Accelerating Tensor Contractions for High-Order FEM on CPUs, GPUs, and KNLs

Azzam Haidar, Ahmad Abdelfattah, Veselin Dobrev, Ian Karlin, Tzanio Kolev, Stanimire Tomov, and Jack Dongarra

Lawrence Livermore National Laboratory · ICL · The University of Tennessee Knoxville

Abstract

High-performance is difficult to obtain using existing libraries, especially for many independent computations where each computation is very small. However, using our framework to batch computation plus application-specifics, we demonstrate close to peak performance results. In particular, to accelerate large scale tensor-formulated high-order finite element method (FEM) simulations, which is the main focus and motivation for this work, we represent contractions as tensor index reordering plus matrix-matrix multiplications (GEMMs). This is a key factor to achieve algorithmically many-fold acceleration (vs. not using it) due to possible reuse of data loaded in fast memory.

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 be written as

$$\textbf{Momentum Conservation:}\quad \frac{d\mathbf{v}}{dt} = -\mathbf{M_v^{-1}F}\cdot\mathbf{1}$$

$$\textbf{Energy Conservation:}\quad \frac{d\mathbf{e}}{dt} = \mathbf{M_e^{-1}F^T}\cdot\mathbf{v}$$

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

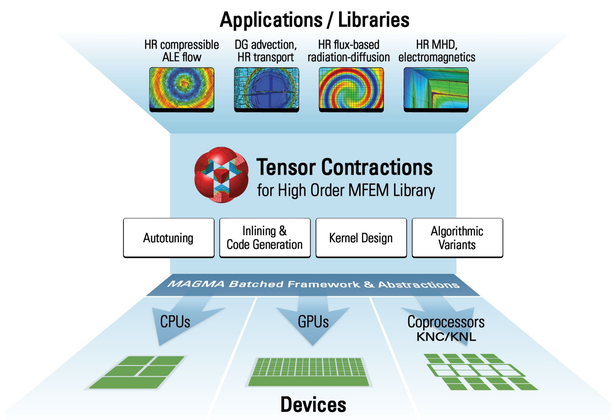
where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M_v}$ and $\mathbf{M_e}$ are independent of time velocity and energy mass matrices; and $\mathbf{F}$ is the generalized corner force matrix depending on $(\mathbf{v}, \mathbf{e}, \mathbf{x})$ 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:

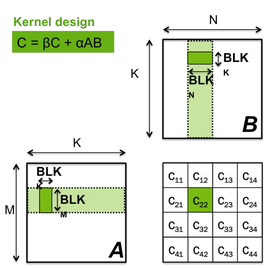
$$C_{i1,i2,i3} = \sum_k A_{k,i1} B_{k,i2,i3}$$

is transformed autom. to

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



Code Autogeneration and Kernel Design



- 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**

GPU

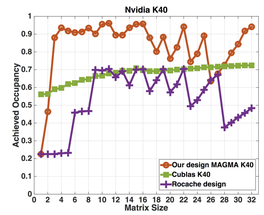
Methodology, Design, and Optimization GPU

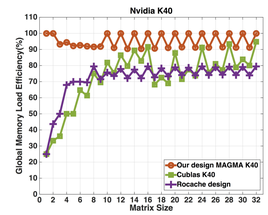
We use a hierarchical blocking model of both communications and computations. We designed CUDA C++ templates to enable unified code base for all the small sizes [2,3,4].

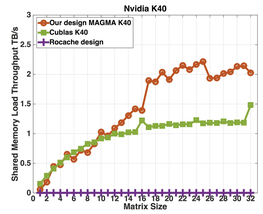
- **A Cache-based Rocache Approach**
  Unlike multi-core CPUs, the L1 cache (per SM) is not intended for global memory accesses, which are cached only in the L2. The L2 cache is shared among all SMs, which makes it difficult to use cache-based optimizations. However, a modern Kepler GPU has a 48 KB per SM of a read-only cache (rocache), which can be used for global memory reads. A possible implementation, is to read the matrices A and B through the read-only cache.

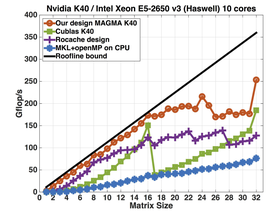
- **A Shared Memory based Approach**
  Another approach is to use shared memory for data reuse rather than rocache. We refer to this implementation as the MAGMA kernel. We performed an extensive set of auto-tuning and performance counter analysis to optimize and improve this implementation. The matrices A and B are loaded by block into the shared memory, and the corresponding block of the matrix C is held into registers. Prefetching can also be 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's CUPTI and PAPI CUDA component). We added the 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.









Xeon PHI / multicore CPU

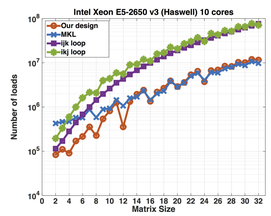
Methodology, Design, and Optimization CPU, PHI

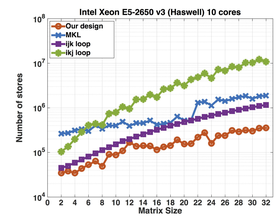
We conducted an extensive study over the performance counters using the PAPI tools to conclude that in order to achieve an efficient execution, we need to maximize the occupancy and minimize the data traffic while respecting the underlying hierarchical memory design. Unfortunately, today's compilers cannot introduce highly sophisticated cache/register based loop transformations [4].

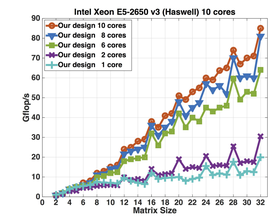
- **Data Access Optimizations and Loop Transformation Techniques**
  we propose to order the iterations of the nested loops in such a way that we increase locality and expose more parallelism for vectorization. Hence, loop unrolling, loop peeling, and loop interchange can be useful techniques. we propose to unroll the two inner-most loops so that the accesses to matrix B are independent from the loop order, which also allows us to reorder the computations for continuous access and improved vectorization. This technique enables us to prefetch and hold some of the data of B into the SIMD registers.

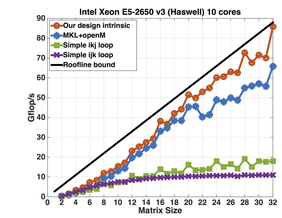
- **Register Data Reuse and Locality**
  We focus on register blocking to increase the performance. Our study concludes that the register reuse ends up being the key factor for performance. The idea is that when data is loaded into SIMD register, it will be reused as much as possible before its replacement by new data. The amount of data that can be kept into registers becomes an important tuning parameter. This reduces the number of load, store, and total instructions from $O(n^2)$ to $O(n)$, compared to a classical ijk or ikj implementation as depicted in figures.

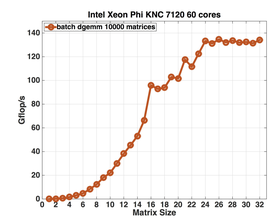
- **Effect of the Multi-threading**
  Operating on matrices of very small sizes is memory-bound computation and thus, increasing the number of CPU cores may not always increase the performance since the performance will be limited by the bandwidth which can be saturated by a few cores. We performed a set of experiments towards clarifying this behavior and illustrate our findings in figures. As shown, the notion of perfect speed-up does not exist for a memory-bound algorithm, and adding more cores increases the performance slightly.

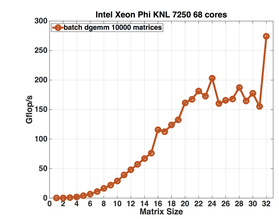












Conclusions and Future directions

- 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 work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL release number LLNL-POST-701583
ICL's work on this material was supported by the National Science Foundation under Grant ACI-1339822, the Department of Energy, and NVIDIA.

Smoky Mountains Computational Sciences and Engineering Conference

Gatlinburg, Tennessee, Aug 30- Sept 1, 2016
http://computing.ornl.gov/workshops/SMC16/

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.
[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.