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. Contraction 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 parameterization related to batched execution of small 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

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 \( \mu \), as a 2-dimensional tensor:

\[
(M_{ij}) = \sum_{a=1}^{n_e} \mu a (u_{ij}^a) (v_{ij}^a) / h_{ij}^a
\]

\( i, j = 1, \ldots, m \), where

- \( n_e \) is the number of FE degrees of freedom (dofs)
- \( u_{ij}^a \) are the FE basis functions on the reference element
- \( v_{ij}^a \) is the determinant of the element transformation
- \( h_{ij}^a \) are the points and weights of the quadrature rule

Take the \( ax \) matrix \( B_{ax} = (v_{ax}) \), and \((y_{ax}) = \mu a (u_{ax}) (v_{ax})\). Then,

\[
(M_{ij}) = \sum_{a=1}^{n_e} \mu a (B_{ax}) (y_{ax})
\]

Using FE or order \( p \), we have \( ax = Op(q) \), and \( ay = Op(p) \), so \( B \) is dense \( Op(q) \times Op(p) \) matrix.

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

\( i_1 \ldots i_m \rightarrow i_1 \ldots i_d \times 1 \ldots 1 \times k_1 \ldots k_q \)

so \( M \) can be viewed as 2-dimensional tensor \( M_{i_1 \ldots i_d, j_1 \ldots j_d} \).

Summary of kernels needed:
- Assembly of \( M \), referred as equations (1) & (2) below
- Evaluations of \( M \) times \( v \), referred as equations (3) & (4) below

Approach and results

User-friendly interface
To provide various interfaces, including one using C++11.

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.

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.

Autotuning
We are developing fixed-size gemm kernels for GPUs, Xeon Phi, and multicore (see Figure on right for a single core intel Xeon E5-2620 and K40) through an autotuning framework. A number of generic versions are developed and parametrized for performance. The parameters are autotuned (empirically) to find “best” kernels for specific size.

Batched LA
Tensor contractions are transformed through reshapes to batched LA operations, many of which are available in MAGMA(1):

- LU, QR, Cholesky, GEMM, GEMV, TRSM, SYRK

Lagrangian Hydrodynamics in the BLAST code

On semi-discrete level our method can be written as:

\begin{align*}
\text{Momentum Conservation:} & \quad \frac{\partial v}{\partial t} = -M_{v} v + \mathbf{F} \\
\text{Energy Conservation:} & \quad \frac{\partial \rho}{\partial t} = \nabla \cdot \mathbf{M}_{\rho} v \\
\text{Equation of Motion:} & \quad \frac{\partial v}{\partial t} = \nabla \cdot \mathbf{M}_{\rho} v
\end{align*}

where \( v \), \( \rho \), and \( \mathbf{F} \) are the unknown velocity, specific internal energy, and grid position, respectively; \( M_{v} \) and \( M_{\rho} \) are independent of time velocity and energy mass matrices; and \( \mathbf{F} \) is the generalized corner force matrix depending on \( v \) and \( \rho \), which needs to be evaluated at every time step.

(1) V. Dobrev, T. Kolev, I. Masliah, High-order finite element formulations for Lagrangian hydrodynamics. SAM J. Sci. Comp. 34(2), 806-841 (38 pages)