Performance Study and Optimization of FleCSALE using Tabular Equation of State

Dana Akhmetova, Sumathi Lakshmiranganatha, Diptajyoti Mukherjee, Frederick Ouellet, Patrick Payne, & Nicholas Stegmeier

Co-Design Summer School 2018

Introduction

Background

- Hydrodynamics codes require material specific thermodynamic data, usually obtained with an analytic or tabular equation of state (EOS)
- EOSPAC is a library that provides access to and interpolation routines for material data tables
- FleCSALE is a continuum dynamics code that is built on the FleCSI framework and uses EOSPAC
- FleCSI is a compile-time configurable framework designed to support multi-physics application development

Objective

- Optimize FleCSALE in the context of EOSPAC by investigating: EOSPAC library optimizations, hybrid programming for FleCSALE, and FleCSI sparse data optimizations
- All scaling simulations were run on Broadwell CPUs to a simulation time of 1.5s

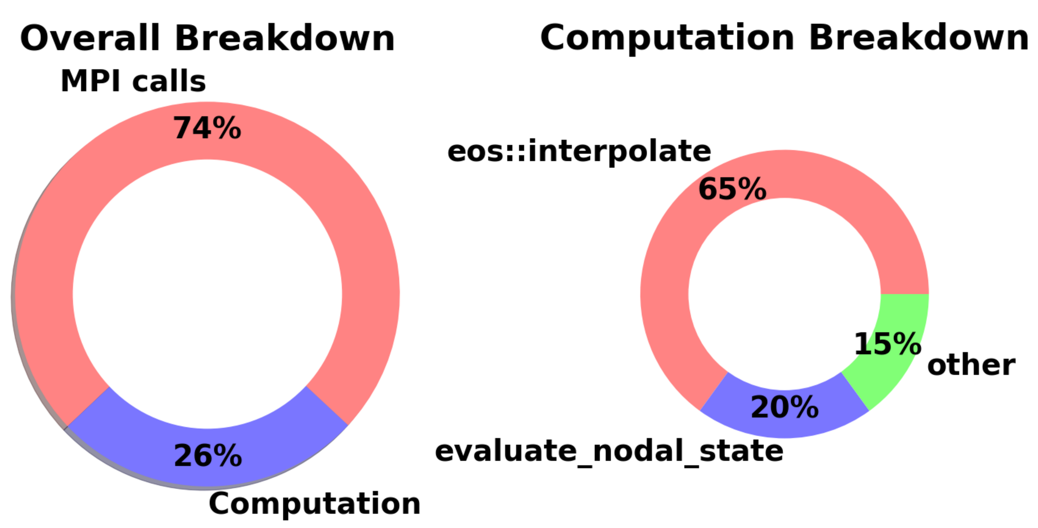

Figure 1: Initial communication and computation breakdown of FleCSALE

Strategies for EOSPAC

Optimizing the EOSPAC Interface

- SESAME data tables are inverted at initialization and stored
- Interpolations performed using groups of common material cells
- Sorted arrays passed to EOSPAC for interpolation

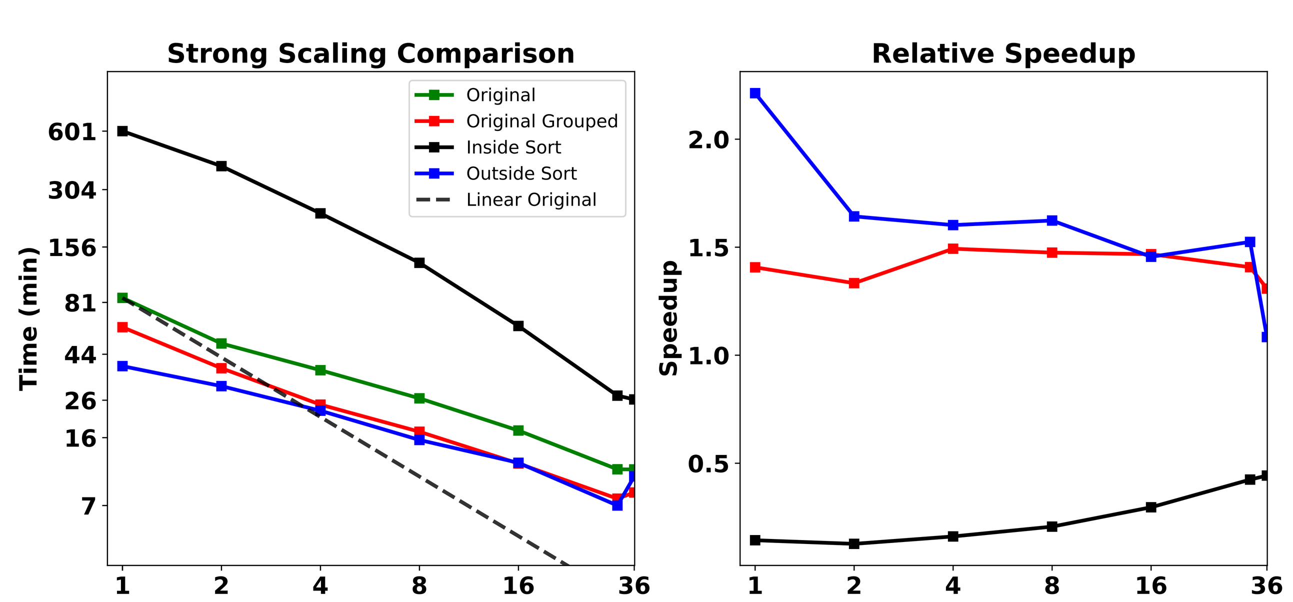

Figure 2: Timing and performance data for FlecSALE runs with various integration techniques. Inversion/grouping speeds up the code by a factor of 1.5, adding sorting increases it to a factor of 1.6-1.7

GPU Porting of EOSPAC Interpolation

- Interpolation algorithms are single threaded but easily parallelizable
- Broadwell CPU and a Tesla P100 GPU used for performance runs

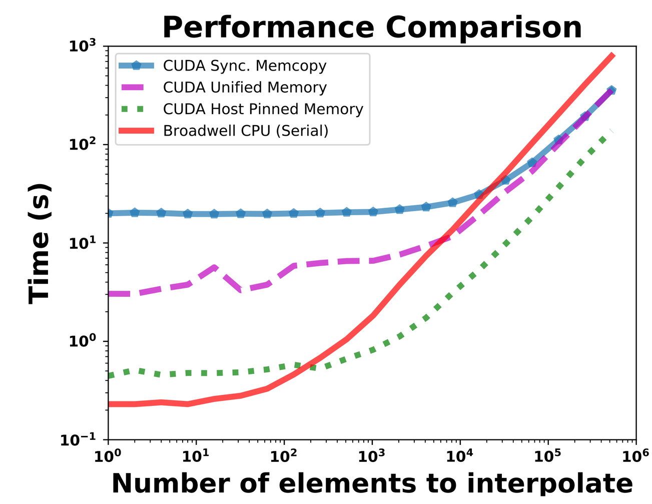

Figure 3: Timing data for 10,000 interpolation calls to EOSPAC

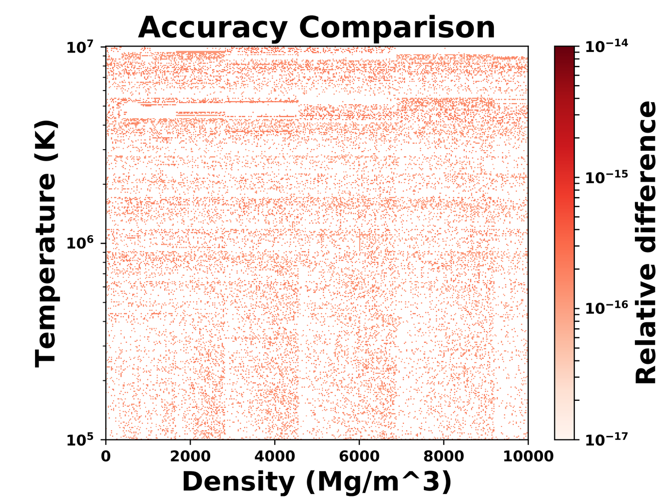

Figure 4: Relative difference between the serial and CUDA implementations

Machine Learning for Equation of State

- Machine learning was used to replace EOSPAC calls in FleCSALE
- DLIB C++ ML library offers kernel ridge regression (KRR) and random forest (RF) regression models
- ML memory usage highly dependent on model

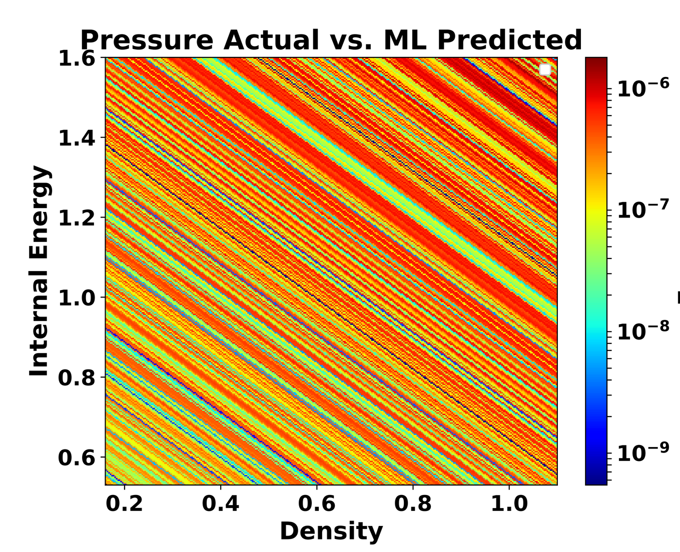

Figure 5: Absolute error of random forest model compared to EOSPAC

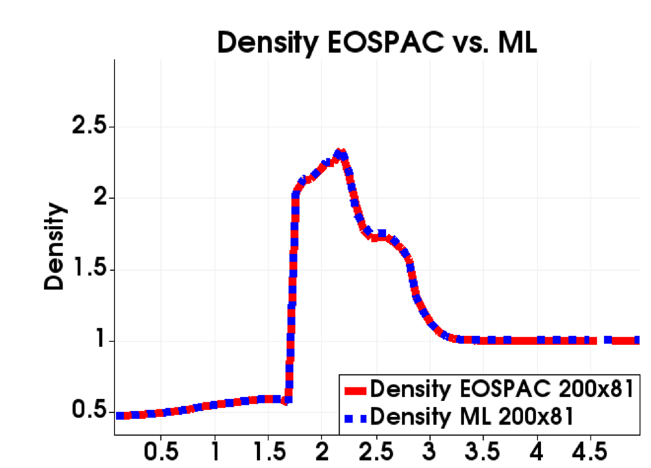

Figure 6: Integration of KRR ML with FleCSALE

OpenMP in FleCSALE

- OpenMP was used in outer for-loop work-sharing constructs in FleCSALE tasks
- This was tested on Intel Haswell E5-2698 v3 (2 sockets, 16 cores per socket, 2 threads per core, 64 logical cores in total) with a 200x81 mesh

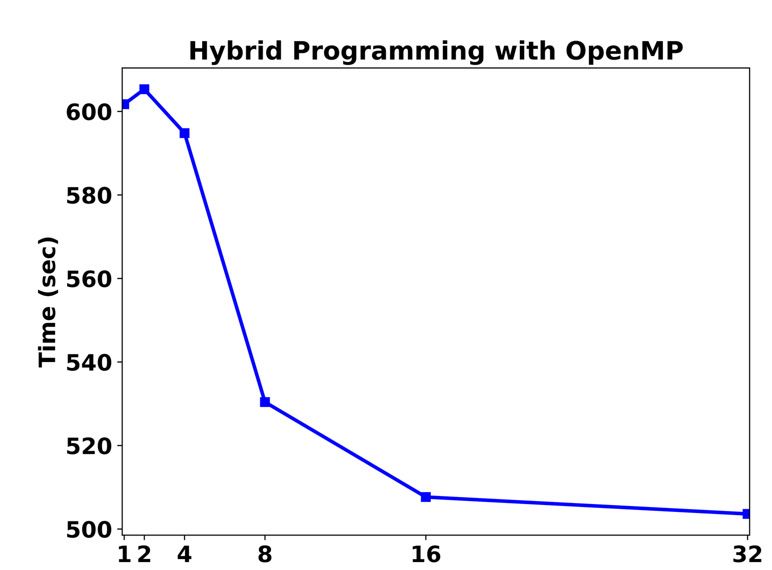

Figure 7: OpenMP performance results in FleCSALE

Sparse Data Optimization for FleCSI

- `MPI_Win_create` calls was the most expensive ~ 38% of the MPI communication time
- Unnecessary data copies and shared window creation was eliminated
- Shared window was created only at the beginning of the execution
- `MPI_Datatype` was used for shared ghost cells to treat as a single message
- Overall performance was improved by ~ 80%

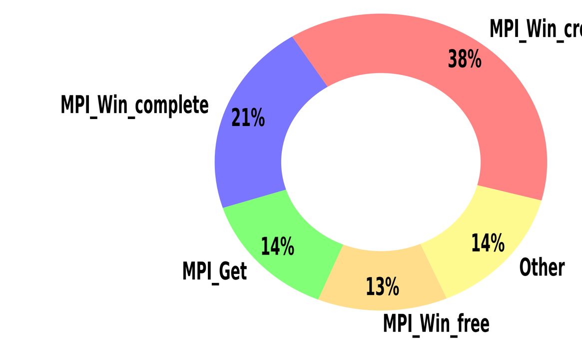

Figure 8: MPI calls breakdown

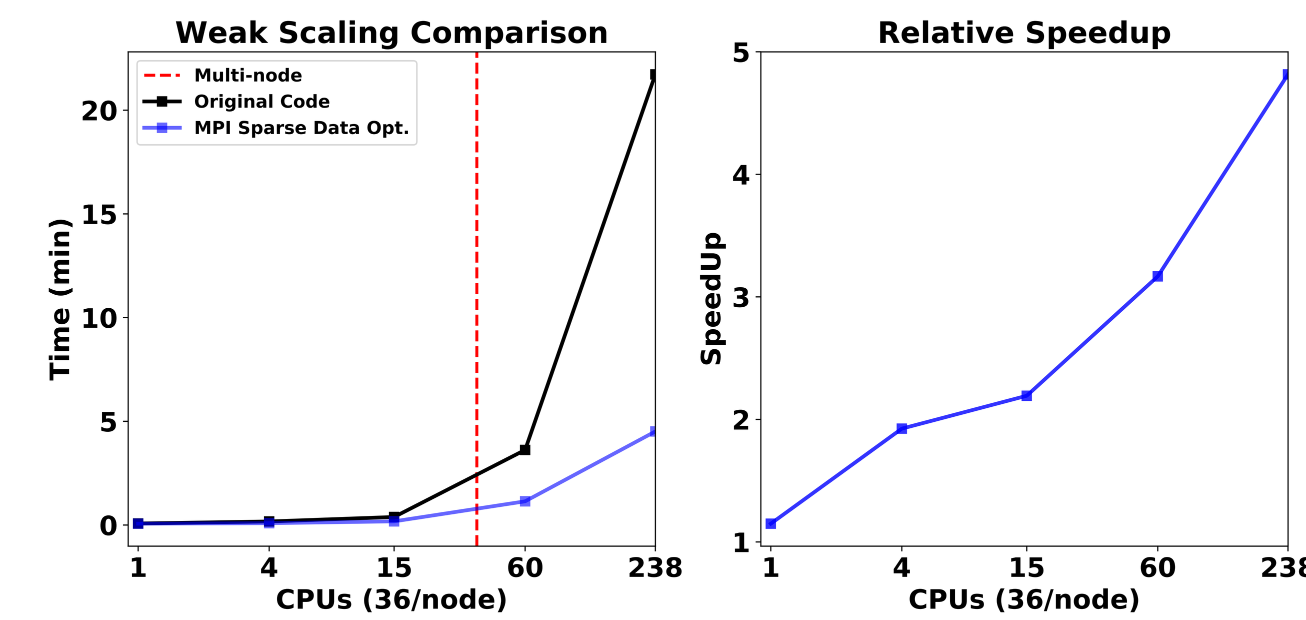

Figure 9: Weak scaling graph for FleCSALE with relative speedup

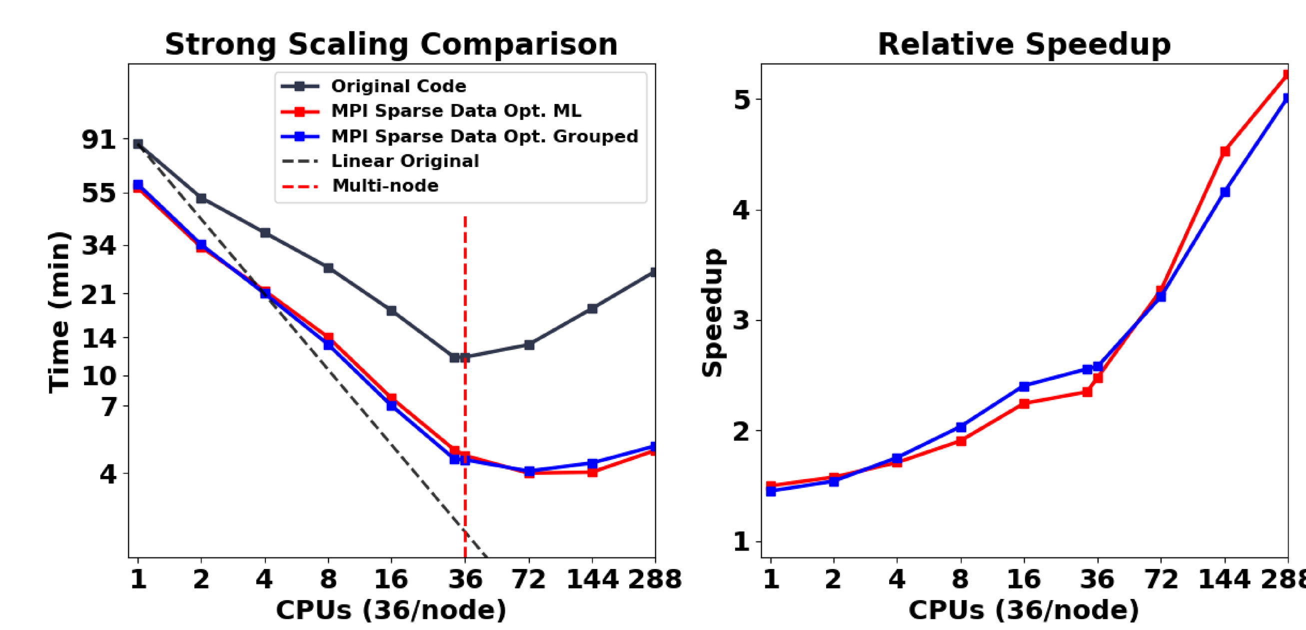

Figure 10: Strong scaling graph for FleCSALE with relative speedup

Summary and Future Work

- Sparse data optimizations for FleCSI show ~5x speedup, but further investigation is required for MPI one-sided communication
- CUDA results are promising, but still require integration with FleCSALE
- Initial machine learning integration was able to reproduce the expected results

ISTI INFORMATION SCIENCE & TECHNOLOGY INSTITUTE