m_2),k_1 B_{k_1,n_1}$
- $C_{m_1,n_1,n_2,m_2} \leftarrow A_{m_1,m_2,k_1} B_{k_1,n_2,n_1}$
- $C_{m_1,n_1} \leftarrow A_{k_1,m_1,k_2} B_{k_2,n_1,k_1}$

\[
\text{for ( } k_1 = 0; \ k_1 < K_1; \ k_1 ++ ) \ \\
\text{gemm} ( M_1, \ N_1, \ K_2, \ A[ k_1, :, :], \ B[ :, :, k_1 ], \ C[ :, :] )
\]
Loop over GEMM (LoG)

**Conceptual Idea**

Identify 2D subtensors and contract them via GEMM

- $C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1}$
- $C_{m_1,m_2,n_1} \leftarrow A_{m_1,m_2,k_1} B_{k_1,n_1}$
- $C_{m_1,n_1,n_2,m_2} \leftarrow A_{m_1,m_2,k_1} B_{k_1,n_2,n_1}$
- $C_{m_1,n_1,m_2} \leftarrow A_{k_1,m_1,k_2} B_{k_2,n_1,k_1}$

```c
for (k_1 = 0; k_1 < K_1; k_1++)
gemm(M_1, N_1, K_2, A[k_1,:,:], B[:, :, k_1], C[:, :])
```
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- \( C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1} \)
- \( C_{(m_1,m_2),n_1} \leftarrow A_{(m_1,m_2),k_1} B_{k_1,n_1} \)
- \( C_{m_1,n_1,n_2,m_2} \leftarrow A_{m_1,m_2,k_1} B_{k_1,n_2,n_1} \)
- \( C_{m_1,n_1} \leftarrow A_{k_1,m_1,k_2} B_{k_2,n_1,k_1} \)

\[
\text{for}( \ k_2 = 0; \ k_2 < K_2; \ k_2++ \ )
\text{gemm}(M_1, N_1, K_1, A[:,:,k_2]^T, B[k_2,:,:]^T, C[:,:,])
\]
Loop over GEMM (LoG)

**Conceptual Idea**

Identify 2D subtensors and contract them via GEMM

- $C_{m_1,n_1} \leftarrow A_{m_1,k_1} B_{k_1,n_1}$
- $C_{(m_1,m_2),n_1} \leftarrow A_{(m_1,m_2),k_1} B_{k_1,n_1}$
- $C_{m_1,n_1,n_2,m_2} \leftarrow A_{m_1,m_2,k_1} B_{k_1,n_2,n_1}$
- $C_{m_1,n_1} \leftarrow A_{k_1,m_1,k_2} B_{k_2,n_1,k_1}$

\[
\text{for ( } k_2 = 0; k_2 < K_2; k_2++ \text{ )} \\
gemm ( M_1, N_1, K_1, A[:, :, k_2]^T, B[k_2, :, :]^T, C[:, :])
\]
Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $C_{m_1, n_1} \leftarrow A_{m_1, k_1} B_{k_1, n_1}$
- $C_{(m_1, m_2), n_1} \leftarrow A_{(m_1, m_2), k_1} B_{k_1, n_1}$
- $C_{m_1, n_1, n_2, m_2} \leftarrow A_{m_1, m_2, k_1} B_{k_1, n_2, n_1}$
- $C_{m_1, n_1} \leftarrow A_{k_1, m_1, k_2} B_{k_2, n_1, k_1}$

```
for ( n = 0; n < N_1; n++ )
  for ( k_2 = 0; k_2 < K_2; k_2++ )
    gemv ( M_1, K_1, A[:, :, k_2]^T, B[k_2, n_1, :], C[:, n] )
```
Loop Over GEMM (LoG)

- Search space:
Search space:
- GEMM indices: $m$, $n$, $k$
Search space:
- GEMM indices: $m, n, k$
- Loop order
Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: $m, n, k$
  - Loop order
  - Batched index
Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: $m$, $n$, $k$
  - Loop order
  - Batched index

- Advantages:
Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: \( m, n, k \)
  - Loop order
  - Batched index

- Advantages:
  - Easy to implement

Exploits existing BLAS libraries
No additional memory required

Some contractions cannot be implemented via straight LoG
GEMM’s arithmetic intensity can be suboptimal
Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: $m$, $n$, $k$
  - Loop order
  - Batched index

- Advantages:
  - Easy to implement
  - Exploits existing BLAS libraries
Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: \( m, n, k \)
  - Loop order
  - Batched index

- Advantages:
  - Easy to implement
  - Exploits existing BLAS libraries
  - No additional memory required
Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: $m, n, k$
  - Loop order
  - Batched index

- Advantages:
  - Easy to implement
  - Exploits existing BLAS libraries
  - No additional memory required

- Disadvantages:
Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: \( m, n, k \)
  - Loop order
  - Batched index

- Advantages:
  - Easy to implement
  - Exploits existing BLAS libraries
  - No additional memory required

- Disadvantages:
  - Some contractions cannot be implemented via straight LoG
Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: \( m, n, k \)
  - Loop order
  - Batched index

- Advantages:
  - Easy to implement
  - Exploits existing BLAS libraries
  - No additional memory required

- Disadvantages:
  - Some contractions cannot be implemented via straight LoG
  - GEMM’s arithmetic intensity can be suboptimal
Map Tensor Contractions to BLAS

- Free indices of $A$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_A \cap I_C$

---

Map Tensor Contractions to BLAS

- Free indices of $A$
  - $l_m := \{m_1, m_2, ..., m_\gamma\} = l_A \cap l_C$

- Free indices of $B$
  - $l_n := \{n_1, n_2, ..., n_\zeta\} = l_B \cap l_C$

---

3 Di Napoli et al. “Towards an Efficient Use of the BLAS Library for Multilinear Tensor Contractions”
4 Yang Shi et al. “Tensor Contractions with Extended BLAS Kernels on CPU and GPU”
Map Tensor Contractions to BLAS

- Free indices of $A$
  - $l_m := \{m_1, m_2, ..., m_\gamma\} = l_A \cap l_C$

- Free indices of $B$
  - $l_n := \{n_1, n_2, ..., n_\zeta\} = l_B \cap l_C$

- Contracted indices
  - $l_k := \{k_1, k_2, ..., k_\xi\} = l_A \cap l_B$

---

3 Di Napoli et al. “Towards an Efficient Use of the BLAS Library for Multilinear Tensor Contractions”
4 Yang Shi et al. “Tensor Contractions with Extended BLAS Kernels on CPU and GPU”
Map Tensor Contractions to BLAS

- Free indices of $A$
  
  \[ l_m := \{m_1, m_2, ..., m_\gamma\} = l_A \cap l_C \]

- Free indices of $B$

  \[ l_n := \{n_1, n_2, ..., n_\zeta\} = l_B \cap l_C \]

- Contracted indices

  \[ l_k := \{k_1, k_2, ..., k_\xi\} = l_A \cap l_B \]

- Tensor contractions can be mapped to BLAS routines$^3$,$^4$:
  
  - GEMM: \( l_m \neq \emptyset \) and \( l_n \neq \emptyset \) and \( l_k \neq \emptyset \).

---

$^3$Di Napoli et al. “Towards an Efficient Use of the BLAS Library for Multilinear Tensor Contractions”

$^4$Yang Shi et al. “Tensor Contractions with Extended BLAS Kernels on CPU and GPU”
Map Tensor Contractions to BLAS

- Free indices of $A$
  - $l_m := \{m_1, m_2, ..., m_\gamma\} = l_A \cap l_C$

- Free indices of $B$
  - $l_n := \{n_1, n_2, ..., n_\zeta\} = l_B \cap l_C$

- Contracted indices
  - $l_k := \{k_1, k_2, ..., k_\xi\} = l_A \cap l_B$

- Tensor contractions can be mapped to BLAS routines\(^3,4\):
  - **GEMM**: $l_m \neq \emptyset$ and $l_n \neq \emptyset$ and $l_k \neq \emptyset$.
  - **GEMV**: $(l_m = \emptyset \text{ or } l_n = \emptyset)$ and $l_k \neq \emptyset$

---

\(^3\) Di Napoli et al. “Towards an Efficient Use of the BLAS Library for Multilinear Tensor Contractions”

\(^4\) Yang Shi et al. “Tensor Contractions with Extended BLAS Kernels on CPU and GPU”
Map Tensor Contractions to BLAS

- Free indices of $A$
  - $l_m := \{m_1, m_2, ..., m_\gamma\} = l_A \cap l_C$

- Free indices of $B$
  - $l_n := \{n_1, n_2, ..., n_\zeta\} = l_B \cap l_C$

- Contracted indices
  - $l_k := \{k_1, k_2, ..., k_\xi\} = l_A \cap l_B$

- Tensor contractions can be mapped to BLAS routines\(^3\)\(^4\):
  - **GEMM:** $l_m \neq \emptyset$ \textbf{and} $l_n \neq \emptyset$ \textbf{and} $l_k \neq \emptyset$.
  - **GEMV:** ($l_m = \emptyset$ \textbf{or} $l_n = \emptyset$) \textbf{and} $l_k \neq \emptyset$
  - **GER:** $l_m \neq \emptyset$ \textbf{and} $l_n \neq \emptyset$ \textbf{and} $l_k = \emptyset$

\(^3\)Di Napoli et al. “Towards an Efficient Use of the BLAS Library for Multilinear Tensor Contractions”
\(^4\)Yang Shi et al. “Tensor Contractions with Extended BLAS Kernels on CPU and GPU”
Free indices of $A$
- $l_m := \{m_1, m_2, ..., m_\gamma\} = l_A \cap l_C$

Free indices of $B$
- $l_n := \{n_1, n_2, ..., n_\zeta\} = l_B \cap l_C$

Contracted indices
- $l_k := \{k_1, k_2, ..., k_\xi\} = l_A \cap l_B$

Tensor contractions can be mapped to BLAS routines$^{3,4}$:
- **GEMM**: $l_m \neq \emptyset$ and $l_n \neq \emptyset$ and $l_k \neq \emptyset$.
- **GEMV**: $(l_m = \emptyset$ or $l_n = \emptyset)$ and $l_k \neq \emptyset$
- **GER**: $l_m \neq \emptyset$ and $l_n \neq \emptyset$ and $l_k = \emptyset$
- **AXPY**: $(l_m = \emptyset$ or $l_n = \emptyset)$ and $l_k = \emptyset$

---

$^3$Di Napoli et al. “Towards an Efficient Use of the BLAS Library for Multilinear Tensor Contractions”

$^4$Yang Shi et al. “Tensor Contractions with Extended BLAS Kernels on CPU and GPU”
Free indices of $A$
- $l_m := \{m_1, m_2, \ldots, m_\gamma\} = l_A \cap l_C$

Free indices of $B$
- $l_n := \{n_1, n_2, \ldots, n_\zeta\} = l_B \cap l_C$

Contracted indices
- $l_k := \{k_1, k_2, \ldots, k_\eta\} = l_A \cap l_B$

Tensor contractions can be mapped to BLAS routines$^3$,$^4$:
- **GEMM**: $l_m \neq \emptyset$ and $l_n \neq \emptyset$ and $l_k \neq \emptyset$.
- **GEMV**: $(l_m = \emptyset$ or $l_n = \emptyset$) and $l_k \neq \emptyset$
- **GER**: $l_m \neq \emptyset$ and $l_n \neq \emptyset$ and $l_k = \emptyset$
- **AXPY**: $(l_m = \emptyset$ or $l_n = \emptyset$) and $l_k = \emptyset$
- **DOT**: else.

---

$^3$ Di Napoli et al. “Towards an Efficient Use of the BLAS Library for Multilinear Tensor Contractions”

$^4$ Yang Shi et al. “Tensor Contractions with Extended BLAS Kernels on CPU and GPU”
Conceptual Idea

1. "Flatten" the tensors to matrices
2. Use GEMM for contraction
3. "Unflatten" output matrix to tensor
Conceptual Idea

1. "Flatten" the tensors to matrices
2. Use GEMM for contraction
3. "Unflatten" output matrix to tensor

$$C_{m_1,n_1} \leftarrow A_{k_1,m_1,k_2} B_{k_2,n_1,k_1}$$
Conceptual Idea

1. “Flatten” the tensors to matrices
2. Use GEMM for contraction
3. “Unflatten” output matrix to tensor

\[
C_{m_1,n_1} \leftarrow A_{k_1,m_1,k_2}B_{k_2,n_1,k_1}
\]

\[
\tilde{A}_{m_1,(k_1,k_2)} \leftarrow A_{k_1,m_1,k_2}
\]
\[
\tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1}
\]
\[
gemm(M_1, N_1, K_1 \times K_2, \tilde{A}, \tilde{B}, C)
\]
Conceptual Idea

1. "Flatten" the tensors to matrices
2. Use GEMM for contraction
3. "Unflatten" output matrix to tensor

\[ C_{m_1,n_1} \leftarrow A_{k_1,m_1,k_2} B_{k_2,n_1,k_1} \]

\[ \tilde{A}_{m_1,(k_1,k_2)} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{A}, \tilde{B}, C) \]

\[ \tilde{A}_{(k_1,k_2),m_1} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{A}^T, \tilde{B}, C) \]
Transpose-Transpose-GEMM-Transpose (TTGT)

Conceptual Idea

1. “Flatten” the tensors to matrices
2. Use GEMM for contraction
3. “Unflatten” output matrix to tensor

\[ C_{m_1,n_1} \leftarrow A_{k_1,m_1,k_2} B_{k_2,n_1,k_1} \]

\[ \tilde{A}_{m_1,(k_1,k_2)} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{A}, \tilde{B}, C) \]

\[ \tilde{A}_{(k_1,k_2),m_1} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{A}^T, \tilde{B}, C) \]
Conceptual Idea
1. “Flatten” the tensors to matrices
2. Use GEMM for contraction
3. “Unflatten” output matrix to tensor

\[ C_{m_1,n_1} \leftarrow \tilde{A}_{k_1,m_1,k_2} \tilde{B}_{k_2,n_1,k_1} \]

\[ \tilde{A}_{m_1,(k_1,k_2)} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{A}, \tilde{B}, \tilde{C}) \]

\[ \tilde{A}_{(k_1,k_2),m_1} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{A}^T, \tilde{B}, \tilde{C}) \]

\[ \tilde{A}_{(k_2,k_1),m_1} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_2,k_1),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{B}^T, \tilde{A}, \tilde{C}) \]
\[ C_{m_1,n_1} \leftarrow \tilde{C}_{n_1,m_1} \]
Conceptual Idea

1. “Flatten” the tensors to matrices
2. Use GEMM for contraction
3. “Unflatten” output matrix to tensor

\[ C_{m_1,n_1} \leftarrow A_{k_1,m_1,k_2} B_{k_2,n_1,k_1} \]

\[ \tilde{A}_{m_1,(k_1,k_2)} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{A}, \tilde{B}, C) \]

\[ \tilde{A}_{(k_1,k_2),m_1} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_1,k_2),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{B}^T, \tilde{A}, \tilde{C}) \]

\[ \tilde{A}_{(k_2,k_1),m_1} \leftarrow A_{k_1,m_1,k_2} \]
\[ \tilde{B}_{(k_2,k_1),n_1} \leftarrow B_{k_2,n_1,k_1} \]
\[ \text{gemm}(M_1, N_1, K_1 \times K_2, \tilde{B}^T, \tilde{A}, \tilde{C}) \]
\[ C_{m_1,n_1} \leftarrow \tilde{C}_{n_1,m_1} \]

... and more.
Transpose-Transpose-GEMM-Transpose (TTGT)

- Search space:

---


Paul Springer (AICES)  High-Performance Tensor Contractions  Feb. 24th 2017  8 / 17
Search space:
- Any permutation of \( I_m, I_n, I_k \)

Advantages:
- Easy to implement
- Exploits existing BLAS libraries
- All TCs can be implemented via TTGT

Disadvantages:
- Transpositions account for pure overhead
- Additional memory required
Search space:
- Any permutation of $I_m, I_n, I_k$
- Transposed $\mathcal{A}$

Advantages:
- Easy to implement
- Exploits existing BLAS libraries
- All TCs can be implemented via TTGT
- Large GEMM $\Rightarrow$ good performance?

Disadvantages:
- Transpositions account for pure overhead
- Additional memory required
Search space:
- Any permutation of $I_m, I_n, I_k$
- Transposed $A$
- Transposed $B$
Search space:

- Any permutation of $l_m, l_n, l_k$
- Transposed $A$
- Transposed $B$
- Interchange $A$ and $B$ within GEMM

Advantages:
- Easy to implement
- Exploits existing BLAS libraries
- All TCs can be implemented via TTGT

Disadvantages:
- Transpositions account for pure overhead
- Additional memory required

---

Search space:
- Any permutation of $I_m, I_n, I_k$
- Transposed $A$
- Transposed $B$
- Interchange $A$ and $B$ within GEMM

Advantages:
Search space:
- Any permutation of $l_m, l_n, l_k$
- Transposed $A$
- Transposed $B$
- Interchange $A$ and $B$ within GEMM

Advantages:
- Easy to implement

---

Search space:
- Any permutation of $l_m, l_n, l_k$
- Transposed $A$
- Transposed $B$
- Interchange $A$ and $B$ within GEMM

Advantages:
- Easy to implement
- Exploits existing BLAS libraries
Search space:
- Any permutation of $l_m, l_n, l_k$
- Transposed $A$
- Transposed $B$
- Interchange $A$ and $B$ within GEMM

Advantages:
- Easy to implement
- Exploits existing BLAS libraries
- All TCs can be implemented via TTGT

---

**Transpose-Transpose-GEMM-Transpose (TTGT)**

- **Search space:**
  - Any permutation of $I_m, I_n, I_k$
  - Transposed $A$
  - Transposed $B$
  - Interchange $A$ and $B$ within GEMM

- **Advantages:**
  - Easy to implement
  - Exploits existing BLAS libraries
  - All TCs can be implemented via TTGT
  - Large GEMM $\Rightarrow$ good performance?

---

Search space:
- Any permutation of $l_m, l_n, l_k$
- Transposed $A$
- Transposed $B$
- Interchange $A$ and $B$ within GEMM

Advantages:
- Easy to implement
- Exploits existing BLAS libraries
- All TCs can be implemented via TTGT
- Large GEMM $\Rightarrow$ good performance?

Disadvantages:

---

Search space:
- Any permutation of $l_m, l_n, l_k$
- Transposed $\mathcal{A}$
- Transposed $\mathcal{B}$
- Interchange $\mathcal{A}$ and $\mathcal{B}$ within GEMM

Advantages:
- Easy to implement
- Exploits existing BLAS libraries
- All TCs can be implemented via TTGT
- Large GEMM $\Rightarrow$ good performance?

Disadvantages:
- Transpositions\(^5\) account for pure overhead

---

Search space:
- Any permutation of $I_m, I_n, I_k$
- Transposed $A$
- Transposed $B$
- Interchange $A$ and $B$ within GEMM

Advantages:
- Easy to implement
- Exploits existing BLAS libraries
- All TCs can be implemented via TTGT
- Large GEMM $\Rightarrow$ good performance?

Disadvantages:
- Transpositions\(^5\) account for pure overhead
- Additional memory required

GEMM-like Tensor-Tensor Multiplication (GETT)

Key Idea

- Eliminate explicit transpositions
- Pack-and-transpose while moving data into the caches\(^5\)
  \[\Rightarrow\] Complexity offloaded into packing routines

---

GEMM-like Tensor-Tensor Multiplication (GETT)

Key Idea

- Eliminate explicit transpositions
- Pack-and-transpose while moving data into the caches\(^5\)
  \[ \Rightarrow \] Complexity offloaded into packing routines

```c
// N-Loop
for n = 1 : nc : S_{ln}

// K-Loop (contracted)
for k = 1 : kc : S_{lk}
  \hat{B} = identify_subtensor(B, n, k)
  // pack \hat{B} into \tilde{B} (L3 cache)
  \tilde{B} = packB(\hat{B})

// M-Loop
for m = 1 : mc : S_{lm}
  \hat{A} = identify_subtensor(A, m, k)
  // pack \hat{A} into \tilde{A} (L2 cache)
  \tilde{A} = packA(\hat{A})
  \hat{C} = identify_subtensor(C, m, n)
  // compute matrix-matrix product of \tilde{A}\tilde{B}
  macroKernel(\tilde{A}, \tilde{B}, \hat{C}, \alpha, \beta)
```

Key Idea

- Eliminate explicit transpositions
- Pack-and-transpose while moving data into the caches
  \[ \Rightarrow \] Complexity offloaded into packing routines

```c
// N-Loop
for n = 1 : nc : S_in
  // pack \( \hat{A} \) into \( \tilde{A} \) (L2 cache)
  \( \hat{A} = \text{identify_subtensor}(A, m, k) \)
  \( \tilde{A} = \text{packA}(\hat{A}) \)

// M-Loop
for m = 1 : mc : S_im
  \( \hat{C} = \text{identify_subtensor}(C, m, n) \)
  \( \text{macroKernel}(\tilde{A}, \tilde{B}, \hat{C}, \alpha, \beta) \)
```

---

Search space:
GEMM-like Tensor-Tensor Multiplication (GETT)

- Search space:
  - Blocking parameters: \( mc, nc, kc \)

Advantages:
- Same arithmetic intensity as GEMM
- No memory overhead

Disadvantages:
- Complex to implement
GEMM-like Tensor-Tensor Multiplication (GETT)

- Search space:
  - Blocking parameters: $mc$, $nc$, $kc$
  - Subtensors $\hat{A}$, $\hat{B}$, $\hat{C}$

Advantages:
- Same arithmetic intensity as GEMM
- No memory overhead

Disadvantages:
- Complex to implement
GEMM-like Tensor-Tensor Multiplication (GETT)

- Search space:
  - Blocking parameters: $mc, nc, kc$
  - Subtensors $\hat{A}, \hat{B}, \hat{C}$

- Advantages:

Same arithmetic intensity as GEMM
No memory overhead

Complex to implement
GEMM-like Tensor-Tensor Multiplication (GETT)

- **Search space:**
  - Blocking parameters: $mc, nc, kc$
  - Subtensors $\hat{A}, \hat{B}, \hat{C}$

- **Advantages:**
  - Same arithmetic intensity as GEMM
GEMM-like Tensor-Tensor Multiplication (GETT)

- **Search space:**
  - Blocking parameters: $mc$, $nc$, $kc$
  - Subtensors $\hat{A}$, $\hat{B}$, $\hat{C}$

- **Advantages:**
  - Same arithmetic intensity as GEMM
  - No memory overhead
GEMM-like Tensor-Tensor Multiplication (GETT)

- **Search space:**
  - Blocking parameters: $mc, nc, kc$
  - Subtensors $\hat{A}, \hat{B}, \hat{C}$

- **Advantages:**
  - Same arithmetic intensity as GEMM
  - No memory overhead

- **Disadvantages:**
GEMM-like Tensor-Tensor Multiplication (GETT)

- **Search space:**
  - Blocking parameters: \( mc, nc, kc \)
  - Subtensors \( \hat{A}, \hat{B}, \hat{C} \)

- **Advantages:**
  - Same arithmetic intensity as GEMM
  - No memory overhead

- **Disadvantages:**
  - Complex to implement
Tensor Contraction Code Generator (TCCG)

- **Input:** Mathematical description of TC
  - e.g., \( C[a,b,i,j] = A[i,k,a] \times B[k,j,b] \);

- **Output:** High-Performance C++ code
**Input:** Mathematical description of TC
- e.g., $C[a,b,i,j] = A[i,k,a] \times B[k,j,b]$;

**Output:** High-Performance C++ code

---

**Figure:** Schematic overview of TCCG.
- Not all TCs can be implemented via LoG
- Not all TCs can be implemented via LoG
- Mixed performance
Performance — Haswell (single core)

LoG vs. TTGT

TTGT: good for compute-bound TCs
TTGT: bad for bandwidth-bound TCs

Paul Springer (AICES)
High-Performance Tensor Contractions
Feb. 24th 2017
**TTGT**: good for compute-bound TCs
TTGT: good for compute-bound TCs
TTGT: bad for bandwidth-bound TCs
Performance — Haswell (single core)

- GETT: excels for bandwidth-bound TCs
GETT: excels for bandwidth-bound TCs
GETT: good for compute-bound TCs
Performance gap increases for bandwidth-bound TCs
Performance — Multi-threaded

- Performance gap increases for bandwidth-bound TCs

(a) 2× Intel Xeon E5-2680 v3
(b) NVIDIA Tesla P100
Performance for equally-sized GEMMs varies greatly for different settings: \(\text{opA}, \text{opB}, \text{interchanged} \). A and B.

Performance Model for TTGT and LoG:

Account for varying GEMM perf.

---

(a) \(2 \times \) Intel Xeon E5-2680 v3  
(b) NVIDIA Tesla P100

---

\[\text{Elmar Peise et al. "On the Performance Prediction of BLAS-based Tensor Contractions"}\]
Performance for equally-sized GEMMs varies greatly
- For different settings: opA, opB, interchanged A and B

---

Performance for equally-sized GEMMs varies greatly

- For different settings: \( \text{opA, opB, interchanged } \mathcal{A} \text{ and } \mathcal{B} \)

- Performance Model for TTGT and LoG:
  - Account for varying GEMM perf

---

Conclusion

- A survey of different approaches to TCs has been presented
- GETT exhibits high performance across a wide range of TCs
- TCCG is available at https://github.com/HPAC/tccg
Conclusion

- A survey of different approaches to TCs has been presented
- GETT exhibits high performance across a wide range of TCs
- TCCG is available at https://github.com/HPAC/tccg

Future Work

- Implement TC library based on GETT
- Parallelize GETT
Conclusion

- A survey of different approaches to TCs has been presented
- GETT exhibits high performance across a wide range of TCs
- TCCG is available at https://github.com/HPAC/tccg

Future Work

- Implement TC library based on GETT
- Parallelize GETT

Thank you for your attention.