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
- Graph Analysis
- Neuroscience and more

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 $M_E$ for an element/zone $E$ with weight $p$, as a 2-dimensional tensor:

$$ (M_E)_{ij} = \sum_k m_k (v_i(v_k) v_j(v_k))_{ij} $$

where $i, j = 1, \ldots, m$, $m$ is the number of quadrature points.

Numerical linear algebra:
- A 4-dimensional tensor contraction
- rank-k update on matrices in tile format (k can be small, e.g., 8)
- 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

- Momentum Conservation:
  $$ \frac{\partial v}{\partial t} = -M_v F $$
- Energy Conservation:
  $$ \frac{\partial e}{\partial t} = -M_e F $$
- Equation of Motion:
  $$ \frac{\partial M}{\partial t} = M_v F $$

where $v, e, M$ are the unknown variables, specific internal energy, and grid position, respectively; $M_v$ and $M_e$ are independent of time velocity and energy mass matrices; and $F$ is the generalized corner force matrix depending on $(v, e, M)$, which needs to be evaluated at every time step.

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.

Index reordering/reshape
If we store tensors as column-wise 1D arrays, $A_{i1,\ldots,i_r;k1,\cdots,k_d}$, we can define

$$ A_{i1,\ldots,i_r;k1,\cdots,k_d} = M_{i1,\ldots,i_r;k1,\cdots,k_d} $$

where $i_1, \ldots, i_r, k_1, \ldots, k_d$. So $M$ is dense $O(p \times d)$.

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