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

- A tensors is a multidimensional array:
  - 0-order tensor: scalar α

- A tensors is a multidimensional array:
  - 0-order tensor: scalar α
  - 1-order tensor: vector $\mathcal{A}_{i_1}$

Introduction

- A tensors is a multidimensional array:
  - 0-order tensor: scalar α
  - 1-order tensor: vector $\mathcal{A}_{i_1}$
  - 2-order tensor: matrix $\mathcal{A}_{i_1,i_2}$

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

[1] Paul Springer et al. "Design of a high-performance GEMM-like Tensor-Tensor Multiplication"

- A tensors is a multidimensional array:
  - 0-order tensor: scalar α
  - 1-order tensor: vector $\mathcal{A}_{i_1}$
  - 2-order tensor: matrix $\mathcal{A}_{i_1,i_2}$
  - *n*-order tensor: $\mathcal{A}_{i_1,i_2,\dots,i_n}$
- Tensor contractions can be thought of as generalized GEMMs

[1] Paul Springer et al. "Design of a high-performance GEMM-like Tensor-Tensor Multiplication"

Introduction

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

[1] Paul Springer et al. "Design of a high-performance GEMM-like Tensor-Tensor Multiplication"

- A tensors is a multidimensional array:
  - 0-order tensor: scalar α
  - 1-order tensor: vector $\mathcal{A}_{i_1}$
  - 2-order tensor: matrix $\mathcal{A}_{i_1,i_2}$
  - n-order tensor: $\mathcal{A}_{i_1,i_2,\ldots,i_n}$
- 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[1]
  - Akin to a high-performance GEMM implementation

[1] Paul Springer et al. "Design of a high-performance GEMM-like Tensor-Tensor Multiplication"

Outline

- Approaches to Tensor Contractions:
  - Loops over GEMM (LoG)
  - Transpose-Transpose-GEMM-Transpose (TTGT)
  - GEMM-like Tensor-Tensor Multiply (GETT)
- Tensor Contraction Code Generator[2]
- Performance Evaluation

[2] Source code available at: `https://github.com/HPAC/tccg`

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \sum_{k_1} \mathcal{A}_{m_1,k_1} \mathcal{B}_{k_1,n_1}$

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$

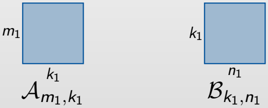
Loop over GEMM (LoG)

RWTH AACHEN UNIVERSITY

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$




```
gemm(M₁, N₁, K₁, A[:,:], B[:,:], C[:,:])
```

AICES

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

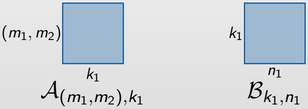
- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,m_2,n_1} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_1}$

Loop over GEMM (LoG)

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$



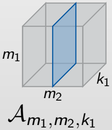
gemm($M_1 \times M_2$, N_1, K_1, $\mathcal{A}[:,:]$, $\mathcal{B}[:,:]$, $\mathcal{C}[:,:]$)

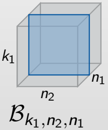
RWTH AACHEN UNIVERSITY

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$





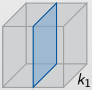

```
for( m2 = 0; m2 < M2; m2++ )
  for( n1 = 0; n1 < N1; n1++ )
    gemm(M1, N2, K1, A[:,m2,:], B[:,:,n1], C[:,n1,:,m2])
```

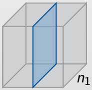
AICES

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$





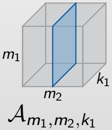

```
for( m2 = 0; m2 < M2; m2++ )
  for( n2 = 0; n2 < N2; n2++ )
    gemm(M1, N1, K1, A[:, m2, :], B[:, n2, :], C[:, :, n2, m2])
```

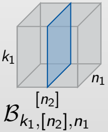
RWTH AACHEN UNIVERSITY

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$





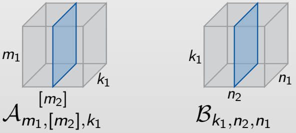

```
for( m₂ = 0; m₂ < M₂; m₂++ )
  gemm_batch(M₁, N₁, K₁, A[:, m₂, :], B[:, n₂, :], C[:, :, n₂, m₂], N₂)
```

AI ces

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$

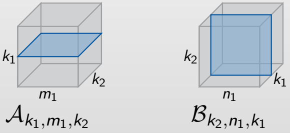



```
for( n2 = 0; n2 < N2; n2++ )
  gemm_batch(M1, N1, K1, A[:, m2, :], B[:, n2, :], C[:, :, n2, m2], M2)
```

Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$
- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$

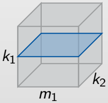



```
for( k1 = 0; k1 < K1; k1++ )
  gemm(M1, N1, K2, A[k1,:,:], B[:,:,k1], C[:,:])
```

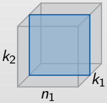
Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$
- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$



$\mathcal{A}_{k_1,m_1,k_2}$



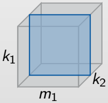
$\mathcal{B}_{k_2,n_1,k_1}$

```
for( k1 = 0; k1 < K1; k1++ )
   gemm(M1, N1, K2, A[k1,:,:], B[:,:,k1], C[:,:])
```

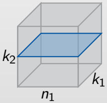
Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$
- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$



$\mathcal{A}_{k_1,m_1,k_2}$



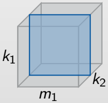
$\mathcal{B}_{k_2,n_1,k_1}$

```
for( k2 = 0; k2 < K2; k2++ )
  gemm(M1, N1, K1, A[:,:,k2]^T, B[k2,:,:]^T, C[:,:])
```

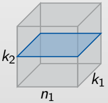
Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$
- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$



$\mathcal{A}_{k_1,m_1,k_2}$



$\mathcal{B}_{k_2,n_1,k_1}$

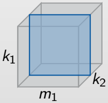
```
for( k2 = 0; k2 < K2; k2++ )
   gemm(M1, N1, K1, A[:,:,k2]^T, B[k2,:,:]^T, C[:,:])
```

Loop over GEMM (LoG)

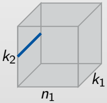
Conceptual Idea

Identify 2D subtensors and contract them via GEMM

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{m_1,k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{(m_1,m_2),n_1} \leftarrow \mathcal{A}_{(m_1,m_2),k_1}\mathcal{B}_{k_1,n_1}$
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$
- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$



$\mathcal{A}_{k_1,m_1,k_2}$



$\mathcal{B}_{k_2,n_1,k_1}$

```
for( n = 0; n1 < N1; n1++ )
  for( k2 = 0; k2 < K2; k2++ )
    gemv(M1, K1, A[:,:,k2]^T, B[k2,n1,:], C[:,n])
```

- Search space:

Loop Over GEMM (LoG)

- Search space:
  - GEMM indices: m, n, k

Loop Over GEMM (LoG)

- 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

- 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

- 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

- Free indices of $\mathcal{A}$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_{\mathcal{A}} \cap I_{\mathcal{C}}$

[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 $\mathcal{A}$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_\mathcal{A} \cap I_\mathcal{C}$
- Free indices of $\mathcal{B}$
  - $I_n := \{n_1, n_2, ..., n_\zeta\} = I_\mathcal{B} \cap I_\mathcal{C}$

[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 $\mathcal{A}$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_\mathcal{A} \cap I_\mathcal{C}$
- Free indices of $\mathcal{B}$
  - $I_n := \{n_1, n_2, ..., n_\zeta\} = I_\mathcal{B} \cap I_\mathcal{C}$
- Contracted indices
  - $I_k := \{k_1, k_2, ..., k_\xi\} = I_\mathcal{A} \cap I_\mathcal{B}$

[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 $\mathcal{A}$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_\mathcal{A} \cap I_\mathcal{C}$
- Free indices of $\mathcal{B}$
  - $I_n := \{n_1, n_2, ..., n_\zeta\} = I_\mathcal{B} \cap I_\mathcal{C}$
- Contracted indices
  - $I_k := \{k_1, k_2, ..., k_\xi\} = I_\mathcal{A} \cap I_\mathcal{B}$
- Tensor contractions can be mapped to BLAS routines[3,4]:
  - `GEMM`: $I_m \neq \emptyset$ **and** $I_n \neq \emptyset$ **and** $I_k \neq \emptyset$.

[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 $\mathcal{A}$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_\mathcal{A} \cap I_\mathcal{C}$
- Free indices of $\mathcal{B}$
  - $I_n := \{n_1, n_2, ..., n_\zeta\} = I_\mathcal{B} \cap I_\mathcal{C}$
- Contracted indices
  - $I_k := \{k_1, k_2, ..., k_\xi\} = I_\mathcal{A} \cap I_\mathcal{B}$
- Tensor contractions can be mapped to BLAS routines[3,4]:
  - `GEMM`: $I_m \neq \emptyset$ **and** $I_n \neq \emptyset$ **and** $I_k \neq \emptyset$.
  - `GEMV`: $(I_m = \emptyset$ **or** $I_n = \emptyset)$ **and** $I_k \neq \emptyset$

[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 $\mathcal{A}$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_\mathcal{A} \cap I_\mathcal{C}$
- Free indices of $\mathcal{B}$
  - $I_n := \{n_1, n_2, ..., n_\zeta\} = I_\mathcal{B} \cap I_\mathcal{C}$
- Contracted indices
  - $I_k := \{k_1, k_2, ..., k_\xi\} = I_\mathcal{A} \cap I_\mathcal{B}$
- Tensor contractions can be mapped to BLAS routines[3,4]:
  - `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** $I_k = \emptyset$

[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 $\mathcal{A}$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_\mathcal{A} \cap I_\mathcal{C}$
- Free indices of $\mathcal{B}$
  - $I_n := \{n_1, n_2, ..., n_\zeta\} = I_\mathcal{B} \cap I_\mathcal{C}$
- Contracted indices
  - $I_k := \{k_1, k_2, ..., k_\xi\} = I_\mathcal{A} \cap I_\mathcal{B}$
- Tensor contractions can be mapped to BLAS routines[3,4]:
  - `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** $I_k = \emptyset$

[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 $\mathcal{A}$
  - $I_m := \{m_1, m_2, ..., m_\gamma\} = I_\mathcal{A} \cap I_\mathcal{C}$
- Free indices of $\mathcal{B}$
  - $I_n := \{n_1, n_2, ..., n_\zeta\} = I_\mathcal{B} \cap I_\mathcal{C}$
- Contracted indices
  - $I_k := \{k_1, k_2, ..., k_\xi\} = I_\mathcal{A} \cap I_\mathcal{B}$
- Tensor contractions can be mapped to BLAS routines[3,4]:
  - `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.

[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

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$

Conceptual Idea

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

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$

$\widetilde{\mathcal{A}}_{m_1,(k_1,k_2)} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$
$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$
$\mathtt{gemm}(M_1,\ N_1,\ K_1 \times K_2,\ \widetilde{\mathcal{A}},\ \widetilde{\mathcal{B}},\ \mathcal{C})$

Conceptual Idea

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

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$

$\widetilde{\mathcal{A}}_{m_1,(k_1,k_2)} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$ $\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$ gemm(M_1, N_1, $K_1 \times K_2$, $\widetilde{\mathcal{A}}$, $\widetilde{\mathcal{B}}$, $\mathcal{C}$)	$\widetilde{\mathcal{A}}_{(k_1,k_2),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$ $\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$ gemm(M_1, N_1, $K_1 \times K_2$, $\widetilde{\mathcal{A}}^T$, $\widetilde{\mathcal{B}}$, $\mathcal{C}$)

RWTH AACHEN UNIVERSITY

Conceptual Idea

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

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$

$\widetilde{\mathcal{A}}_{m_1,(k_1,k_2)} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$
$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$
$\texttt{gemm}(M_1,\ N_1,\ K_1 \times K_2,\ \widetilde{\mathcal{A}},\ \widetilde{\mathcal{B}},\ \mathcal{C})$

$\widetilde{\mathcal{A}}_{(k_1,k_2),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$
$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$
$\texttt{gemm}(M_1,\ N_1,\ K_1 \times K_2,\ \widetilde{\mathcal{A}}^T,\ \widetilde{\mathcal{B}},\ \mathcal{C})$

$\widetilde{\mathcal{A}}_{(k_1,k_2),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$
$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$
$\texttt{gemm}(M_1,\ N_1,\ K_1 \times K_2,\ \widetilde{\mathcal{B}}^T,\ \widetilde{\mathcal{A}},\ \widetilde{\mathcal{C}})$
$\mathcal{C}_{m_1,n_1} \leftarrow \widetilde{\mathcal{C}}_{n_1,m_1}$

AICES

Conceptual Idea

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

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2} \mathcal{B}_{k_2,n_1,k_1}$

$\widetilde{\mathcal{A}}_{m_1,(k_1,k_2)} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$
$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$
$\mathtt{gemm}(M_1, N_1, K_1 \times K_2, \widetilde{\mathcal{A}}, \widetilde{\mathcal{B}}, \mathcal{C})$

$\widetilde{\mathcal{A}}_{(k_1,k_2),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$
$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$
$\mathtt{gemm}(M_1, N_1, K_1 \times K_2, \widetilde{\mathcal{A}}^T, \widetilde{\mathcal{B}}, \mathcal{C})$

$\widetilde{\mathcal{A}}_{(k_1,k_2),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$
$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$
$\mathtt{gemm}(M_1, N_1, K_1 \times K_2, \widetilde{\mathcal{B}}^T, \widetilde{\mathcal{A}}, \widetilde{\mathcal{C}})$
$\mathcal{C}_{m_1,n_1} \leftarrow \widetilde{\mathcal{C}}_{n_1,m_1}$

$\widetilde{\mathcal{A}}_{(k_2,k_1),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$
$\widetilde{\mathcal{B}}_{(k_2,k_1),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$
$\mathtt{gemm}(M_1, N_1, K_1 \times K_2, \widetilde{\mathcal{B}}^T, \widetilde{\mathcal{A}}, \widetilde{\mathcal{C}})$
$\mathcal{C}_{m_1,n_1} \leftarrow \widetilde{\mathcal{C}}_{n_1,m_1}$

Conceptual Idea

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

- $\mathcal{C}_{m_1,n_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}\mathcal{B}_{k_2,n_1,k_1}$

$$\widetilde{\mathcal{A}}_{m_1,(k_1,k_2)} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$$
$$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$$
$$\texttt{gemm}(M_1,\ N_1,\ K_1 \times K_2,\ \widetilde{\mathcal{A}},\ \widetilde{\mathcal{B}},\ \mathcal{C})$$

$$\widetilde{\mathcal{A}}_{(k_1,k_2),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$$
$$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$$
$$\texttt{gemm}(M_1,\ N_1,\ K_1 \times K_2,\ \widetilde{\mathcal{A}}^T,\ \widetilde{\mathcal{B}},\ \mathcal{C})$$

$$\widetilde{\mathcal{A}}_{(k_1,k_2),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$$
$$\widetilde{\mathcal{B}}_{(k_1,k_2),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$$
$$\texttt{gemm}(M_1,\ N_1,\ K_1 \times K_2,\ \widetilde{\mathcal{B}}^T,\ \widetilde{\mathcal{A}},\ \widetilde{\mathcal{C}})$$
$$\mathcal{C}_{m_1,n_1} \leftarrow \widetilde{\mathcal{C}}_{n_1,m_1}$$

$$\widetilde{\mathcal{A}}_{(k_2,k_1),m_1} \leftarrow \mathcal{A}_{k_1,m_1,k_2}$$
$$\widetilde{\mathcal{B}}_{(k_2,k_1),n_1} \leftarrow \mathcal{B}_{k_2,n_1,k_1}$$
$$\texttt{gemm}(M_1,\ N_1,\ K_1 \times K_2,\ \widetilde{\mathcal{B}}^T,\ \widetilde{\mathcal{A}},\ \widetilde{\mathcal{C}})$$
$$\mathcal{C}_{m_1,n_1} \leftarrow \widetilde{\mathcal{C}}_{n_1,m_1}$$

... and more.

- Search space:

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_k

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_k
  - Transposed $\mathcal{A}$

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_k
  - Transposed $\mathcal{A}$
  - Transposed $\mathcal{B}$

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_k
  - Transposed $\mathcal{A}$
  - Transposed $\mathcal{B}$
  - Interchange $\mathcal{A}$ and $\mathcal{B}$ within `GEMM`

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_k
  - Transposed $\mathcal{A}$
  - Transposed $\mathcal{B}$
  - Interchange $\mathcal{A}$ and $\mathcal{B}$ within GEMM
- Advantages:

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_k
  - Transposed $\mathcal{A}$
  - Transposed $\mathcal{B}$
  - Interchange $\mathcal{A}$ and $\mathcal{B}$ within `GEMM`
- Advantages:
  - Easy to implement

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_k
  - Transposed $\mathcal{A}$
  - Transposed $\mathcal{B}$
  - Interchange $\mathcal{A}$ and $\mathcal{B}$ within `GEMM`
- Advantages:
  - Easy to implement
  - Exploits existing BLAS libraries

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_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

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_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?

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_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:

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
    - Any permutation of I_m, I_n, I_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

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

- Search space:
  - Any permutation of I_m, I_n, I_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
  - Additional memory required

[5] Paul Springer et al. "TTC: A high-performance Compiler for Tensor Transpositions"

Key Idea

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

[5] Sharan Chetlur et al. "CUDNN: Efficient Primitives for Deep Learning"

Key Idea

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

```
// N-Loop
for $n$ = 1 : $nc$ : $S_{I_n}$
   // K-Loop (contracted)
   for $k$ = 1 : $kc$ : $S_{I_k}$
      $\widehat{\mathcal{B}}$ = identify_subtensor($B, n, k$)
      // pack $\widehat{\mathcal{B}}$ into $\widetilde{\mathcal{B}}$ (L3 cache)
      $\widetilde{\mathcal{B}}$ = packB($\widehat{\mathcal{B}}$)
      // M-Loop
      for $m$ = 1 : $mc$ : $S_{I_m}$
         $\widehat{\mathcal{A}}$ = identify_subtensor($A, m, k$)
         // pack $\widehat{\mathcal{A}}$ into $\widetilde{\mathcal{A}}$ (L2 cache)
         $\widetilde{\mathcal{A}}$ = packA($\widehat{\mathcal{A}}$)
         $\widehat{\mathcal{C}}$ = identify_subtensor($C, m, n$)
         // compute matrix-matrix product of $\widetilde{\mathcal{A}}\widetilde{\mathcal{B}}$
         macroKernel($\widetilde{\mathcal{A}}, \widetilde{\mathcal{B}}, \widehat{\mathcal{C}}, \alpha, \beta$)
```

[5] Sharan Chetlur et al. "CUDNN: Efficient Primitives for Deep Learning"

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

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

Monday, February 27 10:15 - 10:35

TCCG: Tensor Contraction Code Generator

```
        for m = 1 : mc : S_{I_m}
            Â = identify_subtensor(A, m, k)
            // pack Â into Ã (L2 cache)
            Ã = packA(Â)
            Ĉ = identify_subtensor(C, m, n)
            // compute matrix-matrix product of ÃB̃
            macroKernel(Ã, B̃, Ĉ, α, β)
```

[5] Sharan Chetlur et al. "CUDNN: Efficient Primitives for Deep Learning"

- Search space:

- Search space:
  - Blocking parameters: mc, nc, kc

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

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

- Search space:
  - Blocking parameters: mc, nc, kc
  - Subtensors $\widehat{\mathcal{A}}$, $\widehat{\mathcal{B}}$, $\widehat{\mathcal{C}}$
- Advantages:
  - Same arithmetic intensity as GEMM

GEMM-like Tensor-Tensor Multiplication (GETT)

- Search space:
  - Blocking parameters: mc, nc, kc
  - Subtensors $\widehat{\mathcal{A}}$, $\widehat{\mathcal{B}}$, $\widehat{\mathcal{C}}$
- Advantages:
  - Same arithmetic intensity as GEMM
  - No memory overhead

- Search space:
  - Blocking parameters: mc, nc, kc
  - Subtensors $\widehat{\mathcal{A}}$, $\widehat{\mathcal{B}}$, $\widehat{\mathcal{C}}$
- Advantages:
  - Same arithmetic intensity as GEMM
  - No memory overhead
- Disadvantages:

- Search space:
  - Blocking parameters: mc, nc, kc
  - Subtensors $\widehat{\mathcal{A}}$, $\widehat{\mathcal{B}}$, $\widehat{\mathcal{C}}$
- Advantages:
  - Same arithmetic intensity as GEMM
  - No memory overhead
- Disadvantages:
  - Complex to implement

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

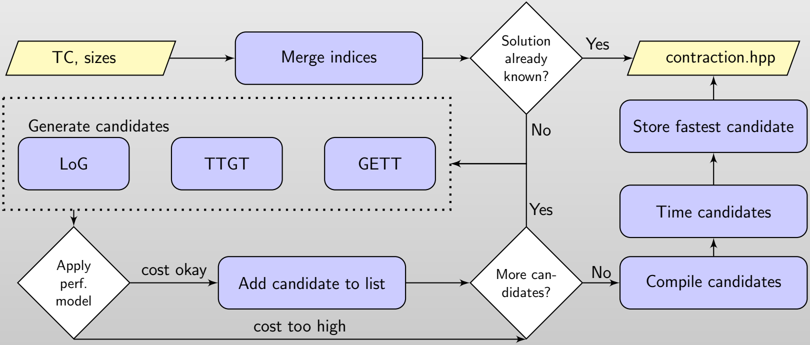
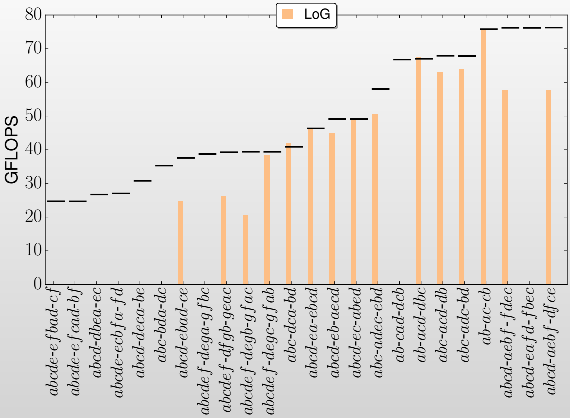
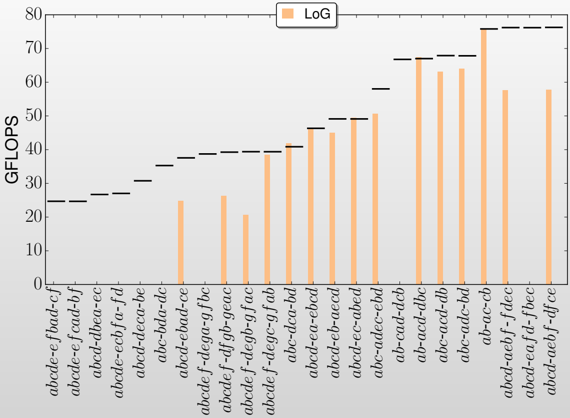


Figure: Schematic overview of TCCG.

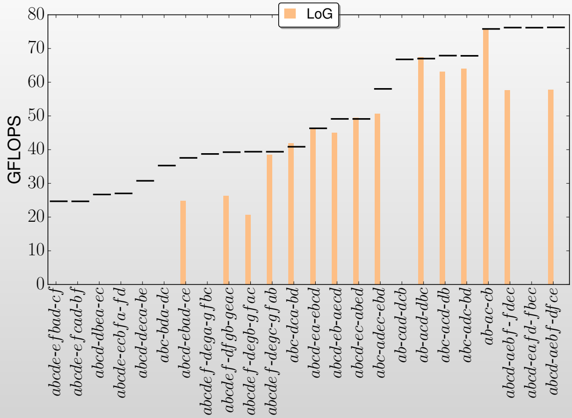
Performance — Haswell (single core)



Performance — Haswell (single core)

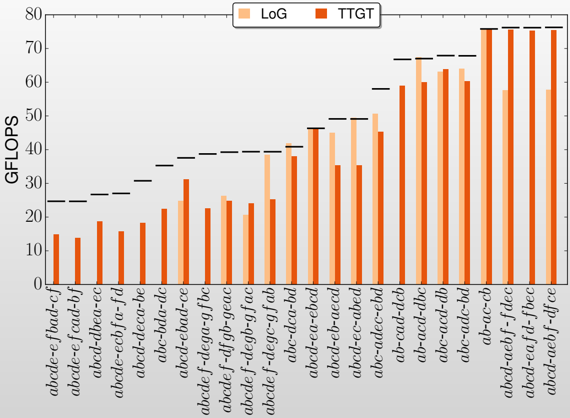


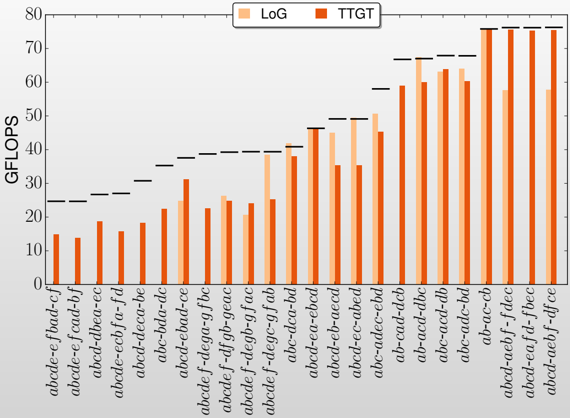
- Not all TCs can be implemented via LoG



- Not all TCs can be implemented via LoG
- Mixed performance

Performance — Haswell (single core)

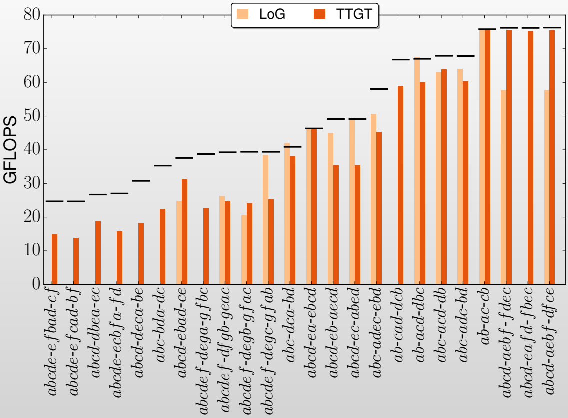




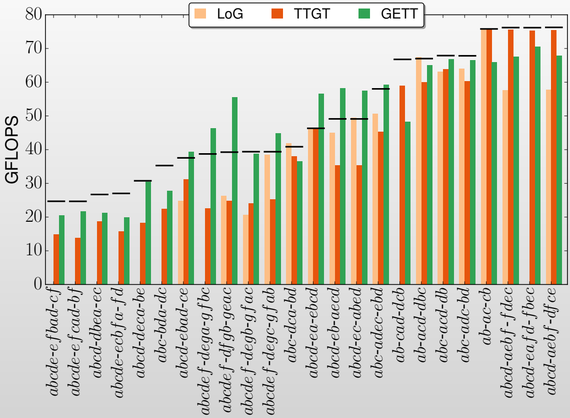
- TTGT: good for compute-bound TCs

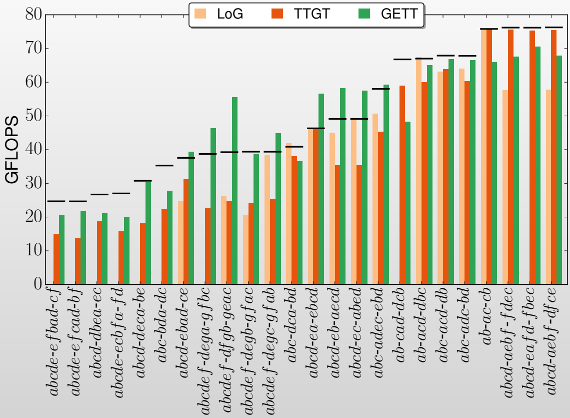
Performance — Haswell (single core)

RWTH AACHEN UNIVERSITY



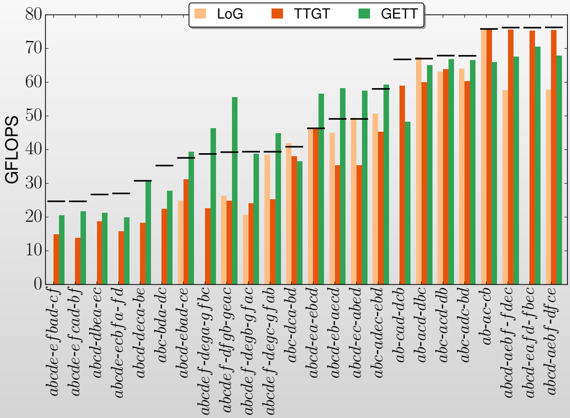
- TTGT: good for compute-bound TCs
- TTGT: bad for bandwidth-bound TCs


LoG
TTGT
GETT
GFLOPS
0
10
20
30
40
50
60
70
80
abcde-efbad-cf
abcde-efcad-bf
abcd-dbea-ec
abcde-ecbfa-fd
abcd-deca-be
abc-bda-dc
abcd-ebad-ce
abcdef-dega-gfbc
abcdef-dfgb-geac
abcdef-degb-gfac
abcdef-degc-gfab
abc-dca-bd
abcd-ea-ebcd
abcd-eb-aecd
abcd-ec-abed
abc-adec-ebd
ab-cad-dcb
ab-acd-dbc
abc-acd-db
abc-adc-bd
ab-ac-cb
abcd-aebf-fdec
abcd-eafd-fbec
abcd-aebf-dfce



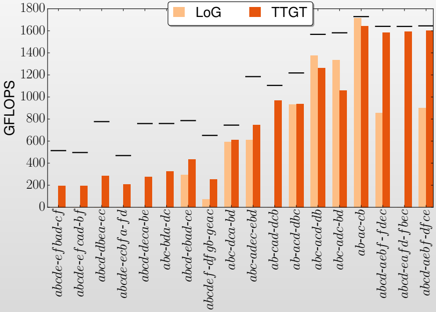
- GETT: excels for bandwidth-bound TCs

Performance — Haswell (single core)



- GETT: excels for bandwidth-bound TCs
- GETT: good for compute-bound TCs

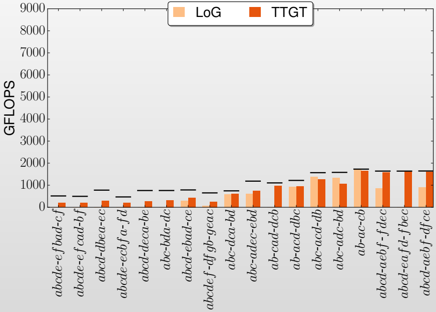
Performance — Multi-threaded



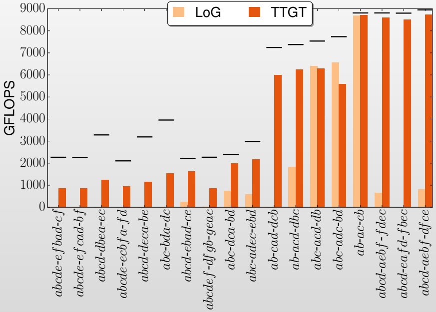
(a) 2×Intel Xeon E5-2680 v3

- Performance gap increases for bandwidth-bound TCs

Performance — Multi-threaded



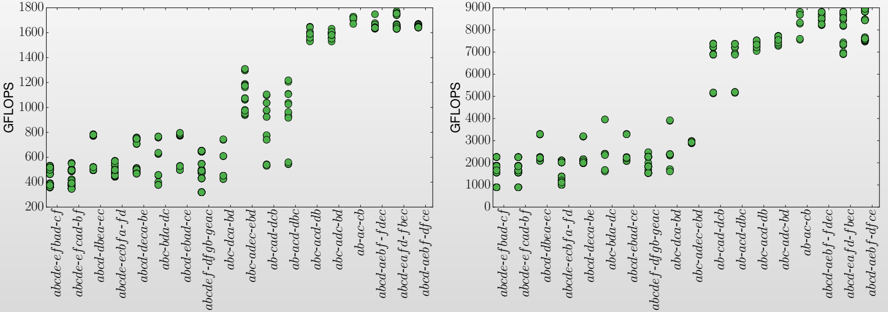
(a) 2×Intel Xeon E5-2680 v3



(b) NVIDIA Tesla P100

- Performance gap increases for bandwidth-bound TCs

GEMM Performance — Multi-threaded

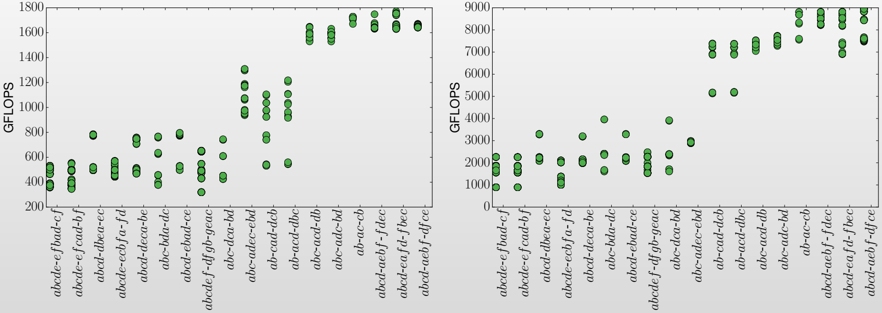


(a) 2×Intel Xeon E5-2680 v3

(b) NVIDIA Tesla P100

[6] Elmar Peise et al. "On the Performance Prediction of BLAS-based Tensor Contractions"

GEMM Performance — Multi-threaded

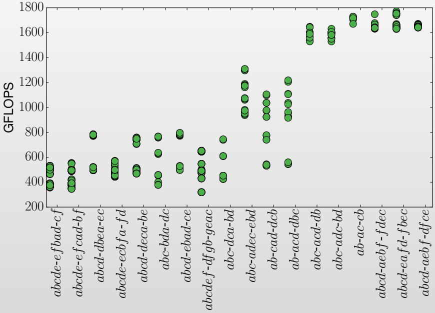


(a) 2×Intel Xeon E5-2680 v3

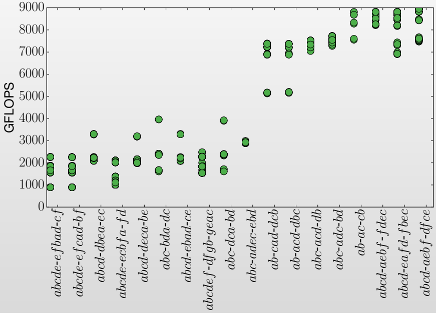
(b) NVIDIA Tesla P100

- Performance for equally-sized GEMMs varies greatly
  - For different settings: `opA`, `opB`, interchanged $\mathcal{A}$ and $\mathcal{B}$

[6] Elmar Peise et al. "On the Performance Prediction of BLAS-based Tensor Contractions"



(a) 2×Intel Xeon E5-2680 v3



(b) NVIDIA Tesla P100

- Performance for equally-sized GEMMs varies greatly
  - For different settings: `opA`, `opB`, interchanged $\mathcal{A}$ and $\mathcal{B}$
- Performance Model for TTGT and LoG:
  - Account for varying GEMM perf[6]

[6] Elmar Peise et al. “On the Performance Prediction of BLAS-based Tensor Contractions”

Conclusion and Future Work

RWTH AACHEN UNIVERSITY

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 and Future Work

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.