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

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

Paul Springer (AICES)
High-Performance Tensor Transpositions
18.06.17 2 / 12
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
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
Tensor Transpositions

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

Applications
- Tensor contractions
  - Flatten a tensor into a matrix
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
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:

\[ B_{i_1, i_2, \ldots, i_N} \leftarrow \alpha \times A_{\Pi(i_1, i_2, \ldots, i_N)} + \beta \times B_{i_1, i_2, \ldots, i_N} \]
Transpositions of the general form:

\[ B_{i_1,i_2,...,i_N} \leftarrow \alpha \times A_{\prod(i_1,i_2,...,i_N)} + \beta \times B_{i_1,i_2,...,i_N} \]

Predecessor: Tensor Transposition Compiler (TTC)
- Limitation: parameters are required at compile-time
Transpositions of the general form:

\[ B_{i_1,i_2,...,i_N} \leftarrow \alpha \times A_{\prod(i_1,i_2,...,i_N)} + \beta \times B_{i_1,i_2,...,i_N} \]

Predecessor: **Tensor Transposition Compiler (TTC)**
- Limitation: parameters are required at compile-time

**C++ 11 library for tensor transpositions**
- Explicitly vectorized
- Multi-threaded
- Autotuning
Transpositions of the general form:

\[ B_{i_1, i_2, ..., i_N} \leftarrow \alpha \times A_{\prod(i_1, i_2, ..., i_N)} + \beta \times B_{i_1, i_2, ..., i_N} \]

Predecessor: Tensor Transposition Compiler (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
for $i_1 = 0 : N$
    for $i_2 = 0 : N$
        $B[i_2, i_1] \leftarrow A[i_1, i_2]$
Reduction to 2D

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

\[
\text{for } i_2 = 0 : N \\
\text{for } i_3 = 0 : N \\
\text{for } i_1 = 0 : N \\
B[i_2, i_3, i_1] \leftarrow A[i_1, i_2, i_3]
\]
Reduction to 2D

for \( i_1 = 0 : N \)
for \( i_2 = 0 : N \)
\( B[i_2, i_1] \leftarrow A[i_1, i_2] \)

for \( i_2 = 0 : N \)
for \( i_3 = 0 : N \)
for \( i_1 = 0 : N \)
\( B[i_2, i_3, i_1] \leftarrow A[i_1, i_2, i_3] \)

for \( i_3 = 0 : N \)
for \( i_1 = 0 : N \)
for \( i_2 = 0 : N \)
\( B[i_2, i_3, i_1] \leftarrow A[i_1, i_2, i_3] \)
Reduction to 2D

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

Summary

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

for $i_3 = 0 : N$
  for $i_1 = 0 : N$
    for $i_2 = 0 : N$
      $B[i_2, i_3, i_1] \leftarrow A[i_1, i_2, i_3]$
Decompose a macro-tile into micro-tiles

Vectorized micro-tiles

Decompose a transposition into macro-tiles

Parallel over macro-tiles
Blocking Overview

Decompose a macro-tile into micro-tiles

Vectorized micro-tiles

Decompose a transposition into macro-tiles

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

Vectorized micro-tiles

Decompose a transposition into macro-tiles

Parallel over macro-tiles
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 $i_2 = 0 : N$
      $B[i_2, i_3, i_1] \leftarrow A[i_1, i_2, i_3]$

(a) Thread 1

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

(b) Thread 2
for \( i_3 = 0 : N/2 \)
  for \( i_2 = 0 : N \)
    for \( i_1 = 0 : N \)
      \( B[i_2, i_3, i_1] \leftarrow A[i_1, i_2, i_3] \)

for \( i_3 = N/2 : N \)
  for \( i_2 = 0 : N \)
    for \( i_1 = 0 : N \)
      \( B[i_2, i_3, i_1] \leftarrow A[i_1, i_2, i_3] \)
Plan Example

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

(a) Thread 1

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

(b) Thread 2
Plan Creation & Autotuning

Permutation, Size, ...

Generate plans

Loop order, Parallelization

Rank plans

Start timer

Plan

Replace best

Yes

Better?

No

Measure plan

No

Timeout?

Yes
Plan Creation & Autotuning

1. Generate plans
   - Loop order
   - Parallelization

2. Rank plans
3. Start timer
4. Measure plan
5. Replace best
   - Yes
   - No
     - Better?
     - Yes
     - No
6. Timeout?
   - Yes
   - No
Example

\[ A_{i_1 i_2 i_3} \quad \rightarrow \quad B_{i_3 i_2 i_1} \]
Code Example:

```cpp
// 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();
```
Plan Reuse

```cpp
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.
Bandwidth

(a) Intel Ivy Bridge E5-2670 v2.

(b) IBM Power7.

(c) Intel KNL Xeon Phi 7210.

(d) ARMv7-A.

Paul Springer (AICES)
Bandwidth

(a) Intel Ivy Bridge E5-2670 v2.

(b) IBM Power7.

(c) Intel KNL Xeon Phi 7210.

(d) ARMv7-A.
Figure: Speedup over Eigen.
Tensor Contractions

Transpositions: Flatten tensors into matrices

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

1Published 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

Thank you for your attention.

1Published under LGPLv3.
Figure: Speedup over Eigen.
Example 2

\[ k \equiv B m_1, m_2, k \]

\[ m \to k \]

\[ HPTT \]
Example 2

\[ A_{m_1, k, m_2} \equiv B_{m_1, k, m_2} \]

Paul Springer (AICES)
High-Performance Tensor Transpositions
18.06.17 14 / 12
Example 2

\[ A_{m_1, m_2, k} \equiv HPTT \]
Example 2

\[ B_{m_1,k,m_2} \equiv T_{m_1,m_2,k} \]

\[ A_{m_1,m_2,k} \]
Example 2

\[ B_{m_1, k, m_2} \equiv HPTT \]

\[ A_{m_1, m_2, k} \]