

High-Performance Tensor Transposition (HPTT) C++ Library

Generated by Doxygen 1.9.3

1 High-Performance Tensor Transposition Library	1
1.1 Introduction	1
1.2 Key Features	1
1.3 Requirements	1
1.4 Install	2
1.5 Getting Started	2
1.6 Benchmark	2
1.7 Citation	3
2 Namespace Index	5
2.1 Namespace List	5
3 Class Index	7
3.1 Class List	7
4 File Index	9
4.1 File List	9
5 Namespace Documentation	11
5.1 hptt Namespace Reference	11
5.1.1 Detailed Description	13
5.1.2 Enumeration Type Documentation	13
5.1.2.1 SelectionMethod	13
5.1.3 Function Documentation	13
5.1.3.1 create_plan()	13
6 Class Documentation	17
6.1 hptt::Transpose< floatType > Class Template Reference	17
6.1.1 Detailed Description	18
6.1.2 Constructor & Destructor Documentation	18
6.1.2.1 Transpose()	18
6.1.3 Member Function Documentation	19
6.1.3.1 addThreadId()	19
6.1.3.2 execute()	20
6.1.3.3 execute_expert()	20
6.1.3.4 setInputPtr()	20
6.1.3.5 setMaxAutotuningCandidates()	21
6.1.3.6 setOutputPtr()	21
7 File Documentation	23
7.1 compute_node.h	23
7.2 hptt.h	23
7.3 hptt_types.h	25
7.4 macros.h	25

7.5 plan.h	26
7.6 transpose.h	26
7.7 utils.h	28
Index	31

Chapter 1

High-Performance Tensor Transposition Library

1.1 Introduction

HPTT supports tensor transpositions of the general form:

$$\mathcal{B}_{\pi(i_0, i_1, \dots, i_{d-1})} = \alpha * \mathcal{A}_{i_0, i_1, \dots, i_{d-1}} + \beta * \mathcal{C}_{i_0, i_1, \dots, i_{d-1}}$$

where α and β are scalars and \mathcal{A} and \mathcal{B} are d-dimensional tensors (i.e., multi-dimensional arrays).

HPTT assumes a column-major data layout, thus indices are stored from left to right (e.g., i_0 is the stride-1 index in $\mathcal{A}_{i_0, i_1, \dots, i_{d-1}}$).

1.2 Key Features

- Multi-threading support
- Explicit vectorization
- Auto-tuning (akin to FFTW)
 - Loop order
 - Parallelization
- Multi-architecture support
 - Explicitly vectorized kernels for (AVX and ARM)
- Support for float, double, complex and double complex data types
- Can operate on sub-tensors

1.3 Requirements

You must have a working C++ compiler with c++11 support. I have tested HPTT with:

- Intel's ICPC 15.0.3, 16.0.3, 17.0.2
- GNU g++ 5.4, 6.2, 6.3
- clang++ 3.8, 3.9

1.4 Install

Clone the repository into a desired directory and change to that location:

```
git clone https://github.com/springer13/hptt.git
cd hptt
export CXX=<desired compiler>
```

Now you have several options to build the desired version of the library:

```
make avx
make arm
make scalar
```

This should create 'libhptt.so' inside the ./lib folder.

1.5 Getting Started

In general HPTT is used as follows:

```
#include <hptt.h>

// allocate tensors
float A* = ...
float B* = ...

// specify permutation and size
int dim = 6;
int perm[dim] = {5,2,0,4,1,3};
int size[dim] = {48,28,48,28,28};

// create a plan (shared_ptr)
auto plan = hptt::create_plan( perm, dim,
                               alpha, A, size, NULL,
                               beta, B, NULL,
                               hptt::ESTIMATE, numThreads);

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

The example above does not use any auto-tuning, but solely relies on HPTT's performance model. To active auto-tuning, please use `hptt::MEASURE`, or `hptt::PATIENT` instead of `hptt::ESTIMATE`.

Please refer to the [hptt::Transpose](#) class for additional information or to [hptt::create_plan\(\)](#).

An extensive example is provided here: `./benchmark/benchmark.cpp`.

1.6 Benchmark

The benchmark is the same as the original TTC benchmark [benchmark for tensor transpositions](#).

You can compile the benchmark via:

```
cd benchmark
make
```

Before running the benchmark, please modify the number of threads and the thread affinity within the `benchmark.sh` file. To run the benchmark just use:

```
./benchmark.sh
```

This will create `hptt_benchmark.dat` file containing all the runtime information of HPTT and the reference implementation.

1.7 Citation

In case you want refer to HPTT as part of a research paper, please cite the following article (pdf) :

```
@inproceedings{hptt2017,  
  author = {Springer, Paul and Su, Tong and Bientinesi, Paolo},  
  title = {{HPTT}: {A} {H}igh-{P}erformance {T}ensor {T}ransposition {C}++ {L}ibrary},  
  booktitle = {Proceedings of the 4th ACM SIGPLAN International Workshop on Libraries, Languages, and  
    Compilers for Array Programming},  
  series = {ARRAY 2017},  
  year = {2017},  
  isbn = {978-1-4503-5069-3},  
  location = {Barcelona, Spain},  
  pages = {56--62},  
  numpages = {7},  
  url = {http://doi.acm.org/10.1145/3091966.3091968},  
  doi = {10.1145/3091966.3091968},  
  acmid = {3091968},  
  publisher = {ACM},  
  address = {New York, NY, USA},  
  keywords = {High-Performance Computing, autotuning, multidimensional transposition, tensor transposition,  
    tensors, vectorization},  
}
```


Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

hptt	11
--------------------------------	----

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

[hptt::Transpose< floatType >](#)

The [Transpose](#) class encodes all information related to the execution of the tensor transposition [17](#)

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

include/ compute_node.h	23
include/ hptt.h	23
include/ hptt_types.h	25
include/ macros.h	25
include/ plan.h	26
include/ transpose.h	26
include/ utils.h	28

Chapter 5

Namespace Documentation

5.1 hptt Namespace Reference

Classes

- class **ComputeNode**
A ComputNode encodes a loop.
- class **Plan**
A plan encodes the execution of a tensor transposition.
- class [Transpose](#)
The [Transpose](#) class encodes all information related to the execution of the tensor transposition.

Typedefs

- using **FloatComplex** = std::complex< float >
- using **DoubleComplex** = std::complex< double >

Enumerations

- enum [SelectionMethod](#) { **ESTIMATE** , **MEASURE** , **PATIENT** , **CRAZY** }
Determines the duration of the auto-tuning process.

Functions

- std::shared_ptr< [hptt::Transpose](#)< float > > [create_plan](#) (const int *perm, const int dim, const float alpha, const float *A, const int *sizeA, const int *outerSizeA, const float beta, float *B, const int *outerSizeB, const [SelectionMethod](#) selectionMethod, const int numThreads, const int *threadIds=nullptr, const bool useRowMajor=false)
Creates a Tensor Transposition plan.
- std::shared_ptr< [hptt::Transpose](#)< double > > **create_plan** (const int *perm, const int dim, const double alpha, const double *A, const int *sizeA, const int *outerSizeA, const double beta, double *B, const int *outerSizeB, const [SelectionMethod](#) selectionMethod, const int numThreads, const int *threadIds=nullptr, const bool useRowMajor=false)

- `std::shared_ptr< hptt::Transpose< FloatComplex > > create_plan` (const int *perm, const int dim, const FloatComplex alpha, const FloatComplex *A, const int *sizeA, const int *outerSizeA, const FloatComplex beta, FloatComplex *B, const int *outerSizeB, const [SelectionMethod](#) selectionMethod, const int numThreads, const int *threadIds=nullptr, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< DoubleComplex > > create_plan` (const int *perm, const int dim, const DoubleComplex alpha, const DoubleComplex *A, const int *sizeA, const int *outerSizeA, const DoubleComplex beta, DoubleComplex *B, const int *outerSizeB, const [SelectionMethod](#) selectionMethod, const int numThreads, const int *threadIds=nullptr, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< float > > create_plan` (const std::vector< int > &perm, const int dim, const float alpha, const float *A, const std::vector< int > &sizeA, const std::vector< int > &outerSizeA, const float beta, float *B, const std::vector< int > &outerSizeB, const [SelectionMethod](#) selectionMethod, const int numThreads, const std::vector< int > &threadIds={}, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< double > > create_plan` (const std::vector< int > &perm, const int dim, const double alpha, const double *A, const std::vector< int > &sizeA, const std::vector< int > &outerSizeA, const double beta, double *B, const std::vector< int > &outerSizeB, const [SelectionMethod](#) selectionMethod, const int numThreads, const std::vector< int > &threadIds={}, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< FloatComplex > > create_plan` (const std::vector< int > &perm, const int dim, const FloatComplex alpha, const FloatComplex *A, const std::vector< int > &sizeA, const std::vector< int > &outerSizeA, const FloatComplex beta, FloatComplex *B, const std::vector< int > &outerSizeB, const [SelectionMethod](#) selectionMethod, const int numThreads, const std::vector< int > &threadIds={}, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< DoubleComplex > > create_plan` (const std::vector< int > &perm, const int dim, const DoubleComplex alpha, const DoubleComplex *A, const std::vector< int > &sizeA, const std::vector< int > &outerSizeA, const DoubleComplex beta, DoubleComplex *B, const std::vector< int > &outerSizeB, const [SelectionMethod](#) selectionMethod, const int numThreads, const std::vector< int > &threadIds={}, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< float > > create_plan` (const int *perm, const int dim, const float alpha, const float *A, const int *sizeA, const int *outerSizeA, const float beta, float *B, const int *outerSizeB, const int maxAutotuningCandidates, const int numThreads, const int *threadIds=nullptr, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< double > > create_plan` (const int *perm, const int dim, const double alpha, const double *A, const int *sizeA, const int *outerSizeA, const double beta, double *B, const int *outerSizeB, const int maxAutotuningCandidates, const int numThreads, const int *threadIds=nullptr, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< FloatComplex > > create_plan` (const int *perm, const int dim, const FloatComplex alpha, const FloatComplex *A, const int *sizeA, const int *outerSizeA, const FloatComplex beta, FloatComplex *B, const int *outerSizeB, const int maxAutotuningCandidates, const int numThreads, const int *threadIds=nullptr, const bool useRowMajor=false)
- `std::shared_ptr< hptt::Transpose< DoubleComplex > > create_plan` (const int *perm, const int dim, const DoubleComplex alpha, const DoubleComplex *A, const int *sizeA, const int *outerSizeA, const DoubleComplex beta, DoubleComplex *B, const int *outerSizeB, const int maxAutotuningCandidates, const int numThreads, const int *threadIds=nullptr, const bool useRowMajor=false)
- `template<> float conj` (float x)
- `template<> double conj` (double x)
- `template<> double getZeroThreshold< double > ()`
- `template<> double getZeroThreshold< DoubleComplex > ()`
- `template<> double getZeroThreshold< float > ()`
- `template<> double getZeroThreshold< FloatComplex > ()`
- `void trashCache` (double *A, double *B, int n)
- `template<typename t > int hasItem` (const std::vector< t > &vec, t value)
- `template<typename t > void printVector` (const std::vector< t > &vec, const char *label)
- `template<typename t > void printVector` (const std::list< t > &vec, const char *label)
- `void getPrimeFactors` (int n, std::list< int > &primeFactors)

- `template<typename t >`
`int findPos (t value, const std::vector< t > &array)`
- `int findPos (int value, const int *array, int n)`
- `int factorial (int n)`
- `void accountForRowMajor (const int *sizeA, const int *outerSizeA, const int *outerSizeB, const int *perm, int *tmpSizeA, int *tmpOuterSizeA, int *tmpouterSizeB, int *tmpPerm, const int dim, const bool useRow↵ Major)`

5.1.1 Detailed Description

Author

: Paul Springer (springer@aices.rwth-aachen.de)

5.1.2 Enumeration Type Documentation

5.1.2.1 SelectionMethod

enum `hptt::SelectionMethod`

Determines the duration of the auto-tuning process.

- ESTIMATE: 0 seconds (i.e., no auto-tuning)
- MEASURE: 10 seconds
- PATIENT: 60 seconds
- CRAZY : 3600 seconds

5.1.3 Function Documentation

5.1.3.1 create_plan()

```
std::shared_ptr< hptt::Transpose< float > > hptt::create_plan (
    const int * perm,
    const int dim,
    const float alpha,
    const float * A,
    const int * sizeA,
    const int * outerSizeA,
    const float beta,
    float * B,
    const int * outerSizeB,
    const SelectionMethod selectionMethod,
    const int numThreads,
```

```
const int * threadIds = nullptr,
const bool useRowMajor = false )
```

Creates a Tensor Transposition plan.

A tensor transposition plan is a data structure that encodes the execution of the tensor transposition. HPTT supports tensor transpositions of the form:

$$B_{\pi(i_0, i_1, \dots)} = \alpha * A_{i_0, i_1, \dots} + \beta * B_{\pi(i_0, i_1, \dots)}.$$

The plan can be reused over several transpositions.

Parameters

in	<i>perm</i>	dim-dimensional array representing the permutation of the indices. <ul style="list-style-type: none">For instance, <code>perm[] = {1,0,2}</code> denotes the following transposition: $B_{i1,i0,i2} \leftarrow A_{i0,i1,i2}$.
in	<i>dim</i>	Dimensionality of the tensors
in	<i>alpha</i>	scaling factor for A
in	<i>A</i>	Pointer to the raw-data of the input tensor A
in	<i>sizeA</i>	dim-dimensional array that stores the sizes of each dimension of A
in	<i>outerSizeA</i>	dim-dimensional array that stores the outer-sizes of each dimension of A. <ul style="list-style-type: none">This parameter may be NULL, indicating that the outer-size is equal to <code>sizeA</code>.If <code>outerSizeA</code> is not NULL, <code>outerSizeA[i] >= sizeA[i]</code> for all $0 \leq i < \text{dim}$ must hold.This option enables HPTT to operate on sub-tensors.
in	<i>beta</i>	scaling factor for B
in, out	<i>B</i>	Pointer to the raw-data of the output tensor B
in	<i>outerSizeB</i>	dim-dimensional array that stores the outer-sizes of each dimension of B. <ul style="list-style-type: none">This parameter may be NULL, indicating that the outer-size is equal to the <code>perm(sizeA)</code>.If <code>outerSizeA</code> is not NULL, <code>outerSizeB[i] >= perm(sizeA)[i]</code> for all $0 \leq i < \text{dim}$ must hold.This option enables HPTT to operate on sub-tensors.
in	<i>selectionMethod</i>	Determines if auto-tuning should be used. See hptt::SelectionMethod for details. ATTENTION: If you enable auto-tuning (e.g., <code>hptt::MEASURE</code>) then the output data will be used during the auto-tuning process. The original data (i.e., A and B), however, is preserved after this function call completes – unless your input data (i.e. A) has invalid data (e.g., NaN, inf).
in	<i>numThreads</i>	number of threads that participate in this tensor transposition.
in	<i>threadIds</i>	Array of OpenMP threadIds that participate in this tensor transposition. This parameter is only important if you want to call HPTT from within a parallel region (i.e., via <code>execute_expert()</code>).
in	<i>useRowMajor</i>	This flag indicates whether a row-major memory layout should be used (default: off = column-major).

Chapter 6

Class Documentation

6.1 `hptt::Transpose< floatType >` Class Template Reference

The [Transpose](#) class encodes all information related to the execution of the tensor transposition.

```
#include <transpose.h>
```

Public Member Functions

- [Transpose](#) (const int *sizeA, const int *perm, const int *outerSizeA, const int *outerSizeB, const int dim, const floatType *A, const floatType alpha, floatType *B, const floatType beta, const [SelectionMethod](#) selection, const int numThreads, const int *threadIds=NULLptr, const bool useRowMajor=false)
- [Transpose](#) (const [Transpose](#) &other)
- bool **getConjA** () noexcept
- void **setConjA** (bool conjA) noexcept
- int **getNumThreads** () const noexcept
- void **setNumThreads** (int numThreads) noexcept
- floatType **getAlpha** () const noexcept
- floatType **getBeta** () const noexcept
- void **setAlpha** (floatType alpha) noexcept
set the scaling factor for A
- void **setBeta** (floatType beta) noexcept
set the scaling factor for B
- void [setInputPtr](#) (const floatType *A) noexcept
Set the pointer for A.
- void [setOutputPtr](#) (floatType *B) noexcept
Set the pointer for B.
- const floatType * **getInputPtr** () const noexcept
Get raw-data pointer to A.
- floatType * **getOutputPtr** () const noexcept
Get raw-data pointer to B.
- void **resetThreadIds** () noexcept
Clears the array that stores the OpenMP threadIds. This function should only be used in conjunction with [addThreadId\(\)](#).
- void [setMaxAutotuningCandidates](#) (int num)
- void [addThreadId](#) (int threadId) noexcept

- void **printThreadIds** () const noexcept
- int **getMasterThreadId** () const noexcept
- void **createPlan** ()
Creates the plan that encodes the execution of the tensor transposition.
- template<bool useStreamingStores = true, bool spawnThreads = true, bool betasZero>
void **execute_expert** () noexcept
- void **execute** () noexcept
- void **print** () noexcept

6.1.1 Detailed Description

```
template<typename floatType>
class hptt::Transpose< floatType >
```

The [Transpose](#) class encodes all information related to the execution of the tensor transposition.

Once a transpose (henceforth referred to as plan) `t` has been created it can be executed via `t->execute()`. Moreover, a plan can be reused multiple times. For this purpose you might want to have a look at the functions:

- [setInputPtr\(\)](#)
- [setOutputPtr\(\)](#)

In addition to the normal [execute\(\)](#) function, this class also offers the [execute_expert\(\)](#) interface. This interface is intended for the expert user and offers more flexibility than [execute\(\)](#). If you want to use the expert interface, then you might want to checkout the following functions as well:

- [resetThreadIds\(\)](#)
- [addThreadId\(\)](#)

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Transpose()

```
template<typename floatType >
hptt::Transpose< floatType >::Transpose (
    const int * sizeA,
    const int * perm,
    const int * outerSizeA,
    const int * outerSizeB,
    const int dim,
    const floatType * A,
    const floatType alpha,
    floatType * B,
    const floatType beta,
    const SelectionMethod selectionMethod,
    const int numThreads,
    const int * threadIds = nullptr,
    const bool useRowMajor = false )
```

Parameters

in	<i>perm</i>	dim-dimensional array representing the permutation of the indices. <ul style="list-style-type: none">For instance, perm[] = {1,0,2} denotes the following transposition: $B_{i1,i0,i2} \leftarrow A_{i0,i1,i2}$.
in	<i>dim</i>	Dimensionality of the tensors
in	<i>alpha</i>	scaling factor for A
in	<i>A</i>	Pointer to the raw-data of the input tensor A
in	<i>sizeA</i>	dim-dimensional array that stores the sizes of each dimension of A
in	<i>outerSizeA</i>	dim-dimensional array that stores the outer-sizes of each dimension of A. <ul style="list-style-type: none">This parameter may be NULL, indicating that the outer-size is equal to sizeA.If outerSizeA is not NULL, outerSizeA[i] >= sizeA[i] for all 0 <= i < dim must hold.This option enables HPTT to operate on sub-tensors.
in	<i>beta</i>	scaling factor for B
in, out	<i>B</i>	Pointer to the raw-data of the output tensor B
in	<i>outerSizeB</i>	dim-dimensional array that stores the outer-sizes of each dimension of B. <ul style="list-style-type: none">This parameter may be NULL, indicating that the outer-size is equal to the perm(sizeA).If outerSizeA is not NULL, outerSizeB[i] >= perm(sizeA)[i] for all 0 <= i < dim must hold.This option enables HPTT to operate on sub-tensors.
in	<i>selectionMethod</i>	Determines if auto-tuning should be used. See hptt::SelectionMethod for details. ATTENTION: If you enable auto-tuning (e.g., hptt::MEASURE) then the output data will be used during the auto-tuning process. The original data (i.e., A and B), however, is preserved after this function call completes – unless your input data (i.e. A) has invalid data (e.g., NaN, inf).
in	<i>numThreads</i>	number of threads that participate in this tensor transposition.
in	<i>threadIds</i>	Array of OpenMP threadIds that participate in this tensor transposition. This parameter is only important if you want to call HPTT from within a parallel region (i.e., via execute_expert()).
in	<i>useRowMajor</i>	This flag indicates whether a row-major memory layout should be used (default: off = column-major). Column-Major: indices are stored from left to right (leftmost = stride-1 index) Row-Major: indices are stored from right to left (right = stride-1 index)

6.1.3 Member Function Documentation

6.1.3.1 addThreadId()

```
template<typename floatType >
void hptt::Transpose< floatType >::addThreadId (
    int threadId ) [inline], [noexcept]
```

This thread-safe function adds an OpenMP threadId to the set of threads that will participate in this tensor transposition. This function is only required in conjunction with the [execute_expert\(\)](#) interface where the transposition is executed from within a parallel region (i.e., ~HPTT does not spawn the threads). It is the programmers responsibility to specify the correct thread IDs that participate in this call.

Parameters

in	<i>threadId</i>	An OpenMP threadId
----	-----------------	--------------------

6.1.3.2 execute()

```
template<typename floatType >
void hptt::Transpose< floatType >::execute ( ) [noexcept]
```

Executes the transposition. This functions requires that the plan has already been created via the [createPlan\(\)](#) function.

6.1.3.3 execute_expert()

```
template<typename floatType >
template<bool useStreamingStores = true, bool spawnThreads = true, bool betaIsZero>
void hptt::Transpose< floatType >::execute_expert ( ) [noexcept]
```

Executes the transposition. This functions requires that the plan has already been created via the [createPlan\(\)](#) function. This function behaves similarly to the [execute\(\)](#) function but it offers additional template parameters to improve performance for very small tensor transpositions. Moreover it adds more flexibility.

Parameters

in	<i>useStreamingStores</i>	Iff this variable is set, HPTT will use streaming stores which improves performance because they avoid the write-allocate traffic incurred by the write to B. However, sometimes the user might want to avoid streaming stores because the packed data fits int cache and is reused shortly (e.g., within BLAS packing routines).
in	<i>spawnThreads</i>	If the variable is set, the threads will be spawned from within this call, otherwise it is expected that this function call executes from within a parallel region.
in	<i>betalsZero</i>	Only set this variable if beta is zero.

6.1.3.4 setInputPtr()

```
template<typename floatType >
void hptt::Transpose< floatType >::setInputPtr (
    const floatType * A ) [inline], [noexcept]
```

Set the pointer for A.

This features is especially useful if one wants to reuse the transposition over multiple invocations.

6.1.3.5 setMaxAutotuningCandidates()

```
template<typename floatType >
void hptt::Transpose< floatType >::setMaxAutotuningCandidates (
    int num ) [inline]
```

[setMaxAutotuningCandidates\(\)](#) enables users to specify the number of candidates that should be tested during the autotuning phase

6.1.3.6 setOutputPtr()

```
template<typename floatType >
void hptt::Transpose< floatType >::setOutputPtr (
    floatType * B ) [inline], [noexcept]
```

Set the pointer for B.

This features is especially useful if one wants to reuse the transposition over multiple invocations.

The documentation for this class was generated from the following file:

- include/transpose.h

Chapter 7

File Documentation

7.1 compute_node.h

```
1 #pragma once
2
3 namespace hptt {
4
5     class ComputeNode
6     {
7     public:
8         ComputeNode() : start(-1), end(-1), inc(-1), lda(-1), ldb(-1), next(nullptr) {}
9
10        ~ComputeNode() {
11            if ( next != nullptr )
12                delete next;
13        }
14
15        size_t start;
16        size_t end;
17        size_t inc;
18        size_t lda;
19        size_t ldb;
20        ComputeNode *next;
21    };
22 }
```

7.2 hptt.h

```
1 /*
2  * Copyright (C) 2017 Paul Springer (springer@ices.rwth-aachen.de)
3  *
4  * This program is free software: you can redistribute it and/or modify
5  * it under the terms of the GNU Lesser General Public License as published by
6  * the Free Software Foundation, either version 3 of the License, or
7  * (at your option) any later version.
8  *
9  * This program is distributed in the hope that it will be useful,
10 * but WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 * GNU General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program. If not, see <http://www.gnu.org/licenses/>.
16 */
17
18 #pragma once
19
20 #include <vector>
21 #include <memory>
22
23 #ifdef _OPENMP
24 #include <omp.h>
25 #endif
26
27 #include "transpose.h"
```

```

152
153
154 namespace hptt {
155
156 std::shared_ptr<hptt::Transpose<float>> > create_plan( const int *perm, const int dim,
157 const float alpha, const float *A, const int *sizeA, const int *outerSizeA,
158 const float beta, float *B, const int *outerSizeB,
159 const SelectionMethod selectionMethod,
160 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
161
162 std::shared_ptr<hptt::Transpose<double>> > create_plan( const int *perm, const int dim,
163 const double alpha, const double *A, const int *sizeA, const int *outerSizeA,
164 const double beta, double *B, const int *outerSizeB,
165 const SelectionMethod selectionMethod,
166 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
167
168 std::shared_ptr<hptt::Transpose<FloatComplex>> > create_plan( const int *perm, const int dim,
169 const FloatComplex alpha, const FloatComplex *A, const int *sizeA, const int
170 *outerSizeA,
171 const FloatComplex beta, FloatComplex *B, const int *outerSizeB,
172 const SelectionMethod selectionMethod,
173 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
174
175 std::shared_ptr<hptt::Transpose<DoubleComplex>> > create_plan( const int *perm, const int dim,
176 const DoubleComplex alpha, const DoubleComplex *A, const int *sizeA, const int
177 *outerSizeA,
178 const DoubleComplex beta, DoubleComplex *B, const int *outerSizeB,
179 const SelectionMethod selectionMethod,
180 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
181
182 std::shared_ptr<hptt::Transpose<float>> > create_plan( const std::vector<int> &perm, const int dim,
183 const float alpha, const float *A, const std::vector<int> &sizeA, const
184 std::vector<int> &outerSizeA,
185 const float beta, float *B, const std::vector<int> &outerSizeB,
186 const SelectionMethod selectionMethod,
187 const int numThreads, const std::vector<int> &threadIds = {}, const bool useRowMajor =
188 false);
189
190 std::shared_ptr<hptt::Transpose<double>> > create_plan( const std::vector<int> &perm, const int dim,
191 const double alpha, const double *A, const std::vector<int> &sizeA, const
192 std::vector<int> &outerSizeA,
193 const double beta, double *B, const std::vector<int> &outerSizeB,
194 const SelectionMethod selectionMethod,
195 const int numThreads, const std::vector<int> &threadIds = {}, const bool useRowMajor =
196 false);
197
198 std::shared_ptr<hptt::Transpose<FloatComplex>> > create_plan( const std::vector<int> &perm, const int
199 dim,
200 const FloatComplex alpha, const FloatComplex *A, const std::vector<int> &sizeA, const
201 std::vector<int> &outerSizeA,
202 const FloatComplex beta, FloatComplex *B, const std::vector<int> &outerSizeB,
203 const SelectionMethod selectionMethod,
204 const int numThreads, const std::vector<int> &threadIds = {}, const bool useRowMajor =
205 false);
206
207 std::shared_ptr<hptt::Transpose<DoubleComplex>> > create_plan( const std::vector<int> &perm, const int
208 dim,
209 const DoubleComplex alpha, const DoubleComplex *A, const std::vector<int> &sizeA, const
210 std::vector<int> &outerSizeA,
211 const DoubleComplex beta, DoubleComplex *B, const std::vector<int> &outerSizeB,
212 const SelectionMethod selectionMethod,
213 const int numThreads, const std::vector<int> &threadIds = {}, const bool useRowMajor =
214 false);
215
216 std::shared_ptr<hptt::Transpose<float>> > create_plan( const int *perm, const int dim,
217 const float alpha, const float *A, const int *sizeA, const int *outerSizeA,
218 const float beta, float *B, const int *outerSizeB,
219 const int maxAutotuningCandidates,
220 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
221
222 std::shared_ptr<hptt::Transpose<double>> > create_plan( const int *perm, const int dim,
223 const double alpha, const double *A, const int *sizeA, const int *outerSizeA,
224 const double beta, double *B, const int *outerSizeB,
225 const int maxAutotuningCandidates,
226 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
227
228 std::shared_ptr<hptt::Transpose<FloatComplex>> > create_plan( const int *perm, const int dim,
229 const FloatComplex alpha, const FloatComplex *A, const int *sizeA, const int
230 *outerSizeA,
231 const FloatComplex beta, FloatComplex *B, const int *outerSizeB,
232 const int maxAutotuningCandidates,
233 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor =
234 false);
235
236 std::shared_ptr<hptt::Transpose<DoubleComplex>> > create_plan( const int *perm, const int dim,
237 const DoubleComplex alpha, const DoubleComplex *A, const int *sizeA, const int
238 *outerSizeA,
239 const DoubleComplex beta, DoubleComplex *B, const int *outerSizeB,
240 const int maxAutotuningCandidates,
241 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor =
242 false);
243
244 std::shared_ptr<hptt::Transpose<float>> > create_plan( const int *perm, const int dim,
245 const float alpha, const float *A, const int *sizeA, const int *outerSizeA,
246 const float beta, float *B, const int *outerSizeB,
247 const int maxAutotuningCandidates,
248 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
249
250 std::shared_ptr<hptt::Transpose<double>> > create_plan( const int *perm, const int dim,
251 const double alpha, const double *A, const int *sizeA, const int *outerSizeA,
252 const double beta, double *B, const int *outerSizeB,
253 const int maxAutotuningCandidates,
254 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
255
256 std::shared_ptr<hptt::Transpose<FloatComplex>> > create_plan( const int *perm, const int dim,
257 const FloatComplex alpha, const FloatComplex *A, const int *sizeA, const int
258 *outerSizeA,
259 const FloatComplex beta, FloatComplex *B, const int *outerSizeB,
260 const int maxAutotuningCandidates,
261 const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
262
263 std::shared_ptr<hptt::Transpose<DoubleComplex>> > create_plan( const int *perm, const int dim,

```

```

262         const DoubleComplex alpha, const DoubleComplex *A, const int *sizeA, const int
    *outerSizeA,
263         const DoubleComplex beta, DoubleComplex *B, const int *outerSizeB,
264         const int maxAutotuningCandidates,
265         const int numThreads, const int *threadIds = nullptr, const bool useRowMajor = false);
266 }
267
268 extern "C"
269 {
270 void sTensorTranspose( const int *perm, const int dim,
271                       const float alpha, const float *A, const int *sizeA, const int *outerSizeA,
272                       const float beta, float *B, const int *outerSizeB,
273                       const int numThreads, const int useRowMajor = 0);
274
275 void dTensorTranspose( const int *perm, const int dim,
276                       const double alpha, const double *A, const int *sizeA, const int *outerSizeA,
277                       const double beta, double *B, const int *outerSizeB,
278                       const int numThreads, const int useRowMajor = 0);
279
280 void cTensorTranspose( const int *perm, const int dim,
281                       const float _Complex alpha, bool conjA, const float _Complex *A, const int *sizeA,
282                       const int *outerSizeA,
283                       const float _Complex beta, float _Complex *B,
284                       const int *outerSizeB,
285                       const int numThreads, const int useRowMajor = 0);
286
287 void zTensorTranspose( const int *perm, const int dim,
288                       const double _Complex alpha, bool conjA, const double _Complex *A, const int *sizeA,
289                       const int *outerSizeA,
290                       const double _Complex beta, double _Complex *B,
291                       const int *outerSizeB,
292                       const int numThreads, const int useRowMajor = 0);
293 }

```

7.3 hptt_types.h

```

1 #pragma once
2
3 #include <complex>
4 #include <complex.h>
5
6 #define REGISTER_BITS 256 // AVX
7 #ifdef HPTT_ARCH_ARM
8 #undef REGISTER_BITS
9 #define REGISTER_BITS 128 // ARM
10 #endif
11
12 namespace hptt {
13
14 enum SelectionMethod { ESTIMATE, MEASURE, PATIENT, CRAZY };
15
16 using FloatComplex = std::complex<float>;
17 using DoubleComplex = std::complex<double>;
18
19 }
20

```

7.4 macros.h

```

1
20 #pragma once
21
22 #ifdef DEBUG
23 #define HPTT_ERROR_INFO(str) fprintf(stdout, "[INFO] %s:%d : %s\n", __FILE__, __LINE__, str); exit(-1);
24 #else
25 #define HPTT_ERROR_INFO(str)
26 #endif
27
28 #if defined(__ICC) || defined(__INTEL_COMPILER)
29 #define INLINE __forceinline
30 #elif defined(__GNUC__) || defined(__GNUG__)
31 #define INLINE __attribute__((always_inline)) inline
32 #endif
33
34 #ifdef _OPENMP
35
36 #define HPTT_DUPLICATE_2(condition, ...) \
37 if (condition) { _Pragma("omp parallel for num_threads(numThreads) collapse(2)") \
38     __VA_ARGS__ } \

```

```

39 else          { __VA_ARGS__ }
40
41 #define HPTT_DUPLICATE(condition, ...) \
42 if (condition) { _Pragma("omp parallel for num_threads(numThreads)") \
43     __VA_ARGS__ } \
44 else          { __VA_ARGS__ }
45
46 #else
47
48 #define HPTT_DUPLICATE(condition, ...) { __VA_ARGS__ }
49 #define HPTT_DUPLICATE_2(condition, ...) { __VA_ARGS__ }
50
51 #endif

```

7.5 plan.h

```

1 #pragma once
2
3 #include <vector>
4
5 #include "plan.h"
6
7
8 namespace hptt {
9
10 class ComputeNode;
11
12 class Plan
13 {
14 public:
15     Plan() : rootNodes_(nullptr), numTasks_(0) { }
16
17     Plan(std::vector<int> loopOrder, std::vector<int> numThreadsAtLoop);
18
19     ~Plan();
20
21     const ComputeNode* getRootNode_const(int threadId) const;
22     ComputeNode* getRootNode(int threadId) const;
23     int getNumTasks() const { return numTasks_; }
24
25     void print() const;
26
27 private:
28     int numTasks_;
29     std::vector<int> loopOrder_;
30     std::vector<int> numThreadsAtLoop_;
31     ComputeNode *rootNodes_;
32 };
33
34 }
35
36 }

```

7.6 transpose.h

```

1 #pragma once
2
3 #include <list>
4 #include <vector>
5 #include <memory>
6 #include <algorithm>
7
8 #include <stdio.h>
9 #ifdef _OPENMP
10 #include <omp.h>
11 #endif
12
13 #include "hptt_types.h"
14
15 namespace hptt {
16
17     class Plan;
18
19 template<typename floatType>
20 class Transpose
21 {
22 public:
23
24     /*****

```

```

43 * Cons, Decons, Copy, ...
44 *****/
76 Transpose( const int *sizeA,
77             const int *perm,
78             const int *outerSizeA,
79             const int *outerSizeB,
80             const int dim,
81             const floatType *A,
82             const floatType alpha,
83             floatType *B,
84             const floatType beta,
85             const SelectionMethod selectionMethod,
86             const int numThreads,
87             const int *threadIds = nullptr,
88             const bool useRowMajor = false );
89
90 Transpose(const Transpose &other);
91
92 ~Transpose();
93
94 /*****
95 * Getter & Setter
96 *****/
97 bool getConjA() noexcept { return conjA_; }
98 void setConjA(bool conjA) noexcept { conjA_ = conjA; }
99 int getNumThreads() const noexcept { return numThreads_; }
100 void setNumThreads(int numThreads) noexcept { numThreads_ = numThreads; }
101 floatType getAlpha() const noexcept { return alpha_; }
102 floatType getBeta() const noexcept { return beta_; }
106 void setAlpha(floatType alpha) noexcept { alpha_ = alpha; }
110 void setBeta(floatType beta) noexcept { beta_ = beta; }
117 void setInputPtr(const floatType *A) noexcept { A_ = A; }
124 void setOutputPtr(floatType *B) noexcept { B_ = B; }
128 const floatType* getInputPtr() const noexcept { return A_; }
132 floatType* getOutputPtr() const noexcept { return B_; }
133
138 void resetThreadIds() noexcept { threadIds_.clear(); }
139
144 void setMaxAutotuningCandidates (int num) { maxAutotuningCandidates_ = num; }
145
156 void addThreadId(int threadId) noexcept {
157 #ifdef _OPENMP
158     omp_set_lock(&writelock);
159     threadIds_.push_back(threadId);
160     std::sort(threadIds_.begin(), threadIds_.end());
161     omp_unset_lock(&writelock);
162 #endif
163 }
164
165 void printThreadIds() const noexcept { for( auto id : threadIds_ ) printf("%d, ",id);
printf("\n"); }
166 int getMasterThreadId() const noexcept { return threadIds_[0]; }
167
168 /*****
169 * Public Methods
170 *****/
174 void createPlan();
175
176 template<bool useStreamingStores=true, bool spawnThreads=true, bool betaIsZero>
177 void execute_expert() noexcept;
178
179 void execute() noexcept;
180
181 void print() noexcept;
182
183 private:
184 /*****
185 * Private Methods
186 *****/
187 void createPlans( std::vector<std::shared_ptr<Plan> > &plans ) const;
188 std::shared_ptr<Plan> selectPlan( const std::vector<std::shared_ptr<Plan> > &plans );
189 void fuseIndices();
190 void skipIndices(const int *_sizeA, const int* _perm, const int *_outerSizeA, const int
*_outerSizeB, const int dim);
191 void computeLeadingDimensions();
192 double loopCostHeuristic( const std::vector<int> &loopOrder ) const;
193 double parallelismCostHeuristic( const std::vector<int> &loopOrder ) const;
194 int getLocalThreadId(int myThreadId) const;
195 template<bool spawnThreads>
196 void getStartEnd(int n, int &myStart, int &myEnd) const;
197 void setParallelStrategy(int id) noexcept { selectedParallelStrategyId_ = id; }
198 void setLoopOrder(int id) noexcept { selectedLoopOrderId_ = id; }
199
200 /*****
201 * Helper Methods
202 *****/
203 // parallelizes the loops by changing the value of parallelismStrategy

```

```

228     void parallelize( std::vector<int> &parallelismStrategy,
229                      std::vector<int> &availableParallelismAtLoop,
230                      int &totalTasks,
231                      std::list<int> &primeFactors,
232                      const float minBalancing,
233                      const std::vector<int> &loopsAllowed) const;
234     float getLoadBalance( const std::vector<int> &parallelismStrategy ) const;
235     float estimateExecutionTime( const std::shared_ptr<Plan> plan); //execute just a few iterations
    and extrapolate the result
236     void verifyParameter(const int *size, const int* perm, const int* outerSizeA, const int*
    outerSizeB, const int dim) const;
237     void getBestParallelismStrategy ( std::vector<int> &bestParallelismStrategy ) const;
238     void getBestLoopOrder( std::vector<int> &loopOrder ) const; //innermost loop idx is stored at
    dim_-1
239     void getLoopOrders(std::vector<std::vector<int> > &loopOrders) const;
240     void getParallelismStrategies(std::vector<std::vector<int> > &parallelismStrategies) const;
241     void getAllParallelismStrategies( std::list<int> &primeFactorsToMatch,
242                                       std::vector<int> &availableParallelismAtLoop,
243                                       std::vector<int> &achievedParallelismAtLoop,
244                                       std::vector<std::vector<int> > &parallelismStrategies) const;
245     void getAvailableParallelism( std::vector<int> &numTasksPerLoop ) const;
246     int getIncrement( int loopIdx ) const;
247     void executeEstimate(const Plan *plan) noexcept; // almost identical to execute, but it just
    executes few iterations and then extrapolates
248     double getTimeLimit() const;
249
250     const floatType* __restrict__ A_;
251     floatType* __restrict__ B_;
252     floatType alpha_;
253     floatType beta_;
254     int dim_;
255     std::vector<size_t> sizeA_;
256     std::vector<int> perm_;
257     std::vector<size_t> outerSizeA_;
258     std::vector<size_t> outerSizeB_;
259     std::vector<size_t> lda_;
260     std::vector<size_t> ldb_;
261     std::vector<int> threadIds_;
262     int numThreads_;
263     int selectedParallelStrategyId_;
264     int selectedLoopOrderId_;
265     bool conjA_;
266 #ifdef _OPENMP
267     omp_lock_t writelock;
268 #endif
269
270     std::shared_ptr<Plan> masterPlan_;
271     SelectionMethod selectionMethod_;
272     int maxAutotuningCandidates_;
273     static constexpr int blocking_micro_ = REGISTER_BITS / 8 / sizeof(floatType);
274     static constexpr int blocking_ = blocking_micro_ * 4;
275
276     static constexpr int infoLevel_ = 0; // determines which auxiliary messages should be printed
277 };
278
279
280 extern template class Transpose<float>;
281 extern template class Transpose<double>;
282 extern template class Transpose<FloatComplex>;
283 extern template class Transpose<DoubleComplex>;
284
285 }

```

7.7 utils.h

```

1
5 #pragma once
6
7 #include <list>
8 #include <vector>
9 #include <iostream>
10
11 #include "hptt_types.h"
12
13 namespace hptt {
14
15 template<typename floatType>
16 static floatType conj(floatType x){
17     return std::conj(x);
18 }
19 template<>
20 float conj(float x){
21     return x;

```



```

22 }
23 template<>
24 double conj(double x){
25     return x;
26 }
27
28 template<typename floatType>
29 static double getZeroThreshold();
30 template<>
31 double getZeroThreshold<double>() { return 1e-16;}
32 template<>
33 double getZeroThreshold<DoubleComplex>() { return 1e-16;}
34 template<>
35 double getZeroThreshold<float>() { return 1e-6;}
36 template<>
37 double getZeroThreshold<FloatComplex>() { return 1e-6;}
38
39 void trashCache(double *A, double *B, int n);
40
41 template<typename t>
42 int hasItem(const std::vector<t> &vec, t value)
43 {
44     return ( std::find(vec.begin(), vec.end(), value) != vec.end() );
45 }
46
47 template<typename t>
48 void printVector(const std::vector<t> &vec, const char* label){
49     std::cout << label << ": ";
50     for( auto a : vec )
51         std::cout << a << ", ";
52     std::cout << "\n";
53 }
54
55 template<typename t>
56 void printVector(const std::list<t> &vec, const char* label){
57     std::cout << label << ": ";
58     for( auto a : vec )
59         std::cout << a << ", ";
60     std::cout << "\n";
61 }
62
63
64 void getPrimeFactors( int n, std::list<int> &primeFactors );
65
66 template<typename t>
67 int findPos(t value, const std::vector<t> &array)
68 {
69     for(int i=0;i < array.size() ; ++i)
70         if( array[i] == value )
71             return i;
72     return -1;
73 }
74
75 int findPos(int value, const int *array, int n);
76
77 int factorial(int n);
78
79 void accountForRowMajor(const int *sizeA, const int *outerSizeA, const int *outerSizeB, const int *perm,
80                        int *tmpSizeA, int *tmpOuterSizeA, int *tmpouterSizeB, int *tmpPerm, const int dim, const
81                        bool useRowMajor);
82
83

```


Index

- addThreadId
 - hptt::Transpose< floatType >, [19](#)
- create_plan
 - hptt, [13](#)
- execute
 - hptt::Transpose< floatType >, [20](#)
- execute_expert
 - hptt::Transpose< floatType >, [20](#)
- hptt, [11](#)
 - create_plan, [13](#)
 - SelectionMethod, [13](#)
- hptt::Transpose< floatType >, [17](#)
 - addThreadId, [19](#)
 - execute, [20](#)
 - execute_expert, [20](#)
 - setInputPtr, [20](#)
 - setMaxAutotuningCandidates, [20](#)
 - setOutputPtr, [21](#)
 - Transpose, [18](#)
- include/compute_node.h, [23](#)
- include/hptt.h, [23](#)
- include/hptt_types.h, [25](#)
- include/macros.h, [25](#)
- include/plan.h, [26](#)
- include/transpose.h, [26](#)
- include/utils.h, [28](#)
- SelectionMethod
 - hptt, [13](#)
- setInputPtr
 - hptt::Transpose< floatType >, [20](#)
- setMaxAutotuningCandidates
 - hptt::Transpose< floatType >, [20](#)
- setOutputPtr
 - hptt::Transpose< floatType >, [21](#)
- Transpose
 - hptt::Transpose< floatType >, [18](#)