Towards a High-Performance Tensor Algebra Package for Accelerators
Marc Baboulin, Veselin Dobrev, Jack Dongarra, Christopher Earl, Joel Falcou, Azzam Haidar, Ian Karlin, Tzanio Kolev, Ian Masliah, Stanimire Tomov

To cite this version:

HAL Id: hal-01231234
https://hal.archives-ouvertes.fr/hal-01231234
Submitted on 19 Nov 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Towards a High-Performance Tensor Algebra Package for Accelerators

Abstract
Numerous important applications, e.g., high-order FEM simulations, can be expressed through tensors. Examples are computation of FE matrices and SpMV products expressed as generalized tensor contractions. Computations by the first index can often be represented as tensor index reordering plus gemm, which is a key factor to achieve high-performance. We present ongoing work on the design of a high-performance package in MAGMA for Tensor algebra that includes techniques to organize tensor contractions, data storage, and parametrization related to batched execution of smaller number of tensor contractions. We apply auto-tuning and code generation techniques to provide an architecture-aware, user-friendly interface.

Motivation
Numerous important applications can be expressed through tensors:

- High-order FEM simulations
- Signal Processing
- Numerical Linear Algebra
- Numerical Analysis
- A 4-dimensional tensor contraction

The goal is to design a:

- High-performance package for Tensor algebra
- Built-in architecture awareness (GPU, Xeon Phi, multicore)
- User-friendly interface

Example cases

Tensor operations in high-order FEM
Consider the FE mass matrix \( M \) for an element/zone \( E \) with weight \( o \) as a 2-dimensional tensor:

\[
(M_{ij}) = \sum_{k=1}^{n} m_k (o_k(x_i) o_k(x_j))/|J_k|,
\]

where:

- \((M_{ij})\) is the number of FE degrees of freedom (dof)
- \(o_k\) is the number of quadrature points
- \((o_k(x_i))\) are the FE basis functions on the reference element
- \(|J_k|\) is the determinant of the element transformation
- \((o_k), (x_k), (l_k)\) are the points and weights of the quadrature rule

Take the \( x \times w \) matrix

\[
B_{kl} = p_{kl}, \quad (B_{kl}) = m_k l_k = \sum_{n=1}^{N} m_k l_k n_k.
\]

Then,

\[
(M_{ij}) = \sum_{k=1}^{n} B_{kl} (B_{kl}) B_{kl} ,
\]

or omitting the \( E \) subscript

\[
M = B^T DB.
\]

Using FE of order \( p \), we have \( w = Op(p) \) and \( m = Op(2p) \), so \( B \) is dense \( Op(p) \times Op(2p) \) matrix.

If the FE basis and the quadrature rule have tensor product structure, we can decompose dofs and quadrature point indices in logical coordinate axes

\[
B_{kl} = l_1 l_2 \ldots l_{d-1} k_1 k_2 \ldots k_d,
\]

so \( M \) can be viewed as 2-dimensional tensor

\[
M_{i_1 j_1 \ldots i_{d-1} j_{d-1}, k_1 \ldots k_d}.
\]

Summary of kernels needed:

- Assembly of \( M \), referred as equations (1) & (2) below
- Evaluations of \( M \) times \( V \), referred as equations (3) & (4) below


tensor contractions are transformed through reshapes to batched LA operations, many of which available in MAGMA (including LU, QR, Cholesky, GEMM, GEMV, TRSM, SYRK).

User-friendly interface
To provide various interfaces, including one using C++11. Top level design to provide features similar to the mshadow library: https://github.com/dnic/mshadow

Conclusions and Future directions
- High-performance package on Tensor Algebra has the potential for high-impact on a number of important applications
- Multidisciplinary effort
- Current results show promising performance, where various components will be leveraged from autotuning MAGMA Batched linear algebra kernels, and BLAST from LLNL
- This is an ongoing work

Code Generation
C++11 features will be used as much as possible. Additional needs will be handled by defining a domain specific embedded language (DSEL). This technique is used in C++ to take advantage of DSL features while using the optimizations provided by a standard compiler: it will handle the generation of versions (index reordering, next) to be empirically evaluated and be part of the autotuning framework.

Numerical linear algebra:

- A 4-dimensional tensor contraction
- rank-k update on matrices in tile format (k can be small, e.g., sub-vector/warp size)
- Must determine (in software) if possible to do it through batched GEMM kernels

Lagrangian Hydrodynamics in the BLAST code
On semi-discrete level our method can be written as

\[
\frac{\partial v}{\partial t} + \mathbf{F}_v = 0,
\]

\[
\frac{\partial e}{\partial t} + \mathbf{F}_e = \frac{1}{\rho} \mathbf{S}_{\text{strain}}
\]

\[
\text{Equation of Motion:} \quad \rho \frac{\partial v}{\partial t} = \mathbf{P}
\]

where \( v, e, \rho \) are the unknown velocity, specific internal energy, and mass density, respectively; \( \mathbf{F}_v, \mathbf{F}_e, \) and \( \mathbf{S}_{\text{strain}} \) are independent of time velocity and energy mass matrices; \( F \) if the generalized corner force matrix depending on \( v, a, e, \) which needs to be evaluated at every time step.


Acknowledgments
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

This work was supported by the National Science Foundation under Grant ACI-1339822, the Department of Energy, and NVIDIA.

User-friendly interface
To provide various interfaces, including one using C++11. Top level design to provide features similar to the mshadow library: https://github.com/dnic/mshadow

Approach and results

User-friendly interface
To provide various interfaces, including one using C++11. Top level design to provide features similar to the mshadow library: https://github.com/dnic/mshadow

Index reordering/reshape
If we store tensors as column-wise 1D arrays,

\[
B_{kl} = \sum_{m=1}^{M} b_{km} l_m,
\]

using a vector of size \( w \) without changing the storage. We can define

\[
\text{Reshape}(T)_{kl} = B_{kl} l_{m(k,l,m,k)} = \sum_{m=1}^{M} b_{km} l_k l_m.
\]

Contractions can be implemented as a sequence of pairwise contractions. There is enough complexity here to search for something better: code generation, index reordering, and autotuning will be used, e.g., contractions (3a) - (4f) can be implemented as tensor index-reordering plus gemm \( A B \to C \).

Batched LA

Tensor contractions are transformed through reshapes to batched LA operations, many of which available in MAGMA (including LU, QR, Cholesky, GEMM, GEMV, TRSM, SYRK).

Code Generation
C++11 features will be used as much as possible. Additional needs will be handled by defining a domain specific embedded language (DSEL). This technique is used in C++ to take advantage of DSL features while using the optimizations provided by a standard compiler: it will handle the generation of versions (index reordering, next) to be empirically evaluated and be part of the autotuning framework.