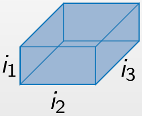
HPTT: A High-Performance Tensor Transposition C++ Library

Paul Springer, Tong Su and Paolo Bientinesi

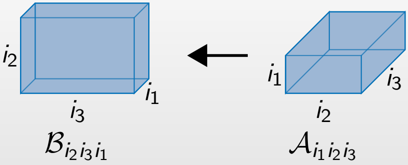
Aachen Institute for Advanced Study in
Computational Engineering Science

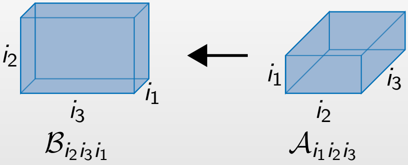
Barcelona, 18.06.17 — ARRAY'17

Tensor Transpositions

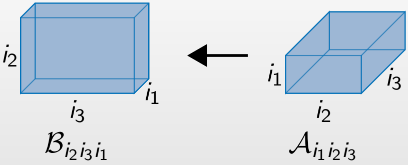


$\mathcal{A}_{i_1 i_2 i_3}$


i_2
i_1
i_3
$\mathcal{B}_{i_2 i_3 i_1}$
i_1
i_3
i_2
$\mathcal{A}_{i_1 i_2 i_3}$

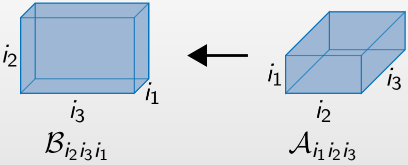


- Challenges
  - Non-consecutive memory accesses
  - Huge search space of viable implementations



- Challenges
    - Non-consecutive memory accesses
    - Huge search space of viable implementations
- Applications
    - Tensor contractions
        - Flatten a tensor into a matrix

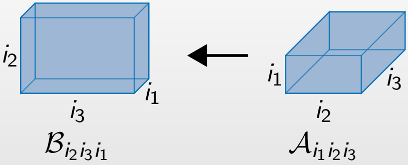
RWTH AACHEN UNIVERSITY



- Challenges
  - Non-consecutive memory accesses
  - Huge search space of viable implementations
- Applications
  - Tensor contractions
    - Flatten a tensor into a matrix
  - Packing routines for dense linear algebra kernels
    - e.g., GEMM

AI ces

Tensor Transpositions



- Challenges
  - Non-consecutive memory accesses
  - Huge search space of viable implementations
- Applications
  - Tensor contractions
    - Flatten a tensor into a matrix
  - Packing routines for dense linear algebra kernels
    - e.g., GEMM
  - Machine Learning

- Transpositions of the general form:

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

- Transpositions of the general form:

$$\mathcal{B}_{i_1,i_2,\dots,i_N} \leftarrow \alpha \times \mathcal{A}_{\Pi(i_1,i_2,\dots,i_N)} + \beta \times \mathcal{B}_{i_1,i_2,\dots,i_N}$$

- Predecessor: **T**ensor **T**ransposition **C**ompiler (TTC)
  - Limitation: parameters are required at compile-time

- Transpositions of the general form:

$$\mathcal{B}_{i_1,i_2,\dots,i_N} \leftarrow \alpha \times \mathcal{A}_{\Pi(i_1,i_2,\dots,i_N)} + \beta \times \mathcal{B}_{i_1,i_2,\dots,i_N}$$

- Predecessor: **T**ensor **T**ransposition **C**ompiler (TTC)
  - Limitation: parameters are required at compile-time
- C++ 11 library for tensor transpositions
  - Explicitly vectorized
  - Multi-threaded
  - Autotuning

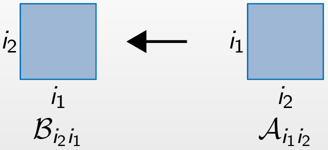
- Transpositions of the general form:

$$\mathcal{B}_{i_1,i_2,\dots,i_N} \leftarrow \alpha \times \mathcal{A}_{\Pi(i_1,i_2,\dots,i_N)} + \beta \times \mathcal{B}_{i_1,i_2,\dots,i_N}$$

- Predecessor: **T**ensor **T**ransposition **C**ompiler (TTC)
  - Limitation: parameters are required at compile-time
- C++ 11 library for tensor transpositions
  - Explicitly vectorized
  - Multi-threaded
  - Autotuning
- Dynamic data structure called `plan` (similar to FFTW)
  - Encodes the execution of a tensor transposition
    - Loop Order
    - Parallelism

Reduction to 2D

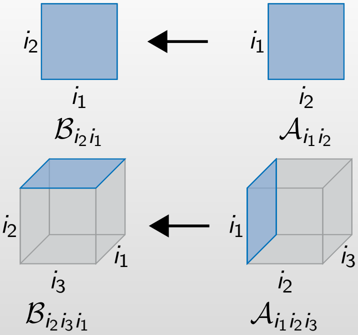
$$
\begin{array}{l}
\texttt{for}\ \ i_1 = 0 : N \\
\quad \texttt{for}\ \ i_2 = 0 : N \\
\quad\quad \mathcal{B}[i_2, i_1] \leftarrow \mathcal{A}[i_1, i_2]
\end{array}
$$



Reduction to 2D

```
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]
```

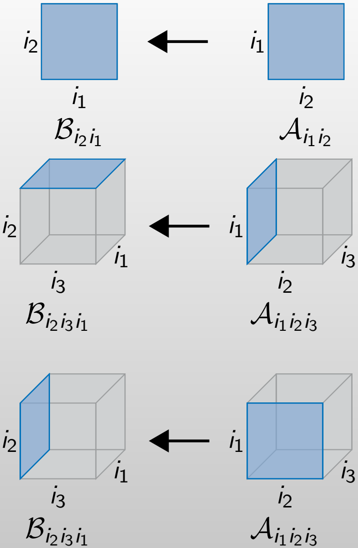


Reduction to 2D

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

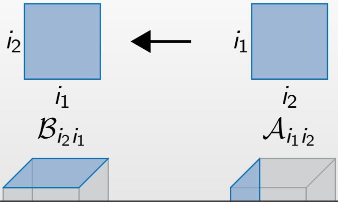
for $i_2 = 0 : N$
  for $i_3 = 0 : N$
    for $i_1 = 0 : N$
      $\mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$

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



Reduction to 2D

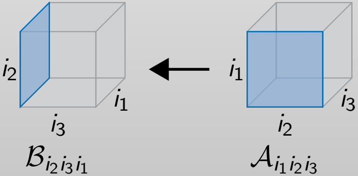
$$\texttt{for}\ i_1 = 0:N\\ \quad\texttt{for}\ i_2 = 0:N\\ \quad\quad \mathcal{B}[i_2, i_1] \leftarrow \mathcal{A}[i_1, i_2]$$



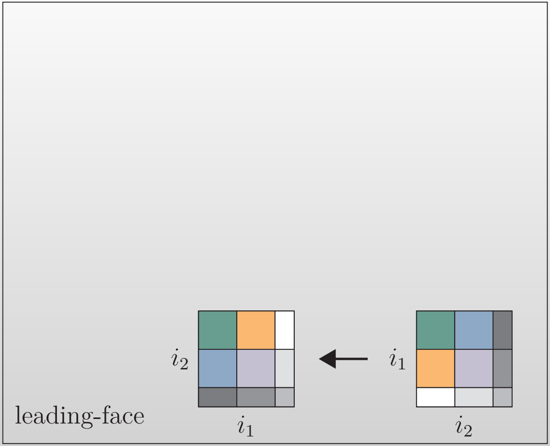
Summary

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

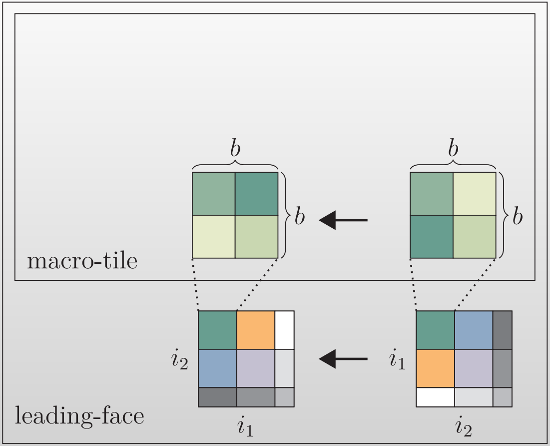
$$\texttt{for}\ i_3 = 0:N\\ \quad\texttt{for}\ i_1 = 0:N\\ \quad\quad\texttt{for}\ i_2 = 0:N\\ \quad\quad\quad \mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$$



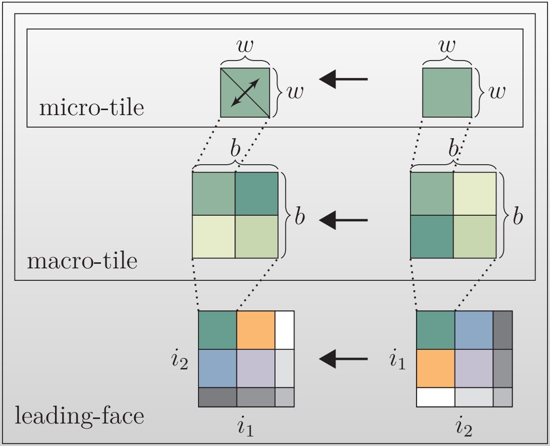
Blocking Overview



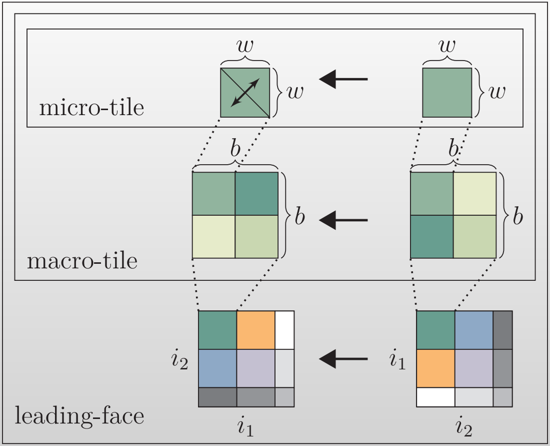
Blocking Overview



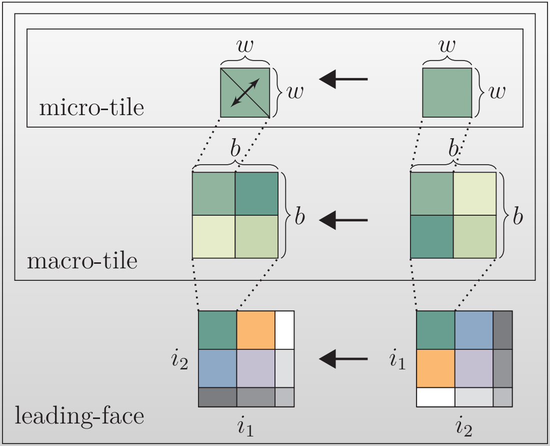
Blocking Overview



Blocking Overview



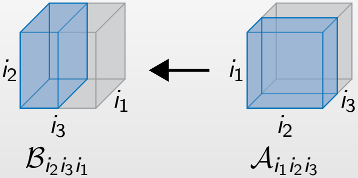
- Decompose a macro-tile into micro-tiles
  - Vectorized micro-tiles



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

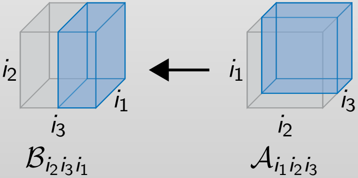
Plan Example

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

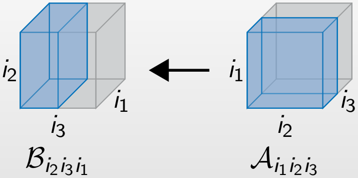
for $i_3 = N/2 : N$
  for $i_1 = 0 : N$
    for $i_2 = 0 : N$
      $\mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$



(b) Thread 2

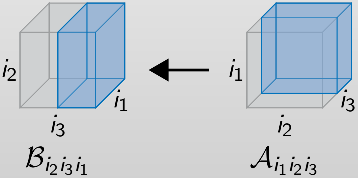
Plan Example

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



(a) Thread 1

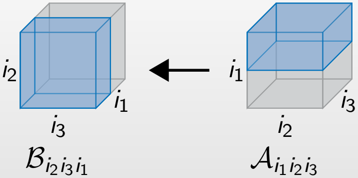
for $i_3 = N/2 : N$
  for $i_2 = 0 : N$
    for $i_1 = 0 : N$
      $\mathcal{B}[i_2, i_3, i_1] \leftarrow \mathcal{A}[i_1, i_2, i_3]$



(b) Thread 2

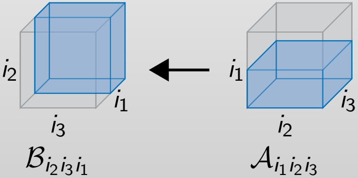
Plan Example

```
for i3 = 0 : N
  for i2 = 0 : N
    for i1 = 0 : N/2
      B[i2, i3, i1] ← A[i1, i2, i3]
```

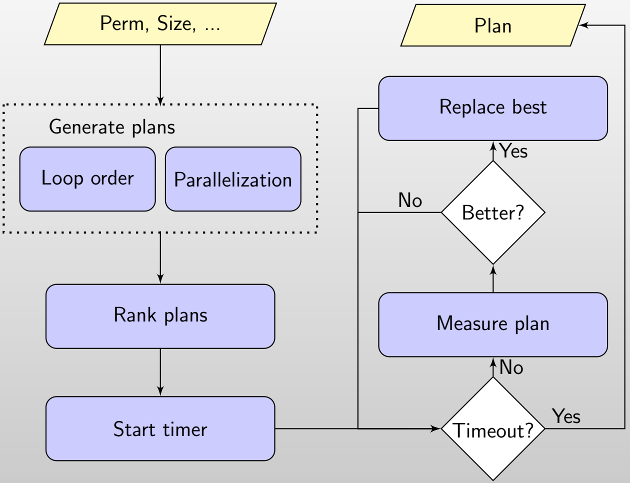


(a) Thread 1

```
for i3 = N
  for i2 = 0 : N
    for i1 = N/2 : N
      B[i2, i3, i1] ← A[i1, i2, i3]
```

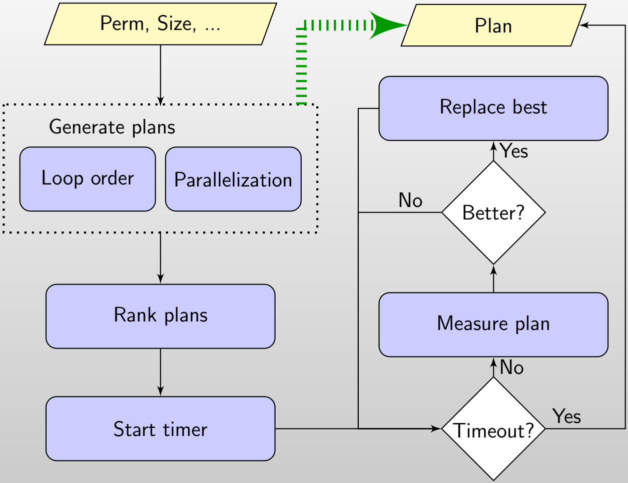


(b) Thread 2

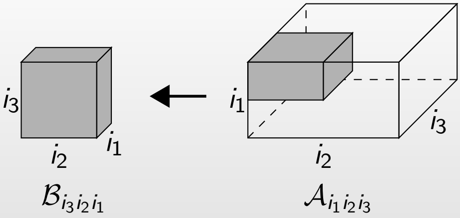


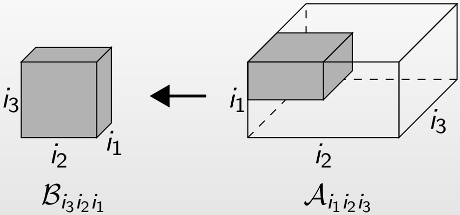
Perm, Size, ...
Generate plans
Loop order
Parallelization
Rank plans
Start timer
Timeout?
Yes
No
Measure plan
Better?
No
Yes
Replace best
Plan

Plan Creation & Autotuning



Example



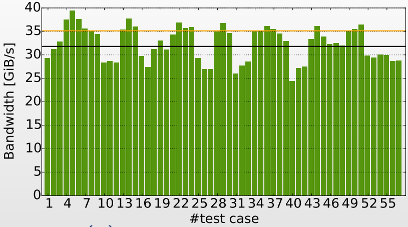


- Code Example:

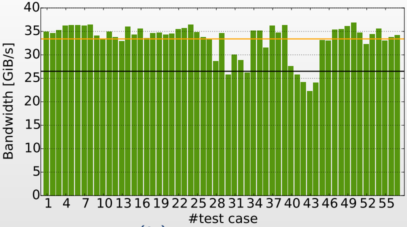
```
// specify permutation and size
std::vector<uint32_t> perm       = { 2,  1,  0};
std::vector<uint32_t> sizeA      = { 8, 16, 16};
std::vector<uint32_t> outerSizeA = {16, 32, 32};
std::vector<uint32_t> outerSizeB = {16, 16,  8};

// create a plan
double timeout = 1.0; // in seconds
auto plan = hptt::create_plan(perm,
                  1.0 /*alpha*/, A, sizeA, outerSizeA,
                  0.0 /*beta*/ , B,        outerSizeB,
                  numThreads, timeout);

// execute the transposition
plan->exec();
```

```
float * A_packed[MC * KC];
int dim     = ...
int perm[] = ...
int size[] = ...
int outerA[]  = ...
int outerB[]  = ...
auto planOuter = hptt::create_plan( perm, dim,
                                    alpha, A, size, outerA,
                                    beta,  A_packed, outerB,
                                    numThreads, 0.0);

for( int im_ = 0; im_ < (M / MC); ++im_ )
{
   for( int ik_ = 0; ik_ < (K_ / KC); ++ik_ )
   {
      // pack A
      planOuter->setInputPtr(&A[im_ + ik_ * lda]);
      planOuter->exec();

      // Use A_packed ...
   }
}
```

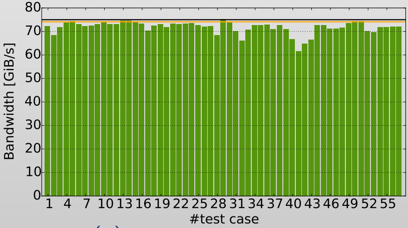
Listing 1: Tensor Contraction Example.



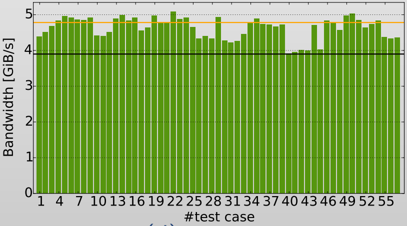
(a) Intel Ivy Bridge E5-2670 v2.



(b) IBM Power7.

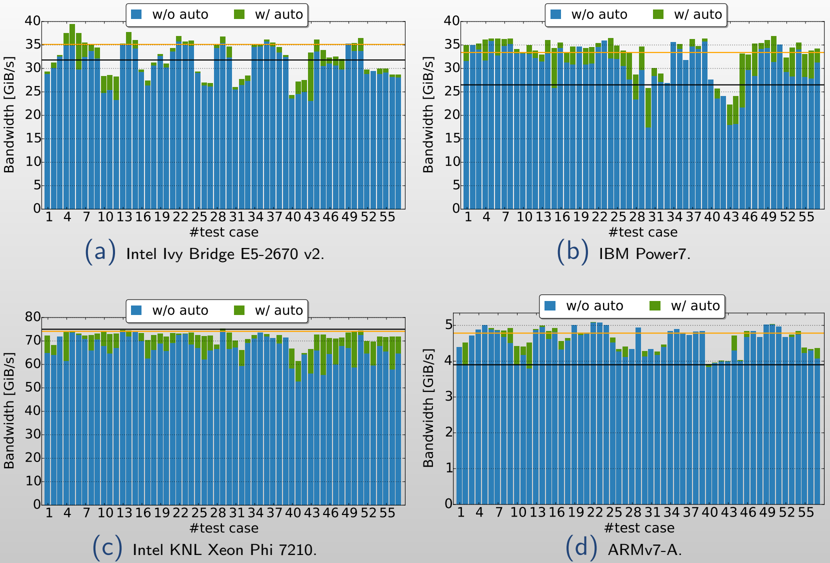


(c) Intel KNL Xeon Phi 7210.



(d) ARMv7-A.

Bandwidth



(a) Intel Ivy Bridge E5-2670 v2.

(b) IBM Power7.

(c) Intel KNL Xeon Phi 7210.

(d) ARMv7-A.

Speedup over Eigen

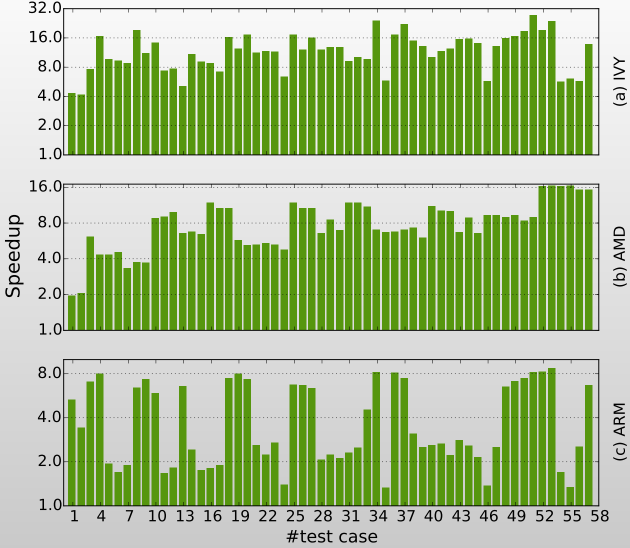


Figure: Speedup over Eigen.

- Tensor Contractions
  - Transpositions: Flatten tensors into matrices

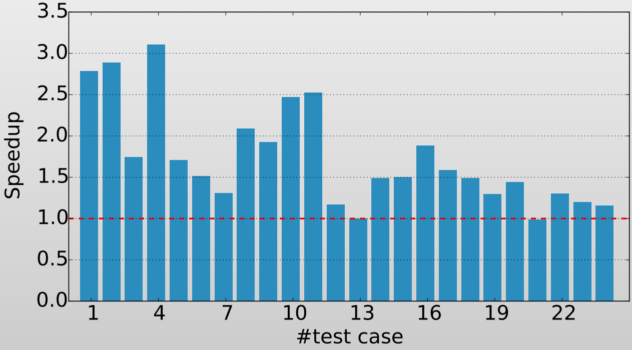


Figure: HPTT's impact on CTF's performance.

Summary

- C++ library for high-performance tensor transpositions
- Features:
  - Multiple architectures
  - All numerical data types
  - Supports subtensors
  - Autotuning
- Give it a try :)
  - `https://github.com/springer13/hptt`[1]

[1] Published under LGPLv3.

Summary

- C++ library for high-performance tensor transpositions
- Features:
  - Multiple architectures
  - All numerical data types
  - Supports subtensors
  - Autotuning
- Give it a try :)
  - `https://github.com/springer13/hptt`[1]

Thank you for your attention.

[1] Published under LGPLv3.

RWTH AACHEN UNIVERSITY

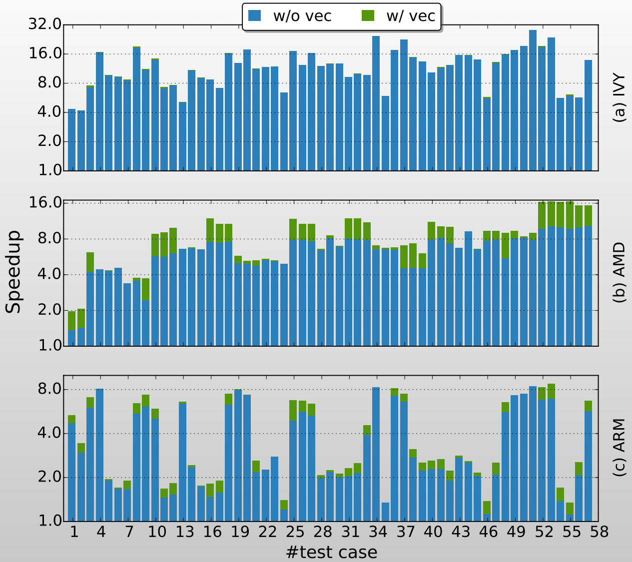
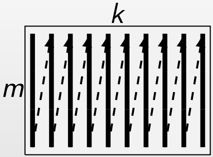
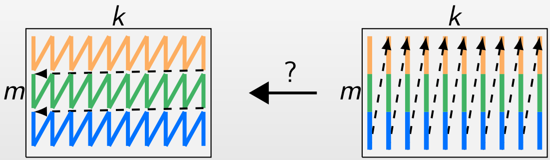
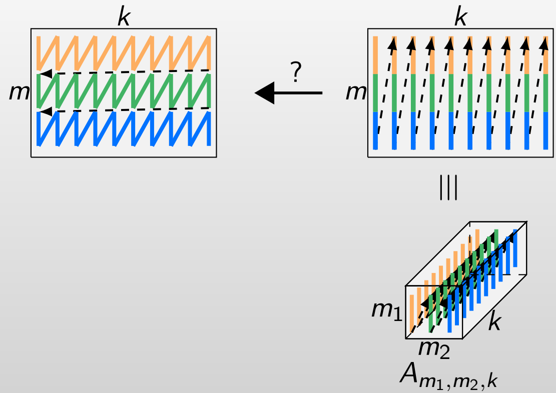


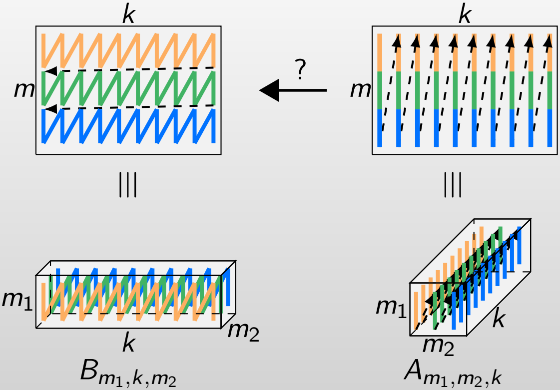
Figure: Speedup over Eigen.

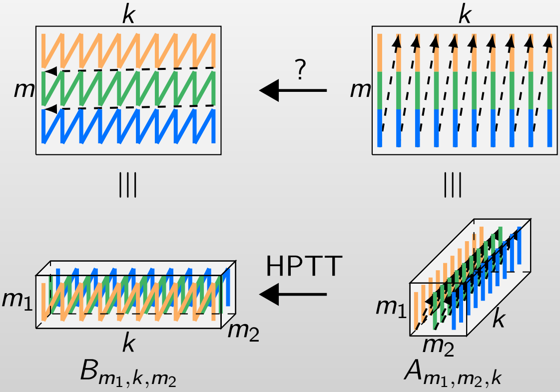
Example 2




k
m
?
k
m


k
m
?
k
m
|||
m_1
k
m_2
$A_{m_1,m_2,k}$


k
m
?
k
m
|||
|||
m1
k
m2
Bm1,k,m2
m1
m2
k
Am1,m2,k


k
m
?
k
m
|||
|||
m1
k
m2
B_{m_1,k,m_2}
HPTT
m1
m2
k
$A_{m_1,m_2,k}$