TensorOperationsXD.jl

Fast tensor operations using a convenient Einstein index notation.

Table of contents

Installation

Install with the package manager, `pkg> add TensorOperationsXD`.

Package features

- A macro `@tensor` for conveniently specifying tensor contractions and index permutations via Einstein's index notation convention. The index notation is analyzed at compile time.
- Ability to optimize pairwise contraction order using the `@tensoropt` macro. This optimization is performed at compile time, and the resulting contraction order is hard coded into the resulting expression. The similar macro `@tensoropt_verbose` provides more information on the optimization process.
- A function `ncon` (for network contractor) for contracting a group of tensors (a.k.a. a tensor network), as well as a corresponding `@ncon` macro that simplifies and optimizes this slightly. Unlike the previous macros, `ncon` and `@ncon` do not analyze the contractions at compile time, thus allowing them to deal with dynamic networks or index specifications.
- Support for any Julia Base array which qualifies as strided, i.e. such that its entries are layed out according to a regular pattern in memory. The only exception are `ReinterpretedArray` objects (implementation provided by Strided.jl, see below). Additionally, `Diagonal` objects whose underlying diagonal data is stored as a strided vector are supported. This facilitates tensor contractions where one of the operands is e.g. a diagonal matrix of singular values or eigenvalues, which are returned as a `Vector` by Julia's `eigen` or `svd` method.
- Support for `CuArray` objects if used together with CUDA.jl, by relying on (and thus providing a high level interface into) NVidia's cuTENSOR library.
- Implementation can easily be extended to other types, by overloading a small set of methods.
- Efficient implementation of a number of basic tensor operations (see below), by relying on Strided.jl and `gemm` from BLAS for contractions. The latter is optional but on by default, it can be controlled by a package wide setting via `enable_blas()` and `disable_blas()`. If BLAS is disabled or cannot be applied (e.g. non-matching or non-standard numerical types), Strided.jl is also used for the contraction.
- A package wide cache for storing temporary arrays that are generated when evaluating complex tensor expressions within the `@tensor` macro (based on the implementation of LRUCache). By default, the cache is allowed to use up to the minimum of either 1GB or 25% of the total memory.

Tensor operations

TensorOperationsXD.jl is centered around 3 basic tensor operations, i.e. primitives in which every more complicated tensor expression is deconstructed.

1. **addition:** Add a (possibly scaled version of) one array to another array, where the indices of the both arrays might appear in different orders. This operation combines normal array addition and index permutation. It includes as a special case copying one array into another with permuted indices.

   The actual implementation is provided by Strided.jl, which contains multithreaded implementations and cache-friendly blocking strategies for an optimal efficiency.

2. **trace or inner contraction:** Perform a trace/contraction over pairs of indices of an array, where the result is a lower-dimensional array. As before, the actual implementation is provided by Strided.jl.
3. **contraction:** Performs a general contraction of two tensors, where some indices of one array are paired with corresponding indices in a second array. This is typically handled by first permuting (a.k.a. transposing) and reshaping the two input arrays such that the contraction becomes equivalent to a matrix multiplication, which is then performed by the highly efficient `gemm` method from BLAS. The resulting array might need another reshape and index permutation to bring it in its final form. Alternatively, a native Julia implementation that does not require the additional transpositions (yet is typically slower) can be selected by using `disable_blas()`.

To do list

- Make it easier to check contraction order and to splice in runtime information, or optimize based on memory footprint or other custom cost functions.

Index notation with macros »