A set of building blocks for tensor operations: transposition, summation, and contraction

Paul Springer, Paolo Bientinesi
Markus Höhnerbach

SIAM PP 2018

Aachen Institute for Advanced Study in Computational Engineering Science
High Performance and Automatic Computing
RWTH AACHEN UNIVERSITY

1979: BLAS Level 1

Vector-Vector operations (e.g., `AXPY`: $\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$)

History

1979: BLAS Level 1

Vector-Vector operations (e.g., `AXPY`: $\mathbf{y} \leftarrow \alpha \mathbf{x} + \mathbf{y}$)

1988: BLAS Level 2

Matrix-Vector operations (e.g., `GEMV`: $\mathbf{y} \leftarrow \alpha A \mathbf{x} + \beta \mathbf{y}$)

1979: BLAS Level 1

Vector-Vector operations (e.g., `AXPY`: $\mathbf{y} \leftarrow \alpha \mathbf{x} + \mathbf{y}$)

1988: BLAS Level 2

Matrix-Vector operations (e.g., `GEMV`: $\mathbf{y} \leftarrow \alpha A \mathbf{x} + \beta \mathbf{y}$)

1990: BLAS Level 3

Matrix-Matrix operations (e.g., `GEMM`: $C \leftarrow \alpha AB + \beta C$)

History

1979: BLAS Level 1

Vector-Vector operations (e.g., `AXPY`: $\mathbf{y} \leftarrow \alpha \mathbf{x} + \mathbf{y}$)

1988: BLAS Level 2

Matrix-Vector operations (e.g., `GEMV`: $\mathbf{y} \leftarrow \alpha A \mathbf{x} + \beta \mathbf{y}$)

1990: BLAS Level 3

Matrix-Matrix operations (e.g., `GEMM`: $C \leftarrow \alpha AB + \beta C$)

2018: BLAS Level 4 ?

Tensor operations

Goals

- Readily usable open-source software (library and/or compiler)
- Substantial speedups over state-of-the-art algorithms

Goals

- Readily usable open-source software (library and/or compiler)
- Substantial speedups over state-of-the-art algorithms

Challenges for High-Performance

Goals

- Readily usable open-source software (library and/or compiler)
- Substantial speedups over state-of-the-art algorithms

Challenges for High-Performance

- High order of tensors
  - Strided memory accesses
  - Difficult to exploit *spatial*- and *temporal-locality*

Goals

- Readily usable open-source software (library and/or compiler)
- Substantial speedups over state-of-the-art algorithms

Challenges for High-Performance

- High order of tensors
  - Strided memory accesses
  - Difficult to exploit *spatial*- and *temporal-locality*
- Vectorization

Goals

- Readily usable open-source software (library and/or compiler)
- Substantial speedups over state-of-the-art algorithms

Challenges for High-Performance

- High order of tensors
  - Strided memory accesses
  - Difficult to exploit *spatial-* and *temporal-locality*
- Vectorization
- Parallelization

Outline

RWTH AACHEN UNIVERSITY

1. Tensor Transpositions

2. Spin Summations

3. Tensor Contractions

- Transpositions of the general form:

$$\mathcal{B}_{i_1,i_2,\ldots,i_N} \leftarrow \alpha\mathcal{A}_{\pi(i_1,i_2,\ldots,i_N)} + \beta\mathcal{B}_{i_1,i_2,\ldots,i_N}$$

[1] P. Springer, T. Su and P. Bientinesi, "HPTT: A High-Performance Tensor Transposition C++ Library", ARRAY 2017

- Transpositions of the general form:

$$\mathcal{B}_{i_1,i_2,\ldots,i_N} \leftarrow \alpha \mathcal{A}_{\pi(i_1,i_2,\ldots,i_N)} + \beta \mathcal{B}_{i_1,i_2,\ldots,i_N}$$

- HPTT: C++ 11 library for tensor transpositions[1]
  - Explicitly vectorized
  - Multi-threaded
  - Autotuning

[1] P. Springer, T. Su and P. Bientinesi, "HPTT: A High-Performance Tensor Transposition C++ Library", ARRAY 2017

- Transpositions of the general form:

$$\mathcal{B}_{i_1,i_2,\ldots,i_N} \leftarrow \alpha \mathcal{A}_{\pi(i_1,i_2,\ldots,i_N)} + \beta \mathcal{B}_{i_1,i_2,\ldots,i_N}$$

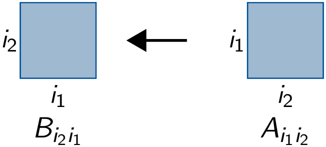
- HPTT: C++ 11 library for tensor transpositions[1]
    - Explicitly vectorized
    - Multi-threaded
    - Autotuning

- Dynamic data structure called `plan` (similar to FFTW)
    - Encodes the execution of a tensor transposition
    - Loop Order
    - Parallelism

[1] P. Springer, T. Su and P. Bientinesi, "HPTT: A High-Performance Tensor Transposition C++ Library", ARRAY 2017

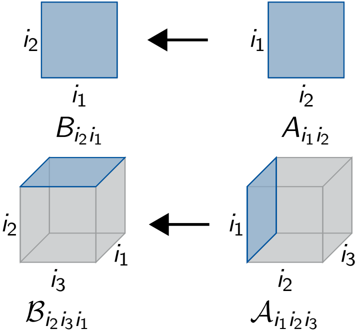
```
for  i1 = 0 : N
   for  i2 = 0 : N
      B[i2, i1] ← A[i1, i2]
```

$$\begin{array}{l} \texttt{for}\ \ i_1 = 0 : N \\ \quad \texttt{for}\ \ i_2 = 0 : N \\ \quad\quad \mathcal{B}[i_2, i_1] \leftarrow \mathcal{A}[i_1, i_2] \end{array}$$



```
for  i1 = 0 : N
  for  i2 = 0 : N
    B[i2, i1] ← A[i1, i2]
```

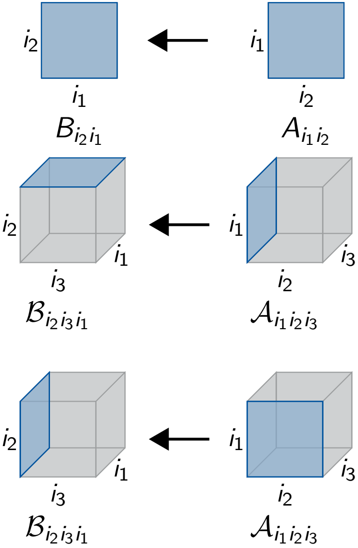
```
for  i2 = 0 : N
  for  i3 = 0 : N
    for  i1 = 0 : N
      B[i2, i3, i1] ← A[i1, i2, i3]
```



for $i_1 = 0 : N$
  for $i_2 = 0 : N$
    $\mathcal{B}[i_2, i_1] \leftarrow \mathcal{A}[i_1, i_2]$

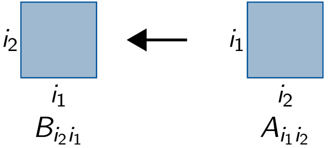
for $i_2 = 0 : N$
  for $i_3 = 0 : N$
    for $i_1 = 0 : N$
      $\mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$

for $i_3 = 0 : N$
  for $i_2 = 0 : N$
    for $i_1 = 0 : N$
      $\mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$



Reduction to 2D

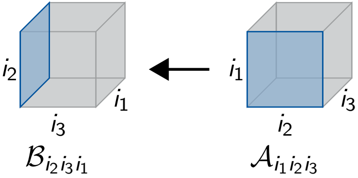
for $i_1 = 0 : N$
  for $i_2 = 0 : N$
    $\mathcal{B}[i_2, i_1] \leftarrow \mathcal{A}[i_1, i_2]$

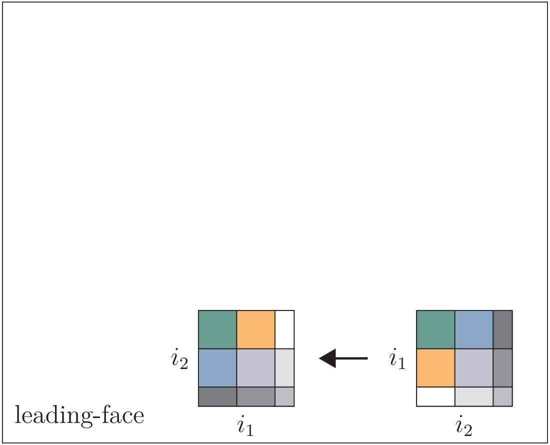


Key Insight

- *Leading-face*: Spanned by the two stride-1 modes
- Reduction to 2D is always possible

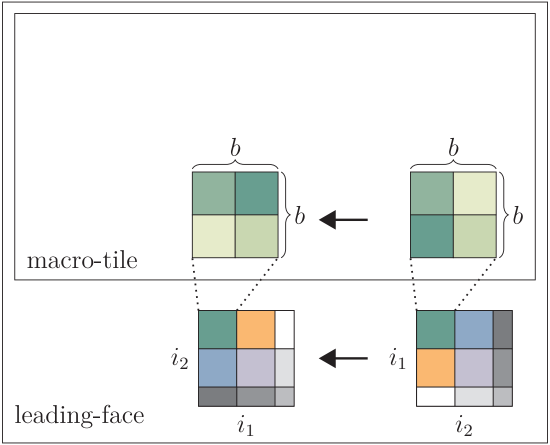
for $i_3 = 0 : N$
  for $i_2 = 0 : N$
    for $i_1 = 0 : N$
      $\mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$

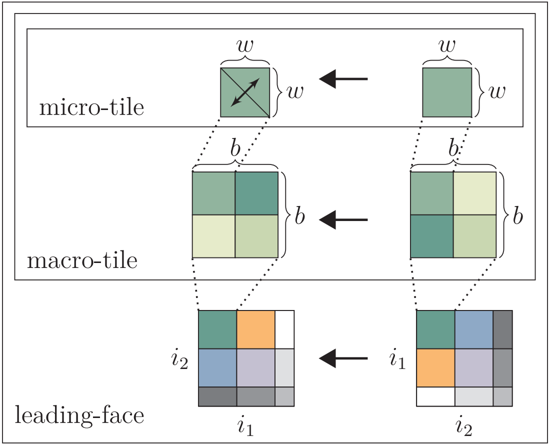


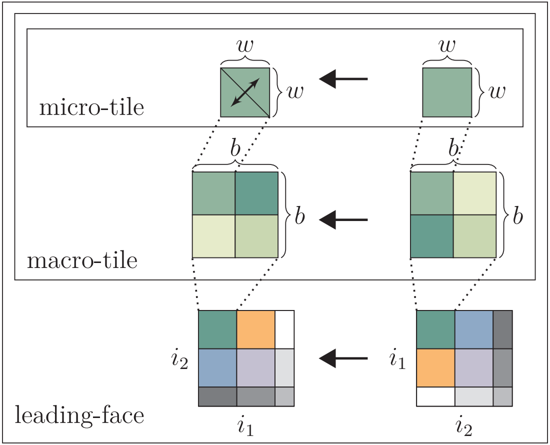


leading-face
i_2
i_1
i_1
i_2

Blocking Overview

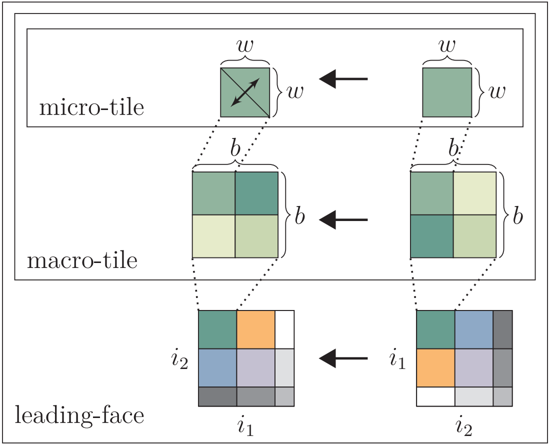



micro-tile
w
w
w
w
macro-tile
b
b
b
b
leading-face
i2
i1
i1
i2



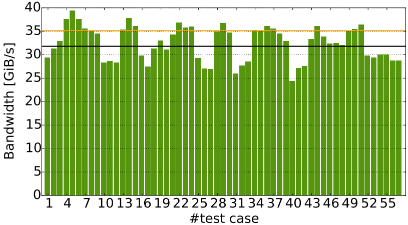
- Decompose a macro-tile into micro-tiles
  - Vectorized micro-tiles

Blocking Overview

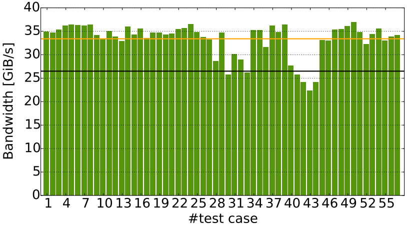


- Decompose a macro-tile into micro-tiles
  - Vectorized micro-tiles
- Decompose a transposition into macro-tiles
  - Parallel over macro-tiles

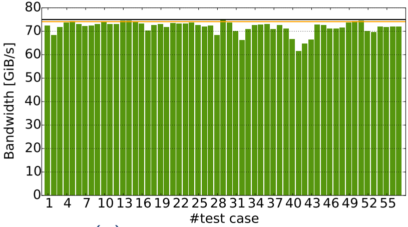
Bandwidth



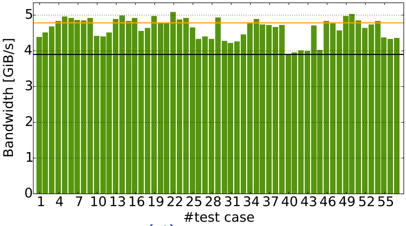
(a) Intel Ivy Bridge E5-2670 v2.



(b) IBM Power7.

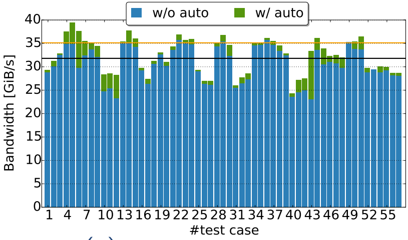


(c) Intel KNL Xeon Phi 7210.

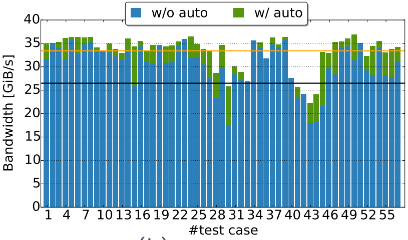


(d) ARMv7-A.

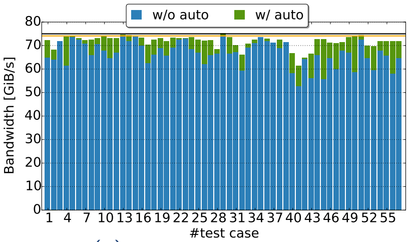
HPTT's bandwidth. Black and orange lines respectively denote STREAM and AXPY bandwidth.



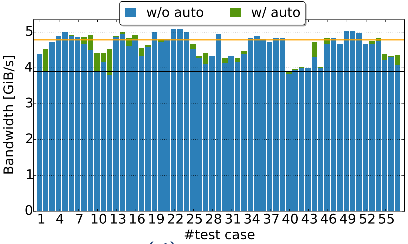
(a) Intel Ivy Bridge E5-2670 v2.



(b) IBM Power7.



(c) Intel KNL Xeon Phi 7210.

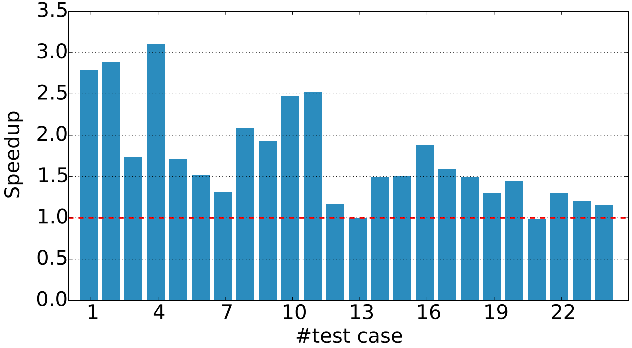


(d) ARMv7-A.

HPTT's bandwidth. Black and orange lines respectively denote STREAM and AXPY bandwidth.

Impact

- Cyclops Tensor Framework
  Tensor Contractions → Transpositions



HPTT's impact on CTF's performance.

Outline

Linear summation over tensor transpositions

- $\mathcal{B}_{i_0 i_1 i_2} = 2\mathcal{A}_{i_0 i_1 i_2} - \mathcal{A}_{i_2 i_1 i_0} - \mathcal{A}_{i_0 i_2 i_1}$
- $\mathcal{B}_{i_0 i_1 i_2} = 4\mathcal{A}_{i_0 i_1 i_2} - 2\mathcal{A}_{i_1 i_0 i_2} - 2\mathcal{A}_{i_2 i_1 i_0} + \mathcal{A}_{i_1 i_2 i_0} - 2\mathcal{A}_{i_0 i_2 i_1} + \mathcal{A}_{i_2 i_0 i_1}$
- $\mathcal{B}_{i_0 i_1 i_2 i_3} = 2\mathcal{A}_{i_0 i_1 i_2 i_3} - \mathcal{A}_{i_2 i_1 i_0 i_3} - \mathcal{A}_{i_0 i_2 i_1 i_3} - \mathcal{A}_{i_0 i_1 i_3 i_2}$
- $\mathcal{B}_{i_0 i_1 i_2 i_3} = 4\mathcal{A}_{i_0 i_1 i_2 i_3} - 2\mathcal{A}_{i_0 i_3 i_2 i_1} - 2\mathcal{A}_{i_0 i_1 i_3 i_2} - 2\mathcal{A}_{i_1 i_0 i_2 i_3} + \mathcal{A}_{i_1 i_3 i_2 i_0} + \mathcal{A}_{i_1 i_0 i_3 i_2} - 2\mathcal{A}_{i_2 i_1 i_0 i_3} + \mathcal{A}_{i_2 i_3 i_0 i_1} + \mathcal{A}_{i_2 i_1 i_3 i_0} - 2\mathcal{A}_{i_3 i_1 i_2 i_0} + \mathcal{A}_{i_3 i_0 i_2 i_1} + \mathcal{A}_{i_3 i_1 i_0 i_2}$

RWTH AACHEN UNIVERSITY

Linear summation over tensor transpositions

- $\mathcal{B}_{i_0 i_1 i_2} = 2\mathcal{A}_{i_0 i_1 i_2} - \mathcal{A}_{i_2 i_1 i_0} - \mathcal{A}_{i_0 i_2 i_1}$
- $\mathcal{B}_{i_0 i_1 i_2} = 4\mathcal{A}_{i_0 i_1 i_2} - 2\mathcal{A}_{i_1 i_0 i_2} - 2\mathcal{A}_{i_2 i_1 i_0} + \mathcal{A}_{i_1 i_2 i_0} - 2\mathcal{A}_{i_0 i_2 i_1} + \mathcal{A}_{i_2 i_0 i_1}$
- $\mathcal{B}_{i_0 i_1 i_2 i_3} = 2\mathcal{A}_{i_0 i_1 i_2 i_3} - \mathcal{A}_{i_2 i_1 i_0 i_3} - \mathcal{A}_{i_0 i_2 i_1 i_3} - \mathcal{A}_{i_0 i_1 i_3 i_2}$
- $\mathcal{B}_{i_0 i_1 i_2 i_3} = 4\mathcal{A}_{i_0 i_1 i_2 i_3} - 2\mathcal{A}_{i_0 i_3 i_2 i_1} - 2\mathcal{A}_{i_0 i_1 i_3 i_2} - 2\mathcal{A}_{i_1 i_0 i_2 i_3} + \mathcal{A}_{i_1 i_3 i_2 i_0} + \mathcal{A}_{i_1 i_0 i_3 i_2} - 2\mathcal{A}_{i_2 i_1 i_0 i_3} + \mathcal{A}_{i_2 i_3 i_0 i_1} + \mathcal{A}_{i_2 i_1 i_3 i_0} - 2\mathcal{A}_{i_3 i_1 i_2 i_0} + \mathcal{A}_{i_3 i_0 i_2 i_1} + \mathcal{A}_{i_3 i_1 i_0 i_2}$

Central Challenge

Spatial and temporal locality in both $\mathcal{A}$ and $\mathcal{B}$

Linear summation over tensor transpositions

- $\mathcal{B}_{i_0 i_1 i_2} = 2\mathcal{A}_{i_0 i_1 i_2} - \mathcal{A}_{i_2 i_1 i_0} - \mathcal{A}_{i_0 i_2 i_1}$
- $\mathcal{B}_{i_0 i_1 i_2} = 4\mathcal{A}_{i_0 i_1 i_2} - 2\mathcal{A}_{i_1 i_0 i_2} - 2\mathcal{A}_{i_2 i_1 i_0} + \mathcal{A}_{i_1 i_2 i_0} - 2\mathcal{A}_{i_0 i_2 i_1} + \mathcal{A}_{i_2 i_0 i_1}$
- $\mathcal{B}_{i_0 i_1 i_2 i_3} = 2\mathcal{A}_{i_0 i_1 i_2 i_3} - \mathcal{A}_{i_2 i_1 i_0 i_3} - \mathcal{A}_{i_0 i_2 i_1 i_3} - \mathcal{A}_{i_0 i_1 i_3 i_2}$
- $\mathcal{B}_{i_0 i_1 i_2 i_3} = 4\mathcal{A}_{i_0 i_1 i_2 i_3} - 2\mathcal{A}_{i_0 i_3 i_2 i_1} - 2\mathcal{A}_{i_0 i_1 i_3 i_2} - 2\mathcal{A}_{i_1 i_0 i_2 i_3} + \mathcal{A}_{i_1 i_3 i_2 i_0} + \mathcal{A}_{i_1 i_0 i_3 i_2} - 2\mathcal{A}_{i_2 i_1 i_0 i_3} + \mathcal{A}_{i_2 i_3 i_0 i_1} + \mathcal{A}_{i_2 i_1 i_3 i_0} - 2\mathcal{A}_{i_3 i_1 i_2 i_0} + \mathcal{A}_{i_3 i_0 i_2 i_1} + \mathcal{A}_{i_3 i_1 i_0 i_2}$

Central Challenge

Spatial and temporal locality in both $\mathcal{A}$ and $\mathcal{B}$

P. Springer, D. Matthews and P. Bientinesi,
"Spin Summations: A High-Performance Perspective", ACM TOMS'18

Outline

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

[2] P. Springer and P. Bientinesi, "Design of a high-performance GEMM-like Tensor-Tensor Multiplication", ACM TOMS'18

[3] Field Van Zee et al. "BLIS: A Framework for Rapidly Instantiating BLAS Functionality", ACM TOMS'15

- Prior Work
  - Nested loops
  - Loops over GEMM (LoG)
  - Transpose-Transpose-GEMM-Transpose (TTGT)
- Disadvantages
  - Suboptimal I/O cost
  - Poor cache utilization
  - Additional memory requirements

[2] P. Springer and P. Bientinesi, "Design of a high-performance GEMM-like Tensor-Tensor Multiplication", ACM TOMS'18

[3] Field Van Zee et al. "BLIS: A Framework for Rapidly Instantiating BLAS Functionality", ACM TOMS'15

- Prior Work
  - Nested loops
  - Loops over GEMM (LoG)
  - Transpose-Transpose-GEMM-Transpose (TTGT)
- Disadvantages
  - Suboptimal I/O cost
  - Poor cache utilization
  - Additional memory requirements
- We propose a fourth approach: GETT[2]
  - Akin to a high-performance GEMM implementation
  - Adopts the BLIS[3] methodology

[2] P. Springer and P. Bientinesi, "Design of a high-performance GEMM-like Tensor-Tensor Multiplication", ACM TOMS'18

[3] Field Van Zee et al. "BLIS: A Framework for Rapidly Instantiating BLAS Functionality", ACM TOMS'15

Matrix-Matrix Multiplication

$C_{m,n} \leftarrow \sum_k A_{m,k} B_{k,n}$

Matrix-Matrix Multiplication (Einstein notation)

$C_{m,n} \leftarrow A_{m,k} B_{k,n}$

Matrix-Matrix Multiplication (Einstein notation)

$$C_{m,n} \leftarrow A_{m,k}B_{k,n}$$

```
// N-Loop
for n = 0 : N−1
    // M-Loop
    for m = 0 : M−1
        tmp = 0
        // K-Loop (contracted)
        for k = 0 : K−1
            tmp += A_{i,k} B_{k,j}
        // update C
        C_{i,j} = α tmp + β C_{i,j}
```

Naive GEMM

RWTH AACHEN UNIVERSITY

Matrix-Matrix Multiplication (Einstein notation)

$$C_{m,n} \leftarrow A_{m,k} B_{k,n}$$

```
// N-Loop
for n = 0 : N − 1
    // M-Loop
    for m = 0 : M − 1
        tmp = 0
        // K-Loop (contracted)
        for k = 0 : K − 1
            tmp += A_{i,k} B_{k,j}
        // update C
        C_{i,j} = α tmp + β C_{i,j}
```

Naive GEMM

```
// N-Loop
for n = 0 : nc : N − 1
    // K-Loop (contracted)
    for k = 0 : kc : K − 1
        B̂ = identify_submatrix(B, n, k)
        // pack B̂ into B̃
        B̃ = packB(B̂) // B̃ ∈ ℝ^{kc×nc}
        // M-Loop
        for m = 0 : mc : M − 1
            Â = identify_submatrix(A, m, k)
            // pack Â into Ã
            Ã = packA(Â) // Ã ∈ ℝ^{mc×kc}
            Ĉ = identify_submatrix(C, m, n)
            // matrix-matrix product: ÃB̃
            macroKernel(Ã, B̃, Ĉ, α, β)
```

High-performance GEMM

Key Idea

Pack-and-transpose while moving data into the caches

GEMM-like Tensor-Tensor Multiplication (GETT)

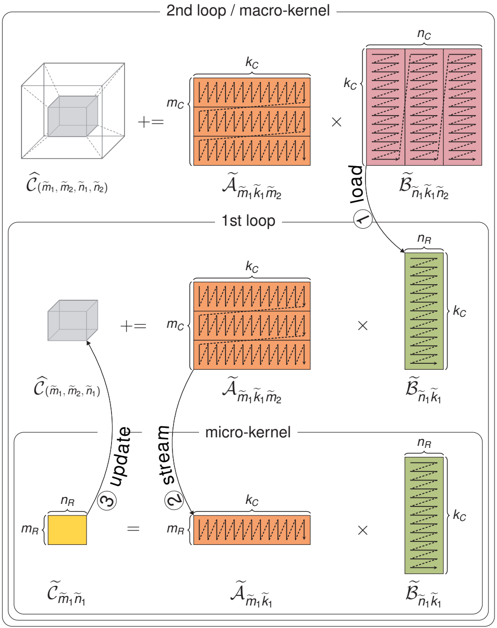
RWTH AACHEN UNIVERSITY

Key Idea

Pack-and-transpose while moving data into the caches

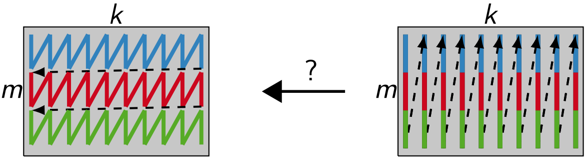
```
// N-Loop
for $n$ = 1 : $nc$ : $S_{I_n}$
   // K-Loop (contracted)
   for $k$ = 1 : $kc$ : $S_{I_k}$
      $\widehat{\mathcal{B}}$ = identify_subtensor($\mathcal{B}, n, k$)
      // pack $\widehat{\mathcal{B}}$ into $\widetilde{\mathcal{B}}$ (L3 cache)
      $\widetilde{\mathcal{B}}$ = packB($\widehat{\mathcal{B}}$)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

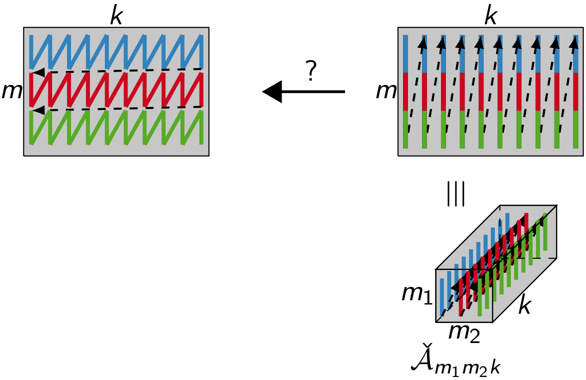
High-performance GETT



- Similar to BLIS
- Blocking
  - L3 cache
  - L2 cache
  - L1 cache
  - Registers
- Vectorized *micro-kernel*

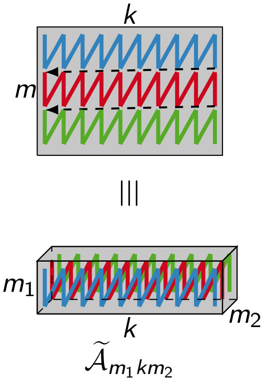
Packing via Tensor Transpositions



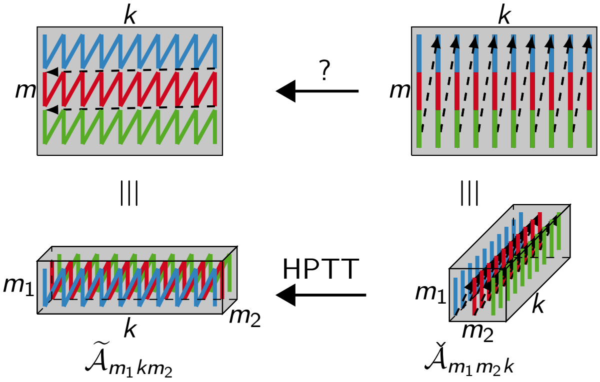


k
m
?
k
m
$|||$
m_1
m_2
k
$\check{\mathcal{A}}_{m_1 m_2 k}$

Packing via Tensor Transpositions



Packing via Tensor Transpositions



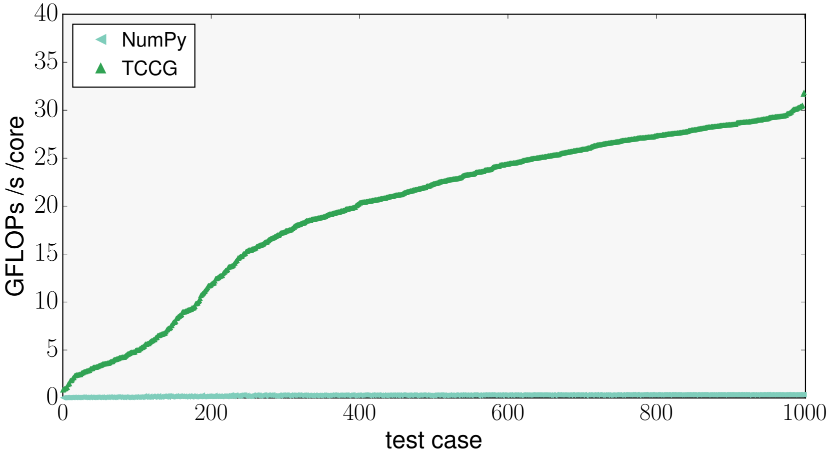
- Multi-dimensional packing routines
  - Preserve stride-1 index
  - Exploit spatial locality ⇒ high efficiency

- Tensor Contraction Code Generator (TCCG)[4]
  - Research Tool
  - Supports: Naive, LoG, TTGT, and **GETT**
  - Autotuning

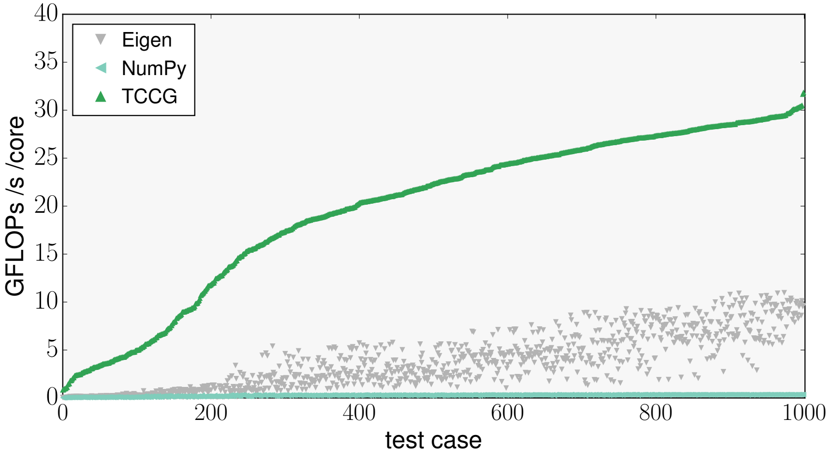
- Tensor Contraction Library (TCL)[5]
  - C++ library
  - Based on TTGT + HPTT
  - Offers C and Python interfaces
  - Supports a large variety of TCs

[4] Available at: `hptts://github.com/hpac/tccg`

[5] Available at: `hptts://github.com/springer13/tcl`

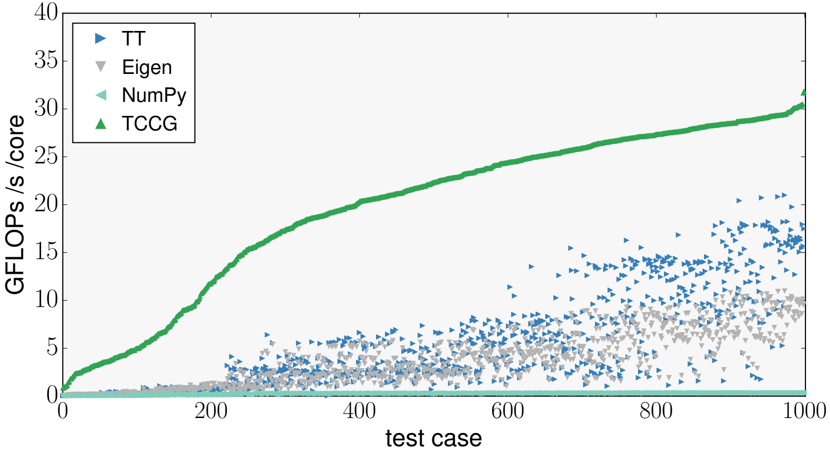


DP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.

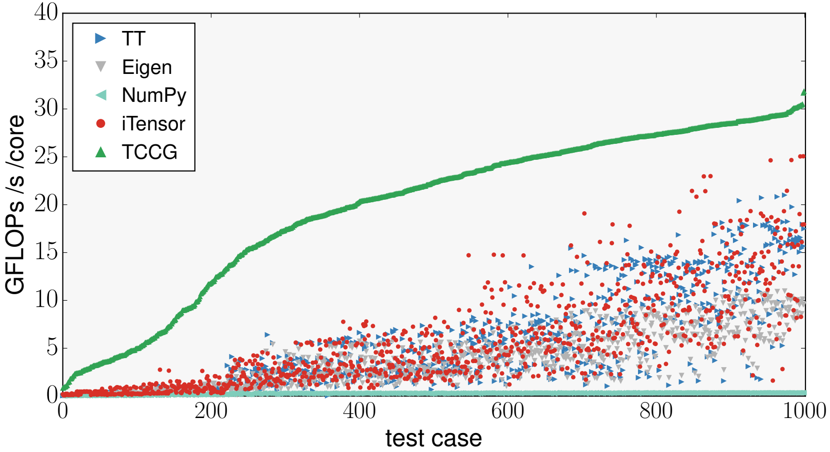


DP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.

RWTH AACHEN UNIVERSITY

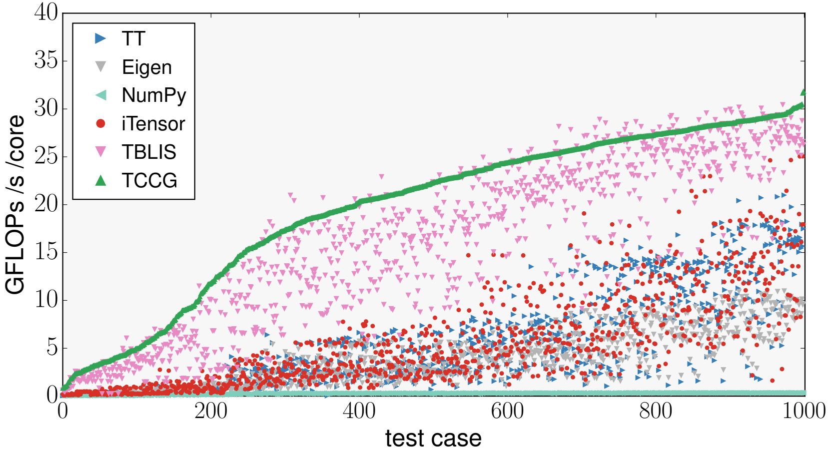


DP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.



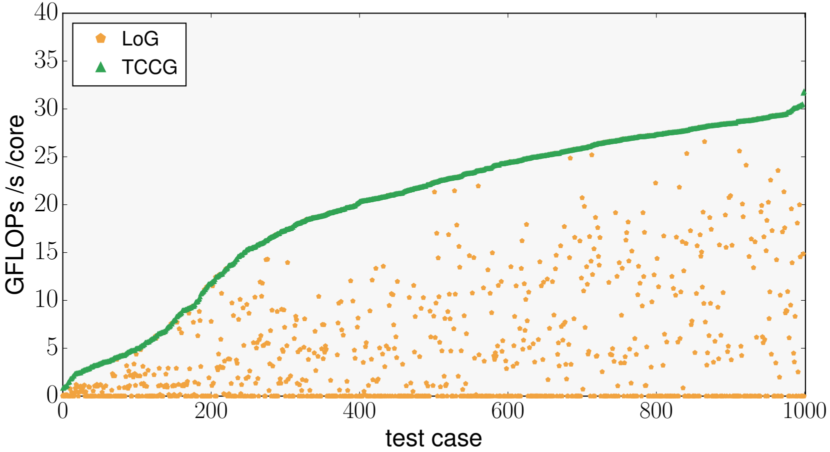
DP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.

Performance



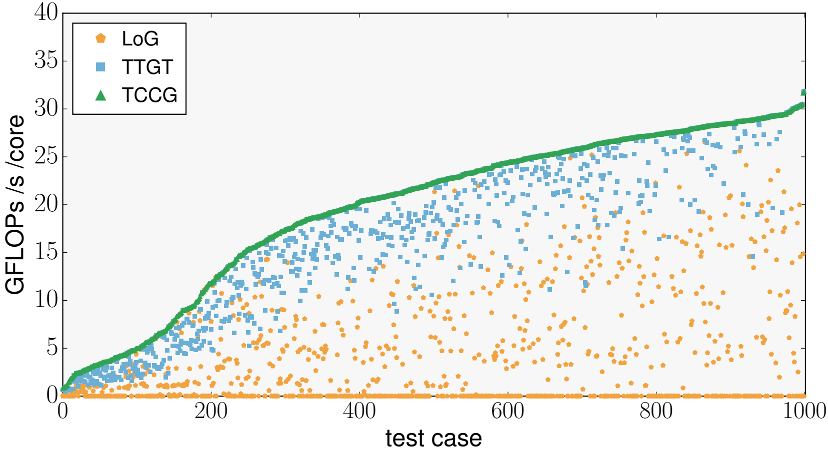
DP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.

RWTH AACHEN UNIVERSITY



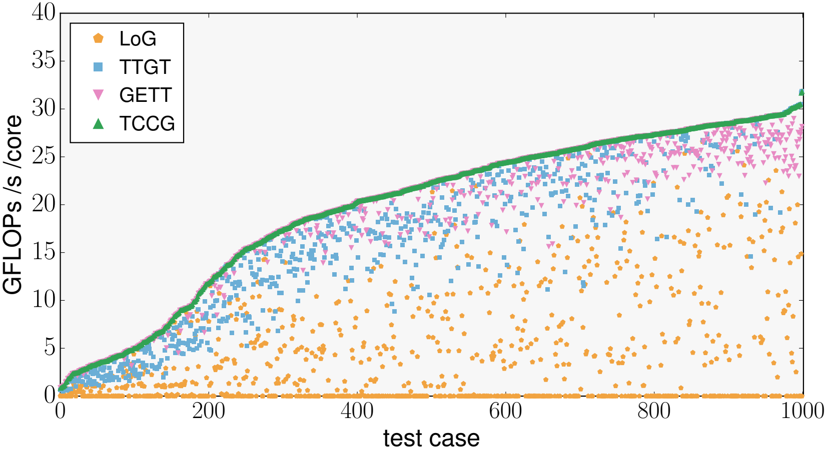
DP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.

RWTH AACHEN UNIVERSITY



DP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.

RWTH AACHEN UNIVERSITY



DP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.

- Noticeable Speedups over state-of-the-art solutions ✓
  - Proper memory accesses are critical for tensor operations
- Released open-source code ✓
  - High-Performance Tensor Transpose C++ Library (HPTT)[6]
  - Code Generator for Spin Summations[7]
  - GEMM-like Tensor-Tensor contraction (GETT)[8]
  - Tensor Contraction C++ Library (TCL)[9]

[6] Available at `hptts://github.com/springer13/hptt`, BSD-3

[7] Available at `hptts://github.com/springer13/spin`, BSD-3

[8] Available at `hptts://github.com/hpac/tccg`, LGPLv3

[9] Available at `hptts://github.com/springer13/tcl`, LGPLv3

- Noticeable Speedups over state-of-the-art solutions ✓
  - Proper memory accesses are critical for tensor operations
- Released open-source code ✓
  - High-Performance Tensor Transpose C++ Library (HPTT)[6]
  - Code Generator for Spin Summations[7]
  - GEMM-like Tensor-Tensor contraction (GETT)[8]
  - Tensor Contraction C++ Library (TCL)[9]

Thank you for your attention

[6] Available at hptts://github.com/springer13/hptt, BSD-3
[7] Available at hptts://github.com/springer13/spin, BSD-3
[8] Available at hptts://github.com/hpac/tccg, LGPLv3
[9] Available at hptts://github.com/springer13/tcl, LGPLv3

Publications

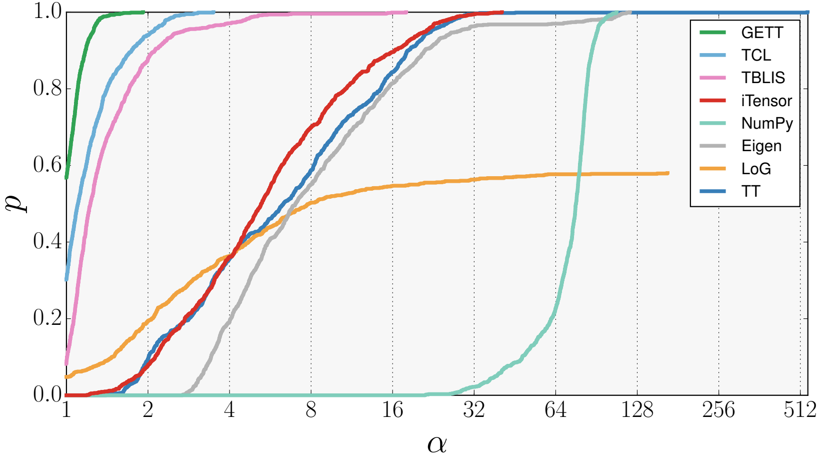
[1] Paul Springer and Paolo Bientinesi. Design of a high-performance gemm-like tensor-tensor multiplication. *ACM Trans. Math. Softw.*, 44(3):28:1–28:29, January 2018.

[2] Paul Springer, Jeff R. Hammond, and Paolo Bientinesi. TTC: A high-performance compiler for tensor transpositions. *ACM Trans. Math. Softw.*, 44(2):15:1–15:21, August 2017.

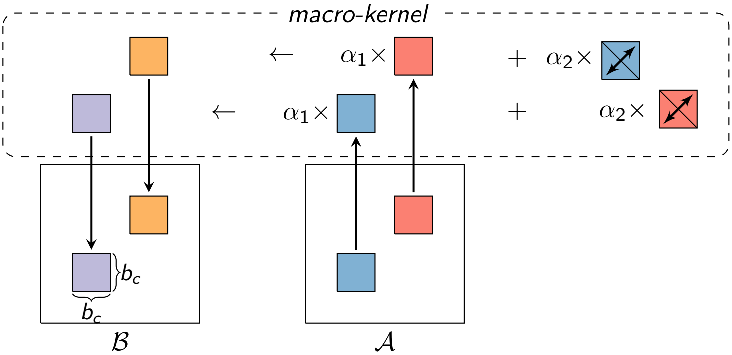
[3] Paul Springer, Devin Matthews, and Paolo Bientinesi. Spin Summations: A High-Performance Perspective. apr 2017. Under review for ACM Transactions on Mathematical Software.

[4] Paul Springer, Aravind Sankaran, and Paolo Bientinesi. Ttc: A tensor transposition compiler for multiple architectures. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY 2016, pages 41–46, New York, NY, USA, 2016. ACM.

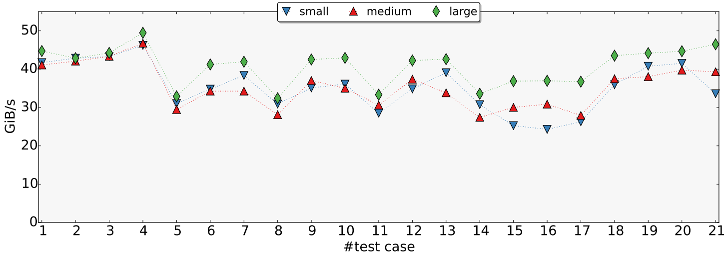
[5] Paul Springer, Tong Su, and Paolo Bientinesi. HPTT: A High-Performance Tensor Transposition C++ Library. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY 2017, pages 56–62, New York, NY, USA, 2017. ACM.



Performance Profile.

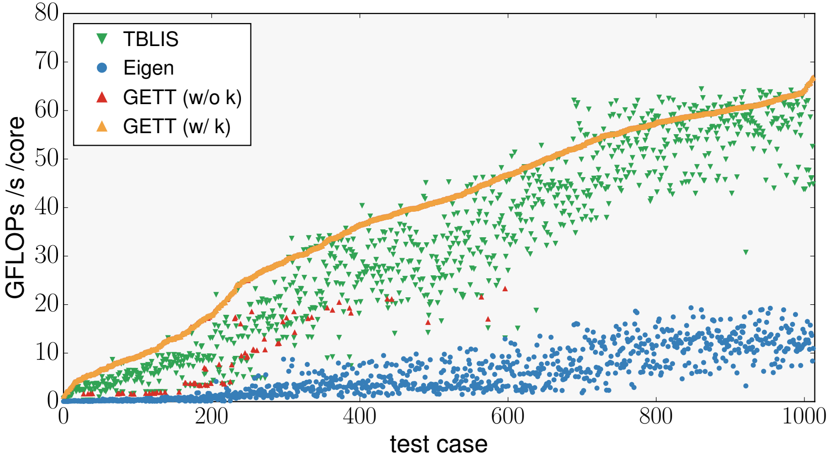


Exemplary blocking for 2-way spin summation: $\mathcal{B}_{i_1 i_2} \leftarrow (\alpha_1 \mathcal{A}_{i_1 i_2} + \alpha_2 \mathcal{A}_{i_2 i_1}$.



Bandwidth; small, medium and large respectively correspond tensors of size 70 MiB, 320 MiB, and 1200 MiB.

RWTH AACHEN UNIVERSITY



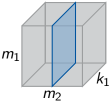
SP tensor contractions. Host: 2× Haswell-EP E5-2680 v3 @ 24 threads.

Key Idea

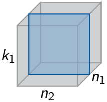
Identify 2D subtensors and contract them via GEMM

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

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



$\mathcal{A}_{m_1,m_2,k_1}$



$\mathcal{B}_{k_1,n_2,n_1}$

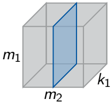
Loop over GEMM (LoG)

Key Idea

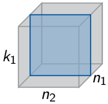
Identify 2D subtensors and contract them via GEMM

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

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

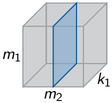


$\mathcal{A}_{m_1,m_2,k_1}$

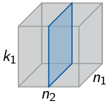


$\mathcal{B}_{k_1,n_2,n_1}$

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



$\mathcal{A}_{m_1,m_2,k_1}$



$\mathcal{B}_{k_1,n_2,n_1}$

Key Idea

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

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

Key Idea

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

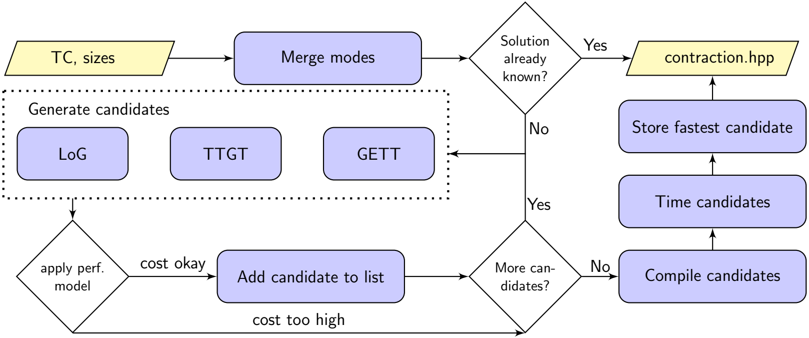
- $\mathcal{C}_{m_1,n_1,n_2,m_2} \leftarrow \mathcal{A}_{m_1,m_2,k_1}\mathcal{B}_{k_1,n_2,n_1}$

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

- **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

[10] Available at: `hptts://github.com/hpac/tccg`

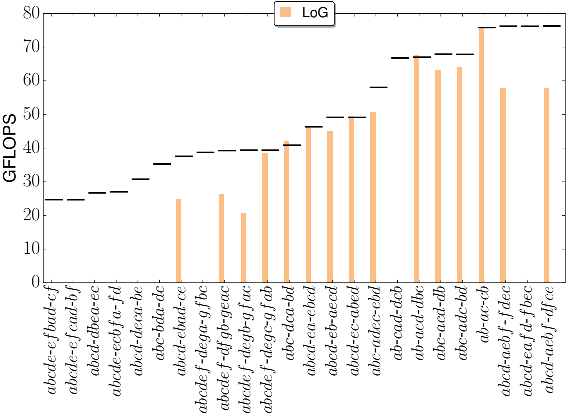
- **Input:** Mathematical description of TC
  - e.g., `C[a,b,i,j] = A[i,k,a] * B[k,j,b];`
- **Output:** High-Performance C++ code



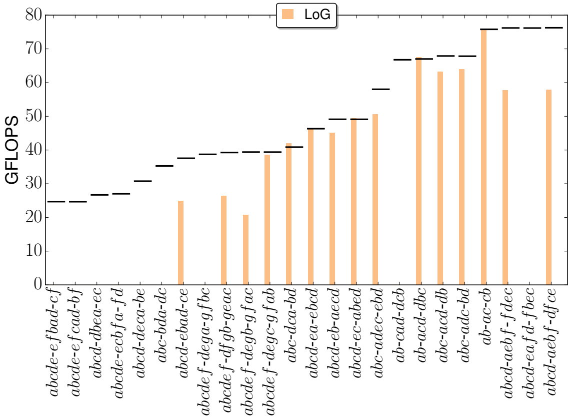
Schematic overview of TCCG[10].

[10] Available at: `hptts://github.com/hpac/tccg`

Performance — Haswell (single core)

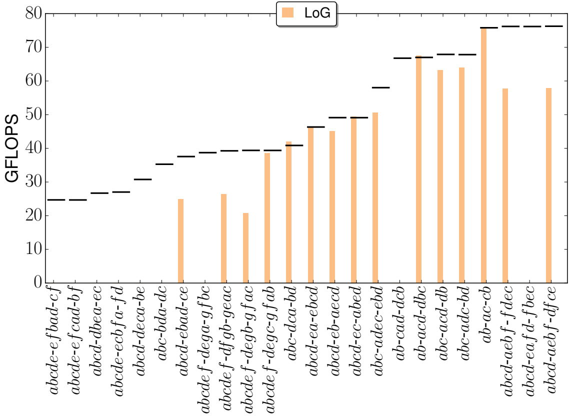


Performance — Haswell (single core)

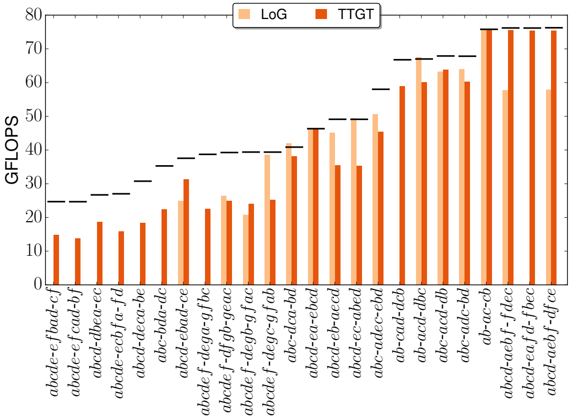


- Not all TCs can be implemented via LoG

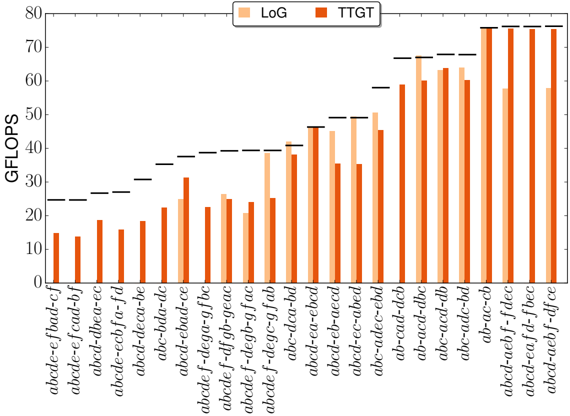
Performance — Haswell (single core)



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

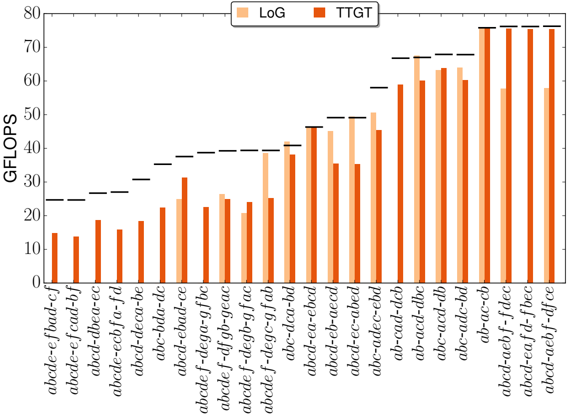

LoG
TTGT
GFLOPS
0
10
20
30
40
50
60
70
80
abcde-efbad-cf
abcde-efcad-bf
abcd-dbea-ec
abcde-ecbfa-fd
abcd-deca-be
abc-bda-dc
abcd-ebad-ce
abcdef-dega-gfbc
abcdef-dfgb-geac
abcdef-degb-gfac
abcdef-degc-gfab
abc-dca-bd
abcd-ea-ebcd
abcd-eb-aecd
abcd-ec-abed
abc-adec-ebd
ab-cad-dcb
ab-acd-dbc
abc-acd-db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

Performance — Haswell (single core)



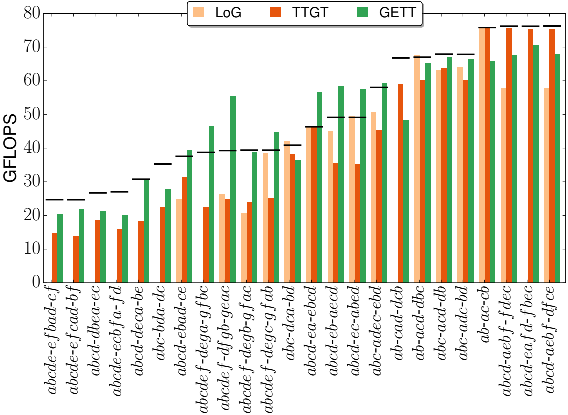
- TTGT: good for compute-bound TCs

Performance — Haswell (single core)

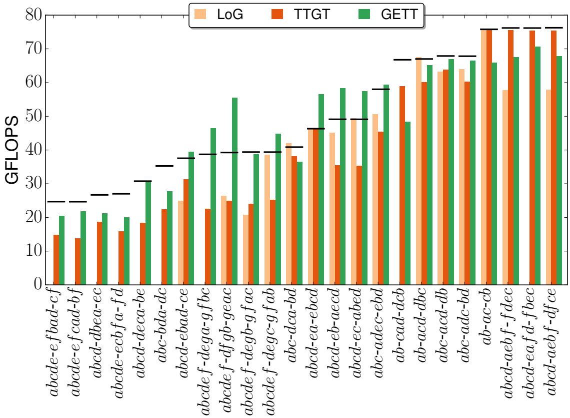


- TTGT: good for compute-bound TCs
- TTGT: bad for bandwidth-bound TCs

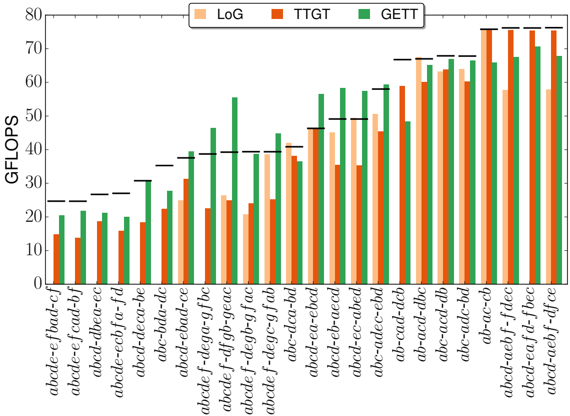
Performance — Haswell (single core)



Performance — Haswell (single core)

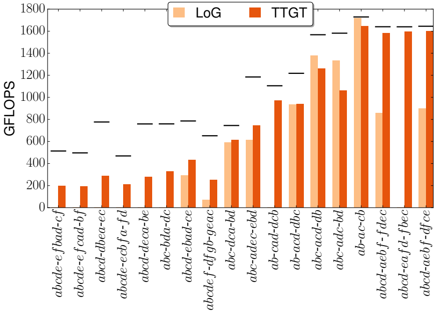


- GETT: excels for bandwidth-bound TCs



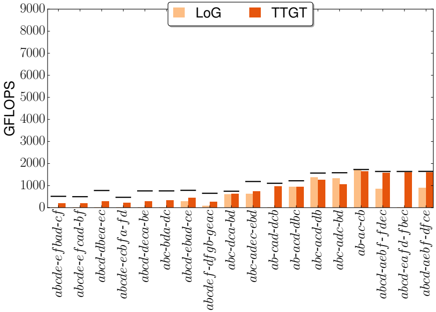
- GETT: excels for bandwidth-bound TCs
- GETT: good for compute-bound TCs

Performance — Multi-threaded

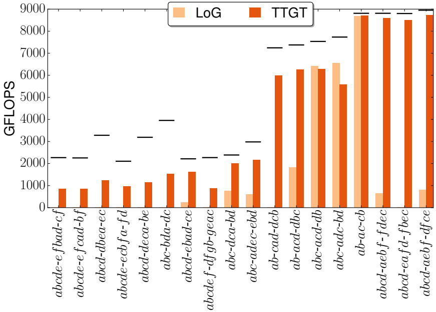


(a) 2×Intel Xeon E5-2680 v3

- Performance gap increases for bandwidth-bound TCs



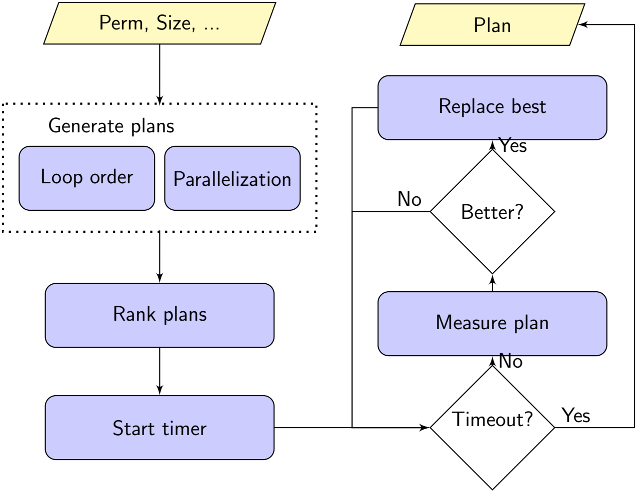
(a) 2×Intel Xeon E5-2680 v3



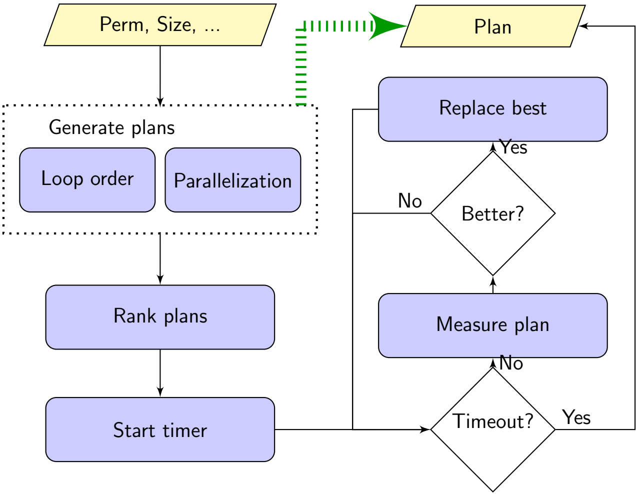
(b) NVIDIA Tesla P100

- Performance gap increases for bandwidth-bound TCs

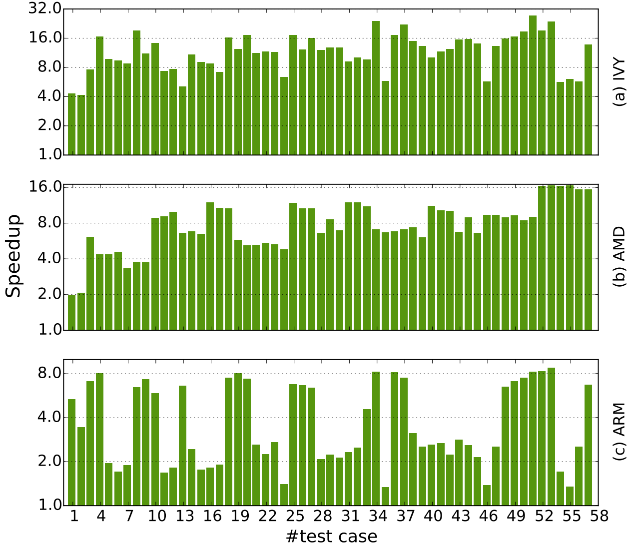
Plan Creation & Autotuning



Plan Creation & Autotuning



Speedup over Eigen

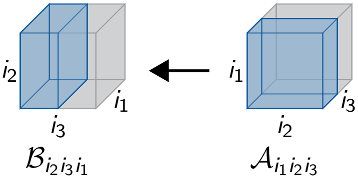


Speedup over Eigen.

j

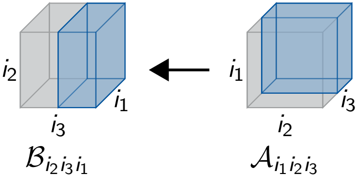
Plan Example

for $i_3 = 0 : N/2$
  for $i_1 = 0 : N$
    for $i_2 = 0 : N$
      $\mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$



(a) Thread 1

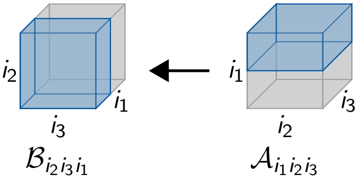
for $i_3 = N/2 : N$
  for $i_1 = 0 : N$
    for $i_2 = 0 : N$
      $\mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$



(b) Thread 2

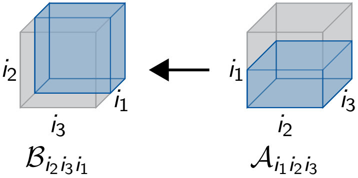
Plan Example

$$
\begin{aligned}
&\texttt{for}\ \ i_3 = 0 : N\\
&\quad \texttt{for}\ \ i_2 = 0 : N\\
&\quad\quad \texttt{for}\ \ i_1 = 0 : N/2\\
&\quad\quad\quad \mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]
\end{aligned}
$$



(a) Thread 1

$$
\begin{aligned}
&\texttt{for}\ \ i_3 = N\\
&\quad \texttt{for}\ \ i_2 = 0 : N\\
&\quad\quad \texttt{for}\ \ i_1 = N/2 : N\\
&\quad\quad\quad \mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]
\end{aligned}
$$



(b) Thread 2