Sophisticated Tools for Performance Analysis and Auto-tuning of Performance Portable Parallel Programming

David Boehme
Lawrence Livermore National Laboratory
Livermore, California, USA
boehme3@llnl.gov

Kevin Huck
University of Oregon
Eugene, Oregon, USA
khuck@cs.uoregon.edu

Shravan Kale
University of Oregon
Eugene, Oregon, USA
shravank@uoregon.edu

Vivek Kale
Sandia National Laboratories
Livermore, California, USA
vlkale@sandia.gov

Vanessa Surjadidjaja
Sandia National Laboratories
Albuquerque, New Mexico, USA
vsurjad@sandia.gov

Keywords

HPC tools, parallel programming, performance portable, performance analysis, auto-tuning, performance monitoring, Kokkos, C++

1 Tooling for Performance Portable Programs

Scientific application programmers need to productively improve performance of applications run on a supercomputer having heterogeneous nodes. Using manual methods via basic tools that, e.g., measure raw timings of a parallel loop, are tedious and can be unreliable. So, sophisticated tools which automatically provide insight via performance analysis and optimizations via auto-tuning are used [8, 27]. Such sophisticated tools are available for vendor-specific GPU languages like NVIDIA's Nsight for CUDA [2, 29, 31].

Given the emergence of performance portable programming libraries having abstractions for node-level parallelism [15, 16], an associated set of sophisticated tools will benefit productivity in understanding and optimizing performance. Key challenges for such tools are ensuring the tools work for all vendor backends, function name demangling, and reducing instrumentation overheads. Providing such tools with heed to these challenges will help sustain mass user adoption of performance portable programming libraries. Considering Kokkos and its ecosystem [15], this work showcases and demonstrates how Kokkos application programs run on a supercomputer benefit from the subset of Kokkos Tools [21] that provides performance analysis and auto-tuning capabilities[1], making these tools a viable alternative to corresponding tools for vendor-specific and/or low-level libraries.

2 Kokkos Tools Common Infrastructure

Kokkos Tools is comprised of a set of community-built and in-house Kokkos Tools connectors and a common supporting infrastructure. The infrastructure, which provides common environment variables, a tools callback interface and implementation, and utility connectors, is important to the effectiveness and efficiency of the connectors. Using Kokkos Tools connectors for a Kokkos application is simple: a sequence of pre-built tool connector dynamic libraries, (*.so* files) are specified in the infrastructure's environment variable `KOKKOS_TOOLS_LIBS`. One then runs the Kokkos application's executable as usual. Developing a Kokkos Tool connector involves implementing a small number of event callbacks. Each connector operates independently, though any subset of Kokkos Tools connectors can be built as one library. The Kokkos Tools implementation facilitates low instrumentation overhead through capabilities such as its sampling and filtering utilities.

[1] https://github.com/kokkos/kokkos-tools

3 Sophisticated Kokkos Tools Connectors

This section describes Kokkos Tools connectors for (1) performance analysis and (2) auto-tuning, noting ease of use, ease of development, and efficiency given the Kokkos Tools infrastructure.

3.1 Ensemble Analysis via Caliper

Caliper [12] is a performance instrumentation and profiling library for HPC codes. It provides an instrumentation API for C, C++, and Fortran codes that lets developers mark regions of interest in the application code, while backend components capture events from common third-party libraries and programming frameworks like MPI, CUDA, and HIP, as well as the Kokkos connector interface. Built-in performance measurement recipes cover a wide range of use cases, from lightweight always-on profiling to detailed event tracing. Caliper also provides a measurement control API, which lets users configure and control performance measurements from within the application as well as query performance at runtime.

Caliper's ability to record user-defined labels for code features, either directly through its annotation API or through user-labelled kernels from the Kokkos connector interface, is especially important for modern C++ codes where compiler-generated symbol names are often too obscure to allow meaningful associations with the original source code [13].

Caliper is particularly well-suited for conducting ensemble analyses, such as scaling studies or comparing different application configurations. Ensemble analyses require not just per-kernel performance metrics of each run, but also metadata describing the build and execution configuration. This data typically includes build and execution information such as compiler and build flags, the number of OpenMP threads and/or MPI ranks used, and the program input configuration. Caliper automatically collects this metadata with the help of the Adiak [3] library. The collected data can be analyzed in Thicket [14], an open-source Python toolkit for exploratory data analysis of multi-run performance experiments. At LLNL, many HPC applications now leverage Caliper's ensemble analysis capabilities for automatic performance regression testing as part of their nightly test runs.

3.2 Auto-tuning via APEX

APEX (Autonomic Performance Environment for eXascale) [22] is a performance measurement library for distributed, asynchronous multitasking runtime systems. Implemented in C++, it provides support for lightweight measurement and high concurrency. To support performance measurement in systems that employ user-level threading, APEX uses a task dependency chain in addition to the call stack to produce traces and dependency graphs. APEX is integrated with the Kokkos Tools interface, and APEX has support for all available Kokkos backends through vendor-provided support (CUPTI/NVML, Roctracer/ROCm-SMI, SYCL), open tool interfaces (OMPT), or direct integration (HPX). This support includes synchronous API calls, asynchronous data transfers and kernel executions, and periodic system utilization data.

A key component of APEX is the *Policy Engine*. The Policy Engine of APEX provides a lightweight API to engineer policies that can improve the performance of the application, execute a desired functionality on the runtime or select important runtime and application parameters. APEX uses the Kokkos runtime adaptation interface to automatically tune at runtime RangePolicy and TeamPolicy parameters associated with kernel launch. Custom tuning of any other Kokkos parameter, e.g., thread counts, scheduling policies, tiling factors, chunk sizes, is also possible through this interface. The policy engine can use algorithms from the search library Active Harmony [34], or with exhaustive, random or simulated annealing search methods. Kokkos policies define a unique context based on the specific Kokkos region, its location in the dependency graph, the input parameters and the requested output variable. Once a search has converged, its settings are cached to disk and can be re-used for subsequent simulations.

4 Production-level HPC System Management

An HPC System is a collection of applications run on a supercomputer [10, 29]. Real-time performance management of an HPC System is essential for continual success of production-level scientific simulation via supercomputing [7, 20].

LDMS (Lightweight Distributed Metric Service)[2] is a monitoring software that continuously provides system and application performance used for DoE science simulations. With continuous monitoring, LDMS provides supercomputer users and system administrators with metrics that can be used in statistical and ML-based analysis to detect performance degradation, system abnormalities, or analyze resource management on HPC systems. LDMS was designed for lightweight data collection with extreme-scalability. As a monitoring software, LDMS's quick delivery of application and system metrics allows for automated and at-runtime feedback.

One such application metric collected by LDMS is Kokkos kernel information. Using the Kokkos Tools sampler utility, the LDMS Kokkos Tools connector utilizes LDMS Streams to obtain application kernel information pushed from the Kokkos sampler. Kernel information, packaged into a JSON message, is passed through LDMS's streams and then to indicated storage. In conjunction with Kokkos, along with Caliper and APEX Kokkos Tools connectors, LDMS can collect multiple Kokkos application metrics and use these alongside system metrics to determine an application's optimal run conditions. With LDMS's bi-directional streams, Kokkos applications can adjust their inputs based on current application and system performance.

[2]https://github.com/ovis-hpc/ovis

5 Related Work

OpenMP and its fine-grained tools support via OMPT can also be a path towards productively improving performance for a performance portable program [26, 28]. However, its drawback is that the number of callback events for OpenMP that an OpenMP tool connector developer has to implement and setup is significantly larger than that of a Kokkos Tool connector.

The ubiquitous performance profiling tools `perf` and `gprofng` [5] provide sophisticated performance analysis for any application program run on a Linux system, but it is generalized towards single-process profiling. Unlike Caliper, it does not have detailed and meaningful tracing capabilities aimed at science applications.

OpenTuner [6] is a generic program auto-tuner using ensembles of built-in or user-defined search techniques. Bliss [30] uses multiple, diverse Bayesian Optimization Models to identify the best model output, i.e., tuning parameter configuration, for an application-architecture pair, and its approach avoids a sub-optimal local minima. APEX uses Active Harmony for search strategies but it also offers performance portable auto-tuning of Kokkos programs which can run across a variety of backends.

MPCDF [33] allows for performance analysis and insight of HPC Systems at TU-Munich's HPC clusters. Unlike MPCDF, LDMS directly integrates with Kokkos, thereby providing real-time insight on performance for a Kokkos application that is more meaningful through, e.g., identifying very high GPU memory utilization of a Kokkos parallel kernel in a Kokkos application.

6 Conclusion

In this work, we demonstrated how Kokkos Tools connectors for performance analysis and auto-tuning along with a common Kokkos Tools infrastructure allow for productively improving performance of a science application needing to be run on a supercomputer. We also showed how these connectors can work seamlessly together and provide benefits to production-level HPC System management.

For future work, we will (1) experiment with inter-node parallel applications using MPI+Kokkos, (2) assess viability of the connectors discussed for parallel Python and Fortran applications and (3) use AI/ML and the connectors discussed to develop an HPC System performance feedback system [7].

Acknowledgments

This work is supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy, Office of Science and the National Nuclear Security Administration. APEX is supported by the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR) under contract DE-SC0021299. Sandia National Laboratories is a multimission laboratory managed and operated by NTESS LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's NNSA under contract DE-NA0003525. This paper is cross-referenced at Sandia as SAND2023-09180C.

References

[1] 2020. The LLVM Compiler Infrastructure. http://llvm.org/.

[2] 2022. Best Practices Guide::CUDAToolkit. Section 9.1: Asynchronous and Overlapping Transfers with Computation. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#asynchronous-transfers-and-overlapping-transfers-with-computation [Online; Accessed September 1, 2022].

[3] 2023. Adiak. http://github.com/LLNL/adiak.

[4] 2023. Perlmutter User Guide. https://www.nersc.gov/systems/perlmutter/.

[5] Yara Ahmad. 2022. Perf vs gprof: Comparing Software Performance Profiling Tools. https://www.redhat.com/architect/perf-vs-gprofng. https://www.redhat.com/architect/perf-vs-gprofng [Online; Accessed August 3rd, 2023].

[6] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman P. Amarasinghe. 2014. OpenTuner: an extensible framework for program autotuning. In *International Conference on Parallel Architectures and Compilation, PACT '14, Edmonton, AB, Canada, August 24-27, 2014*, José Nelson Amaral and Josep Torrellas (Eds.). ACM, 303–316. https://doi.org/10.1145/2628071.2628092

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2009. *Above the Clouds: A Berkeley View of Cloud Computing.* Technical Report. University of California at Berkeley. http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html

[8] D H Bailey, J Chame, C Chen, J Dongarra, M Hall, J K Hollingsworth, P Hovland, S Moore, K Seymour, J Shin, A Tiwari, S Williams, and H You. 2008. PERI Auto-tuning. *Journal of Physics: Conference Series* 125, 1 (2008), 012089. http://stacks.iop.org/1742-6596/125/i=1/a=012089

[9] Seonmyeong Bak, Colleen Bertoni, Swen Boehm, Reuben Budiardja, Barbara Chapman, Johannes Doerfert, Markus Eisenbach, Hal Finkel, Oscar Hernandez, Joseph Huber, Shintaro Iwasaki, Vivek Kale, Paul Kent, JaeHyuk Kwack, Meifeng Lin, Piotr Luszczek, Ye Luo, Buu Pham, Swaroop Pophale, Kiran Ravikumar, Vivek Sarkar, Thomas Scogland, Shilei Tian, and P.K. Yeung. 2021. OpenMP Application Experiences: Porting to Accelerated Nodes. In *Journal of Parallel Computing*. https://doi.org/10.1016/j.parco.2021.102856

[10] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick. 2008. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems.* Technical Report 0. University of Notre Dame, Computational Science and Engineering Department.

[11] Abhinav Bhatele, Lukasz Wesolowski, Eric Bohm, Edgar Solomonik, and Laxmikant V. Kale. 2010. Understanding Application Performance via Micro-benchmarks on Three Large Supercomputers: Intrepid, Ranger and Jaguar. *International Journal of High Performance Computing Applications (IJHPCA)* (2010). http://hpc.sagepub.com/cgi/content/abstract/1094342010370603v1.

[12] David Boehme, Todd Gamblin, David Beckingsale, Peer-Timo Bremer, Alfredo Gimenez, Matthew LeGendre, Olga Pearce, and Martin Schulz. 2016. Caliper: Performance Introspection for HPC Software Stacks. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 550–560. https://doi.org/10.1109/SC.2016.46

[13] David Boehme, Kevin Huck, Jonathan Madsen, and Josef Weidendorfer. 2019. The Case for a Common Instrumentation Interface for HPC Codes. In *2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools)*. 33–39. https://doi.org/10.1109/ProTools49597.2019.00010

[14] Stephanie Brink, Michael McKinsey, David Boehme, Connor Scully-Allison, Ian Lumsden, Daryl Hawkins, Treece Burgess, Vanessa Lama, Jakob Luettgau, Katherine E Isaacs, et al. 2023. Thicket: Seeing the Performance Experiment Forest for the Individual Run Trees. *High-Performance Parallel and Distributed Computing (HPDC 2023)* 6, 7 (2023), 23.

[15] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. 2014. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *J. Parallel and Distrib. Comput.* 74, 12 (2014), 3202–3216. https://doi.org/10.1016/j.jpdc.2014.07.003 Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[16] R. Hornung H. Jones W. Killian A. J. Kunen O. Pearce P. Robinson B. S. Ryujin T. R. W. Scogland D. A. Beckingsale, J. Burmark. 2019. RAJA: Portable Performance for Large-Scale Scientific Applications. *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)* (2019).

[17] Kaushik Datta. 2009. *Auto-tuning Stencil Codes for Cache-Based Multicore Platforms.* Ph. D. Dissertation. EECS Department, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-177.html

[18] LLVM Developers. [n. d.]. Optimizations Remarks in LLVM. http://llvm.org/docs/Remarks.html.

[19] LLVM Developers. 2020. OpenMP in LLVM. http://openmp.llvm.org/docs/.

[20] Constantinos Evangelinos and Chris N. Hill. 2008. Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. In *Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2* (Chicago, IL). Cloud Computing and Its Applications.

[21] Si Hammond. [n. d.]. Profiling and Debugging Support for Kokkos. https://www.osti.gov/servlets/purl/1531140

[22] Kevin A. Huck. 2022. Broad Performance Measurement Support for Asynchronous Multi-Tasking with APEX. In *2022 IEEE/ACM 7th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*. 20–29. https://doi.org/10.1109/ESPM256814.2022.00008

[23] Susan Flynn Hummel, Jeanette Schmidt, R. N. Uma, and Joel Wein. 1996. Load-sharing in Heterogeneous Systems via Weighted Factoring. In *Proceedings of the Eighth Annual ACM symposium on Parallel Algorithms and Architectures* (Padua, Italy) *(SPAA '96)*. ACM, Padua, Italy, 318–328.

[24] Shoaib Kamil, Cy Chan, Samuel Williams, Leonid Oliker, John Shalf, Mark Howison, and E. Wes Bethel. 2009. A Generalized Framework for Auto-tuning Stencil Computations. In *In Proceedings of the Cray User Group Conference*.

[25] Tobias Klug, Michael Ott, Josef Weidendorfer, Carsten Trinitis, and Technische Universität München. 2008. autopin, Automated Optimization of Thread-to-Core Pinning on Multicore Systems.

[26] LLVM Developers 2020. LLVM/Clang OpenMP Support. https://clang.llvm.org/docs/OpenMPSupport.html.

[27] Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A High-performance, Portable Implementation of the Message Passing Interface Standard. *Parallel Comput.* 22 (1996), 789–828.

[28] Joachim Protze. 2017. How to Get the Most out of the ompt Profiling Interface. https://openmpcon.org/wp-content/uploads/openmpcon2017/Day2-Session3-Protze.pdf.

[29] Daniel Reed, Dennis Gannon, and Jack Dongarra. 2022. Reinventing High Performance Computing: Challenges and Opportunities. arXiv:2203.02544 [cs.DC]

[30] Rohan Basu Roy, Tirthak Patel, Vijay Gadepally, and Devesh Tiwari. 2021. Bliss: Auto-Tuning Complex Applications Using a Pool of Diverse Lightweight Learning Models. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 1280–1295. https://doi.org/10.1145/3453483.3454109

[31] Usman Saleem. 2022. NVIDIA's Tensor Cores for Machine Learning and AI – Explained. https://appuals.com/nvidias-tensor-cores-for-machine-learning-and-ai-explained/. https://appuals.com/nvidias-tensor-cores-for-machine-learning-and-ai-explained/ [Online; Accessed September 1, 2022].

[32] Sameer S Shende and Allen D Malony. 2006. The TAU parallel performance system. *The International Journal of High Performance Computing Applications* 20, 2 (2006), 287–311.

[33] Luka Stanisic and Klaus Reuter. 2019. MPCDF HPC Performance Monitoring System: Enabling Insight via Job-Specific Analysis. arXiv:1909.11704 [cs.DC]

[34] Cristian Tapus, I-Hsin Chung, and Jeffrey K Hollingsworth. 2002. Active harmony: Towards automated performance tuning. In *SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE, 44–44.

[35] Xingfu Wu, Prasanna Balaprakash, Michael Kruse, Jaehoon Koo, Brice Videau, Paul Hovland, Valerie Taylor, Brad Geltz, Siddhartha Jana, and Mary Hall. 2023. ytopt: Autotuning Scientific Applications for Energy Efficiency at Large Scales. arXiv:2303.16245 [cs.DC]