Integrated Measurement for Cross-Platform OpenMP Performance Analysis

Kevin A. Huck[1], Allen D. Malony[1], Sameer Shende[1], and Doug W. Jacobsen[2]

[1] University of Oregon, Eugene, OR 97403, USA
[2] Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Abstract. The ability to measure the performance of OpenMP programs portably across shared memory platforms and across OpenMP compilers is a challenge due to the lack of a widely-implemented performance interface standard. While the OpenMP community is currently evaluating a tools interface specification called OMPT, at present there are different instrumentation methods possible at different levels of observation and with different system and compiler dependencies. This paper describes how support for four mechanisms for OpenMP measurement has been integrated into the TAU performance system. These include source-level instrumentation (Opari), a runtime "collector" API (called ORA) built into an OpenMP compiler (OpenUH), a wrapped OpenMP runtime library (GOMP using ORA), and an OpenMP runtime library supporting an OMPT prototype (Intel). The capabilities of these approaches are evaluated with respect to observation visibility, portability, and measurement overhead for OpenMP benchmarks from the NAS parallel benchmarks, Barcelona OpenMP Task Suite, and SPEC 2012. The integrated OpenMP measurement support is also demonstrated on a scientific application, MPAS-Ocean.

1 Introduction

Any parallel language system based on compiler interpretation of parallel syntax, code transformation, and runtime systems support introduces the question of how external tools can observe the execution-time performance of the generated program code. In general, the parallel language observability problem is one of *performance visibility* coupled with *semantic context*. Performance visibility is the means by which a tool can gain access to and can acquire information about the parallel execution performance. Because the parallel program is being transformed during compilation and is utilizing a runtime environment for execution, there are different levels where visibility can be made operative. However, it is precisely for this reason that performance data must be augmented with semantic information about the context in which the data was acquired, in order to understand its relevance and be able to link performance behavior across levels. The coupling between visibility and context can be implicit (e.g., an instrumented event could carry semantics) or explicit (e.g., the compiler or runtime allows certain state to be queried). The key point is that observing parallel language performance requires both types of capabilities and with support across different levels of program generation and execution.

L. DeRose (Eds.): IWOMP 2014, LNCS 8766, pp. 146–160, 2014.

OpenMP [2] is a widely supported and commonly used programming specification for multi-threaded parallelism in which performance observability has been an important consideration as well as a challenge to provide in a portable manner. It is a pragma-based language extension for C/C++ and Fortran plus a user-level library that relieves the programmer from the burden of creating, scheduling, managing, and destroying threads explicitly. Compilers supporting OpenMP hide this complexity by generating code that supports the (user facing) concurrency semantics in the OpenMP language, but interfaces with a runtime environment that is free to build (system facing) thread-level operation in whatever way is desired. Thus, available OpenMP compilers, both open source (e.g., GNU, OpenUH/Open64) and vendor supplied (e.g., IBM, Microsoft, Intel, PGI, Cray), offer alternative implementations on different platforms.

How can portable OpenMP performance measurement and analysis be provided across compilers and platforms given this situation? The OpenMP community discussed performance observability requirements early on in its history and there has been a strong interest in defining a tools API as part of the OpenMP standard. Efforts are underway by the OpenMP Tools Working Group in this regard. In the meantime, various techniques have been created to enable OpenMP performance analysis. This paper presents the present landscape and looks at the integration of different OpenMP observation techniques with a common measurement infrastructure, the TAU Performance System® [31].

The main contribution of this work is the creation of a robust, cross-platform OpenMP performance measurement system that utilizes the available support for observing OpenMP runtime system behavior. Four different approaches were integrated with TAU serving as the core measurement framework, including a portable tool for GCC OpenMP performance analysis. The overall goal is to provide significant coverage across OpenMP environments and parallel systems to allow portable performance evaluation of OpenMP applications.

2 Background

As described by Mohsen et al. [24], several performance tools can perform thread-specific measurement of an OpenMP application. In some cases, vendor-supplied tools such as the Intel Thread Profiler [12] provide specific details about one particular OpenMP compiler by utilizing proprietary instrumentation in the OpenMP runtime. On the other hand, portable performance tools such as TAU [31], ompP [5], Kojak [23], Scalasca [6] and Vampir [14] provide thread-specific measurements that can include the OpenMP static and runtime context, such as the parallel region location or synchronization operation (barrier, lock, atomic, critical), but only if that information is available. Typically, to access the OpenMP context these performance tools rely on either instrumentation of the application code or the OpenMP runtime itself, if it is available.

Two early efforts to define a tools interface for OpenMP performance observation were *POMP* [22] and the Sun/Oracle OpenMP Runtime API (ORA), better known as the *Collector API* [13]. The POMP interface was intended to be used