Kokkos implementation of Albany: a performance-portable finite element application

Irina Demeshko*, H. Carter Edwards *, Michael A. Heroux*, Roger P. Pawlowski*,
Eric T. Phipps* and Andrew G. Salinger*
*Sandia National Laboratories, Albuquerque, NM, Email: ipdemes@sandia.gov

***Abstract*—Modern HPC applications need to be run on many different platforms and performance portability has become a critical issue: parallel code needs to be executed correctly and performant despite variation in the architecture, operating system and software libraries. The numerical solution of partial differential equations using the finite element method is one of the key applications of high performance computing. This poster presents a performance portable implementation of the the finite element assembly in the Albany code, based on Kokkos programming model from Trilinos. Evaluation experiments show good performance results for a single implementation across three multicore/manycore architectures: NVIDIA GPUs, Multicore CPUs, Intel Xeon Phi.**

I. Introduction

Albany [1] is a C++ object-oriented, parallel, unstructured-grid, implicit finite element code for solving partial differential equations (PDEs) in various fields of engineering applications. The code is designed for the rapid development of finite-element analysis capabilities enabled through the concept of components-based code design, making pervasive use of libraries from the Trilinos [3] suite. Albany hosts several science and engineering application projects – including the Ice Sheet simulation capability used in this study, the LCM mechanics research code, and the QCAD quantum device simulator – each of which desires performance on new architectures without the need for rewriting the code.

The main objective of this work is creating a performance portable multicore implementation of the Albany code. Porting large, complex scientific and engineering applications to multicore/manycore architectures is a major challenge given the diverse programming models, application programming interfaces (APIs), and performance requirements.

We choose the Kokkos [2] programming model from Trilinos [3] as a tool for creating a performance-portable version of the Albany code. Kokkos is a library-based programming model, which has been developed to provide scientific and engineering codes with a user accessible many core performance portable programming model. There are two main abstractions in Kokkos: 1) dispatch work to a manycore device for parallel execution and 2) manage multidimensional arrays with polymorphic layouts. Kokkos separates `Host` memory space (CPU) from the `Device` memory space (NVIDIA GPUs, IntelPhi etc.). Kokkos `Device` is an abstraction implemented as a C++ template parameter that specify which memory space we want to use, and can be `CUDA` for NVIDIA GPUs, `OpenMP` or `Threads` for multicore CPUs and Intel Xeon Phi. The integration of these abstractions enables users code to satisfy multiple architecture specific memory access pattern performance constraints without having to modify their source code.

By using Kokkos library we write a single code which runs on different architectures by only switching Kokkos `Device` parameter.

II. Albany/FELIX code

We used Albany/FELIX code for performance evaluation tests. FELIX is a dynamical core for ice sheet simulations that solves a variant of Stokes

flow with a 3D unstructured grid Galerkin finite element method, and has solved problems with over 1 Billion unknowns using only distributed-memory – MPI parallelism. This code is being developed as both a stand-alone model for scientific investigations and as the land ice component of coupled climate simulations in DOEs "ACME" Earth System Model. The land ice component is responsible for simulating the evolution of the Greenland and Antarctic Ice Sheets. The Albany/FELIX code is a general finite element code, which includes: gathering data to local data structures, performing element integrations for for viscous and body force terms, and assembling element data into the global data structure. This work was focused on adding on-node parallelism over the number of elements to the existing MPI-partitioned code. The first two steps are inherently thread safe, while the assembly step is not, since a single equation gets contributions from multiple elements. Our Kokkos implementation is based on moving all these steps to the Kokkos `Device`: for each iteration 1) we copy data from the `Host` (CPU) to the `Device`, 2) perform computations on the `Device`, 3) copy results to the `Host`. In order to avoid thread collision in the assembly step, Kokkos Atomic operations have been used: when a thread performs an atomic operation, the other threads see it as happening instantaneously and avoid accessing the same data.

III. Performance Evaluation

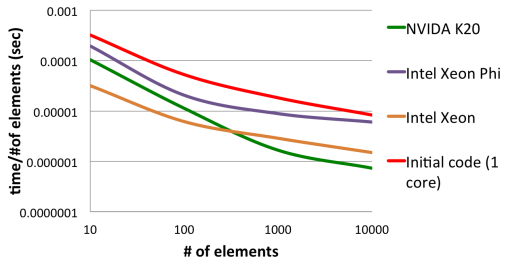

Fig. 1. Time as a function of the number of elements for the Kokkos Albany/FELIX code on Intel Xeon (Sandy Bridge), a pre-production Intel Xeon Phi, and a NVIDIA K20 GPU

Performance evaluation results for the Kokkos implementation of FELIX code are presented in Figure 1. We compare execution time as a function of the number of elements for different architectures. along with the initial code implementation (single CPU core). *Note that the same finite element code base was used for all runs, just with a different configuration option setting the Kokkos* `Device` *template parameter.* It is shown that our Kokkos implementation is faster than initial code. NVIDIA K 20 GPU gives the best performance for the number of nodes more than 500, while Intel Xeon shows faster results when we have less than 500 elements. This can be explained by the fact that in case of small number of elements, almost the entire problem fits into the cache on Intel Xeon, while there is no cache on the GPU. For high number of elements Intel Xeon can't get all benefits from using cache anymore, while GPU exploit more parallelism. Intel Xeon Phi shows the lowest performance results comparing to NVIDIA K20 GPU and Intel Xeon due to its specific architecture.

IV. Conclusion

We have developed a performance portable implementation of the Albany finite element application code based on the Kokkos library from Trilinos. A new Albany-Kokkos implementation is a single code base which runs and is performant on diverse HPC architectures, and is expected to be performant on future architectures that are supported by the Kokkos library. For the example of the FELIX Ice Sheet model in Albany, we showed that our Kokkos implementation gives good performance on multicore-CPUs and manycore-accelerator based machines.

Acknowledgment

This paper is crossreferenced at Sandia as SAND2014-18789A.

References

[1] A Salinger et al., *Albany: A Component-Based Partial Differential Equation Code Built on Trilinos*, Sandia National Labs Technical Report, SAND2013-8430J, 2013.

[2] H. Carter Edwards, Christian R. Trott and Daniel Sundrland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 2014.

[3] M. A Heroux et al., *An overview of the Trilinos project.*, ACM Trans. Math. Softw.31(3) (2005).