A Performance Evaluation of Kokkos & RAJA using the TeaLeaf Mini-App

[Extended Abstract]

Matt Martineau
University of Bristol
Merchant Venturers Building
Bristol, UK
m.martineau@bristol.ac.uk

Simon McIntosh-Smith
University of Bristol
Merchant Venturers Building
Bristol, UK
cssnmis@bris.ac.uk

Mike Boulton
University of Bristol
Merchant Venturers Building
Bristol, UK

Wayne Gaudin
High Peformance Computing
UK Atomic Weapons
Establishment
Aldermaston, UK

David Beckingsale
Lawrence Livermore National
Laboratory
California, USA

ABSTRACT

In this research project we have taken the TeaLeaf 'mini-app' and developed several ports using a mixture of new and mature parallel programming models. We have discovered that RAJA is a promising model with an intuitive development approach, that exhibits good performance on central processing units (CPUs), but does not currently support other devices. Kokkos requires greater up-front development, but represents a highly competitive option for performance portability on CPUs and NVIDIA general-purpose graphics processing units (GPUs). The results showed that Kokkos can exhibit portable performance to within 5% of OpenMP and hand-optimised CUDA for some solvers, but our implementation suffered some issues on the Intel Xeon Phi Knights Corner (KNC).

General Terms

High Performance Computing, Parallel Programming, TeaLeaf

1. INTRODUCTION

TeaLeaf is an open source project that belongs to both the UK Mini App Consortium (UKMAC) [3] and Mantevo project [1], a collection of "mini-apps" that are miniature proxies representing important physics algorithms. Mini-apps support the investigation of optimisation, scalability and performance portability, free from the limitations imposed by attempting such analyses with fully functional scientific applications.

TeaLeaf is a memory bandwidth bound benchmarking tool that solves the heat conduction equation implicitly over a spatially decomposed grid. The application hosts three double precision iterative sparse matrix solvers: Conjugate Gradient (CG), Chebyshev and Preconditioned Polynomial CG (PPCG).

RAJA (developed by Lawrence Livermore National Laboratories) and Kokkos (developed by Sandia National Laboratories) are two new parallel programming models, that are designed to support the performance portability of HPC codes. They both utilise C++ abstractions to simplify the development process and allow applications to be written such that they can be re-compiled for different devices with minimal code impact.

In particular, RAJA utilises the C++11 lambda feature and supports abstract array indirection, but this does currently limit the model to CPUs, as GPU support for lambda kernels is limited [2]. Kokkos utilises template meta-programming to output pthreads, OpenMP, CUDA or native MIC code with only minimal configuration changes.

2. DEVELOPMENT AND TESTING

During the development process it was found that each programming model differed greatly in the complexity and architectural change required to port the TeaLeaf application. RAJA was particularly intuitive, utilising lambda functions to wrap core logic and simple generic parallel dispatch functions. Porting the existing C++ OpenMP implementation of TeaLeaf involved changing the core loops to use the built-in *for_all* style functions and pre-computing the indirection arrays. We found that this port required a similar level of effort as would be required to introduce OpenMP into a similar code.

Implementing the Kokkos port required extensive architectural changes to the code, but those familiar with C++ templates will be comfortable with the resulting kernelisation. All logic was wrapped into functors, and data structures into generic 'Views', which, in TeaLeaf's case, are simple one dimensional arrays of double precision values. Some additional copying semantics were required to handle moving

data from one execution space to another (e.g. host space to CUDA space), but the initialisation of devices and concepts such as queues were abstracted from the developer.

Each implementation was optimised without affecting the core logic of the solvers, also avoiding non-portable directives for the performance portable models. To ensure consistency of results, all of the ports were tested using identical application configurations. The tests were performed across a range of problem sizes to identify change points within the model's performance profiles, using the Blue Crystal phase 3 supercomputer at the University of Bristol, and Cray Inc.'s Swan supercomputer. The devices listed in Table 1 were used for single node testing, where the CPU testing was performed on a pair of processors.

Device	Cores	Bandwidth
Intel Xeon E5-2670	8	51.2 GB/s
NVIDIA Tesla K20X	2688	250 GB/s
Intel Xeon Phi KNC 5110P	60	320 GB/s

Table 1: Devices utilised for testing.

3. RESULTS

The results for the CPU show that all of the programming models exhibit similar levels of performance, but OpenMP is the most performant by between 5% and 40%. Both RAJA and Kokkos use OpenMP as their underlying parallel technology, suggesting that the performance difference is caused by overheads in the implementation. Kokkos was able to achieve impressive performance for the Chebyshev and PPCG solvers on the GPU, but the CG solver was around 50% slower than the CUDA implementation.

Also, Kokkos did not perform as well on the KNC, and collaboration with Sandia National Laboratories uncovered an issue with the native compilation of halo exclusions. A solution was provided by Sandia that utilised hierarchical parallelism, which improved the performance on the KNC to within 5% of the natively compiled OpenMP implementation. This solution was not presented in the results because implementing the changes throughout lead to a performance trade-off, significantly increasing the runtime of the Chebyshev and PPCG solvers on the GPU.

Through testing each model's performance as the problem size was increased in regular intervals (steps of approximately 100,000 cells), we were able to demonstrate the ability for each model to take advantage of a device's available bandwidth. A notable feature of the results is that as the problem size increases, the high bandwidth devices (GPUs and accelerators) continue to scale linearly, whereas the models targeting the CPU suffer from rapidly worsening performance.

4. CONCLUSIONS

Our research has found that RAJA offers clean C++ abstraction that can result in lean code through the use of modern C++11 abstraction. The performance is currently competitive on the CPU but needs to support additional devices to be more appealing to application developers than OpenMP. Kokkos is a much more mature framework that can provide good performance on both CPUs and NVIDIA GPUs, and has features that can currently enable good performance on KNC devices at the expense of complexity and performance on the GPU.

5. REFERENCES

[1] M. Heroux, D. Doerfler, et al. Improving Performance via Mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, 2009.

[2] R. Hornung and J. Keasler. A Case for Improved C++ Compiler Support to Enable Performance Portability in Large Physics Simulation Codes. Technical Report LLNL-TR-635681, Lawrence Livermore National Laboratory, 2013.

[3] UKMAC. UK Mini-App Consortium: TeaLeaf. http://uk-mac.github.io/TeaLeaf, 2015.