Tensor Contractions using Optimized Batch GEMM Routines

Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Abstract

We present a high performance solution for tensor contractions using CUDA. In particular, we consider large scale tensor-formulated high-order finite element method (FEM) simulations, which can be represented as a sequence of batch GEMM operations. We show that a highly tuned batch GEMM kernel can achieve significant speedups against cuBLAS. Thanks to an extensive tuning process, we are able to maintain a performance advantage for each size of interest. Further performance gains are achieved by fusing the batch GEMM operation into one GPU kernel, which leads to an optimal data reuse.

Motivation

Numerous important applications can be expressed through tensors:

- High-order FEM simulations
- Signal Processing
- Numerical Linear Algebra
- Numerical Analysis
- Data Mining
- Deep Learning
- Graph Analysis
- Neuroscience and more

Accelerating High-order FEM

Lagrangian Hydrodynamics in the BLAST code[1]

On semi-discrete level our method can written as

$$\textbf{Momentum Conservation:} \quad \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbf{M}_\mathbf{v}^{-1}\mathbf{F}\cdot\mathbf{1}$$

$$\textbf{Energy Conservation:} \quad \frac{\mathrm{d}\mathbf{e}}{\mathrm{d}t} = \mathbf{M}_\mathbf{e}^{-1}\mathbf{F}^\mathrm{T}\cdot\mathbf{v}$$

$$\textbf{Equation of Motion:} \quad \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}$$

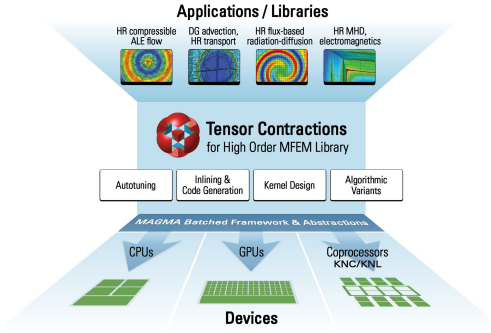
where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M}_\mathbf{v}$ and $\mathbf{M}_\mathbf{e}$ are independent of time velocity and energy mass matrices; and $\mathbf{F}$ is the generalized corner force matrix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

- Expressed in terms of tensor contractions [2];
- Contractions can be implemented as sequence of pairwise contractions (slow);
- Code-generation, index-reordering, and auto-tuning are used to cast computations as Batched GEMMs:

$C_{i1,i2,i3} = \sum A_{i,j1} B_{i,j2,j3}$ is transformed autom. to

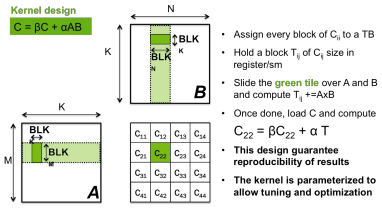
$C^{d1x(d2,d3)} = A^{T}$

$B^{d1x(d2,d3)}$



Code Autogeneration and Kernel Design

- CUDA C++ Templates
- Problem size is a template parameter to maximize loop unrolling
- Inline GPU functions
- 2D thread configuration
- Tunable concurrency on the multiprocessor level
- Part of the MAGMA library



- Assign every block of C_{ij} to a TB
- Hold a block T_{ij} of C_{ij} size in register/sm
- Slide the green tile over A and B and compute T_{ij} +=AxB
- Once done, load C and compute $C_{22} = \beta C_{22} + \alpha T$
- **This design guarantee reproducibility of results**
- **The kernel is parameterized to allow tuning and optimization**

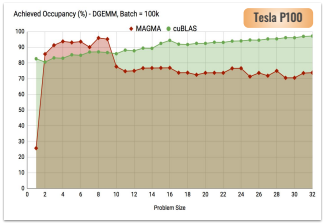
Methodology using Standard GEMM

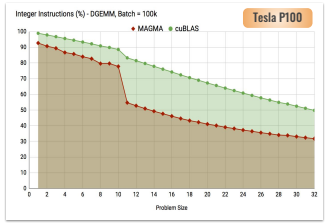
Batch GEMM Design and Optimization

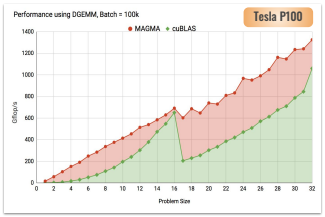
We want to compute batched (over the finite elements) sequences of matrix-matrix (GEMM) multiplications of the form: $C = B^T D .* (B A B^T) B$

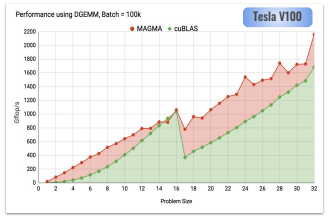
The sizes of interest are up to 32. The operation can be performed using 4 GEMM operations and an elementwise multiplication with the matrix D (currently ignored). The designed GEMM kernels use CUDA C++ templates. This enables a unified code base that can be explicitly instantiated for every small problem size [2,3,4].

- **Shared Memory Blocking and Double Buffering**
  The MAGMA kernel caches the input submatrices from A and B in shared memory, while all computations for C are accumulated in registers. We performed an extensive set of auto-tuning and performance counter analysis to optimize and improve the implementation. Prefetching is also used to load the next blocks of A and B and is controlled by a tunable parameter.
- **Analysis of Hardware Counters**
  We performed a detailed performance study based on the collection and analysis of hardware counters. Counter readings were taken using performance tools (Nvidia CUPTI and PAPI CUDA component). We added the GEMM sizes (M, N, K) to the template parameters such a way to use a unified code base to produce a fully unrolled and optimized implementation for any of these very small sizes.
- **Performance Speedups**
  MAGMA is 1.06x-to-25.9x faster than cuBLAS on the P100 GPU, and is up to 13.6x faster than cuBLAS on the V100 GPU.







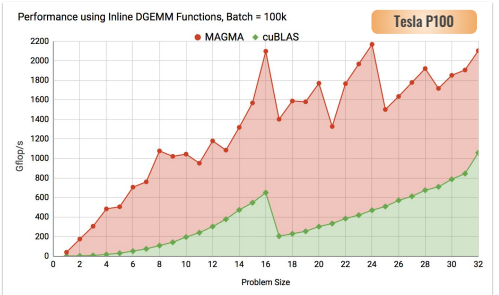


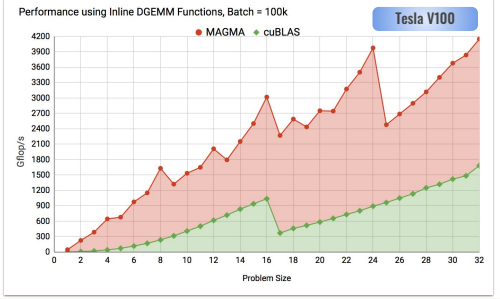
Methodology Using Inline GEMM Functions

Inline GEMM Functions for Optimal Data Reuse

Since the individual problem size is extremely small, the sequence of GEMM operations can be fused into a single GPU kernel to maximize data reuse. In order to perform in a portable reusable way, we have applied the following design choices.

- **Demote GEMM from a Kernel to an Inline Device Function**
  The core computational code of the MAGMA kernel has been demoted into an inline device functions that is callable from within a higher level kernel. In general, four functions are provided to support the different transposition modes of the GEMM operations. The device functions assume that all matrices are stored in shared memory. They perform no global memory transactions at all.
- **Read and Write Device Functions**
  Since the GEMM device functions do not interact with the global memory, two additional functions are provided to read from global memory to shared memory, and to write from shared memory to global memory. The user is responsible for calling these functions at the beginning and the end of the CUDA kernel.
- **Performance Tuning**
  All device functions are written based on a generic 2D thread configuration that is oblivious to the actual problem size. However, the thread configuration affects the required shared memory space, which is a requirement for a fully unrolled code. The shown results are based on preliminary performance tuning experiment. An extensive autotuning effort is required for best results.
  The developed framework also supports non square problems, but it requires a more sophisticated tuning experiments.
- **Performance Speedups**
  MAGMA is now 1.99x-to-79.8x faster than cuBLAS on the P100 GPU. It is also 2.5x-to-37x faster than cuBLAS on the V100 GPU.





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

Acknowledgement This work was supported by the Exascale Computing Project, a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work was also partially supported by the National Science Foundation under Grant OAC-1740250 and NVIDIA.

REFERENCES: [1] V. Dobrev, T.Kolev, R.Rieben. *High order curvilinear finite element methods for Lagrangian hydrodynamics*. SIAM J.Sci.Comp.34(5), B606–B641, 2012.
[2] A. Abdelfattah, M. Baboulin, V. Dobrev, J. Dongarra, C. Earl, J. Falcou, A. Haidar, I. Karlin, T. Kolev, I. Masliah, S. Tomov, *High-Performance Tensor Contractions for GPUs*, ICCS'16, San Diego, CA, June 2016.
[3] A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra, *Performance, Design, and Autotuning of Batched GEMM for GPUs*, ISC High Performance 2016, Frankfurt, Germany, June 2016.
[4] I. Masliah, A. Abdelfattah, A. Haidar, S. Tomov, M. Baboulin, J. Falcou, J. Dongarra High-performance matrix-matrix multiplications of very small matrices, Euro-Par 2016, Grenoble, France, August 22-26, 2016.