BLAST Motivated Small Dense Linear Algebra Library Comparison

Pate Motter
Department of Computer Science
University of Colorado
Email: pate.motter@colorado.edu

Ian Karlin
Livermore Computing
Lawrence Livermore National Laboratory
Email: karlin1@llnl.gov

Chris Earl
Lawrence Livermore National Laboratory
Email: earl2@llnl.gov

***Abstract*—The goal of LLNL's SHOCX project is to develop a new multiphysics coupled radiation-hydro code based on arbitrary-order finite elements. BLAST is the arbitrary Lagrangian-Eulerian (ALE) portion of this project and its finite-element formulation requires both global sparse and local dense matrix solves. The local dense matrix solves and sparse matrix assembly perform numerous calculations on small, dense matrices. Currently these operations are handled by MFEM, but using third party libraries such as Eigen, Armadillo, and Blaze might be more productive and faster. Using a benchmark we created to mimic the most expensive portions of BLAST, we explore the performance and maintainability of these libraries compared to MFEM and a version of MFEM optimized via compile-time templates. Our results show that for our benchmarks Armadillo, Blaze, and templated MFEM all show promising results. Eigen was initially promising with its available features, but was not performant enough to use it in BLAST.**

I. BLAST

BLAST is a high-order finite element hydrodynamics research code designed for use on exascale architectures. ...

II. Motivation

BLAST moves the same amount of data regardless of the order, this allows for increased computational intensity for the same amount of data transferred as well as higher accuracy. By adding more degrees of freedom we are able to represent more details within a given zone. BLAST contains many small, dense linear algebra operations such as matrix-matrix multiplies, transposes, eigenvector calculations, and others. These matrices vary in size depending on the order and dimension of the problem, but are typically $< 20 \times 20$. Because there are so many repeated linear algebra function calls, optimizing their performance is a critical component for reducing BLAST's overall run-time.

Many available third-party libraries exist which claim for faster performance and ease-of-use compared to those found in our current linear algebra routines provided my MFEM. MFEM is an LLNL library focusing on scalable finite element methods. Since a large portion of our run-time is spent manipulating and operating on these small matrices, any performance gains would be noticeable especially at higher-order and/or with large meshes.

III. Profiling BLAST

In order to determine what types of linear algebra functions were taking the most of amount of time and resources we needed to profile our current implementation. We started by identifying potential functions/kernels which consumed a significant amount of overall run-time using gprof and HPC-Toolkit. By profiling both the Sedov and Noh test problems we were able to estimate both a lower and upper bound on the intensity of the problem. The Sedov problem expands from a singular point causing many of the mesh's zones to remain undisturbed. These undisturbed zones are not actively computing the interactions with its neighbors and therefore contribute very little to the overall run-time. The Noh example however, starts as a spherical volume which then collapses upon itself. Since many zones are moving from the first time step there are more 'active' zones in the Noh problem and this can be viewed as an upper bound of sorts for the complexity of a given mesh. Taking the profile of both of these problems at various dimensions and orders provided us with a clear look at the differences and similarities between the two problems and which functions became dominant.

IV. Libraries

There are many libraries which promise high-performance for dense linear algebra problems. For BLAST many of these matrices are defined at compile time and are computed using LLNL's MFEM library. Currently there is little literature available which compares the various linear algebra libraries in a manner consistent with BLAST's needs. We started by evaluating the wide variety of available libraries to select a set which showed some promise in meeting our needs. These were then benchmarked to determine which showed the most promising results.

V. Testing and Results

The libraries which met our minimum requirements were run through synthetic benchmarks simulating the types of linear algebra functions found in BLAST. Performance was compared to the results of using our current BLAST objects and functions which come from the MFEM library. These tests were used to determine a rough estimate of performance without requiring a full port of BLAST to the target library. Libraries which showed the most potential were then ported

into BLAST to have a more direct comparison to our existing code.

VI. Results

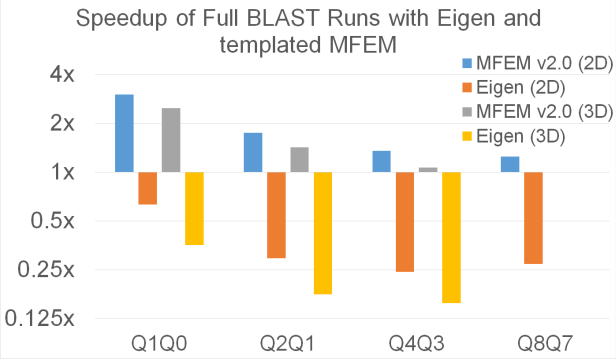

Fig. 1.