A Standard Debug Interface for OpenMP Target Regions

Extended Abstract

Andreas Erik Hindborg
Technical University of Denmark
ahin@dtu.dk

Ignacio Laguna
Lawrence Livermore National Laboratory
ilaguna@llnl.gov

Sven Karlsson
Technical University of Denmark
svea@dtu.dk

Dong H. Ahn
Lawrence Livermore National Laboratory
ahn1@llnl.gov

1. INTRODUCTION
As OpenMP covers more on-node parallelism options, debugging OpenMP programs is becoming increasingly challenging. Worse, debuggers often debug OpenMP at a system-thread level, leaving the programmers with limited state information on OpenMP threads. The OpenMP debugging interface, OMPD, has been proposed as a standard interface to overcome these limitations [3]. However, the current specification works only for OpenMP 3.1—it cannot handle OpenMP 4.0 programs because it does not consider OpenMP target construct. In this work, we study how to extend OMPD to OpenMP 4.0 and propose a set of OMPD extensions. We identify crucial scalability issues with the way OMPD currently represents the threads and propose a new representation, Construction of Online Lightweight Thread handle, COLT, as a scalable solution to OpenMP target regions. Our evaluation shows that our scheme is feasible as far as basic OMPD support is provided for OpenMP 4 runtime component and can also significantly reduce the memory and performance overheads of the debuggers.

2. TARGET REGIONS WITH OMPD
The OMPD interface commonly is implemented as a shared library, which is loaded into the address space of the debugger such as GDB, see figure 1. OMPD provides the debugger with an ability to reconstruct the data structures and state of the OpenMP runtime running in the address space of the debugged application. It must allow the debugger to discover and traverse the hierarchy of OpenMP objects associated with any OpenMP thread (see figure 2a). Thus, when an OpenMP target region is offloaded to a device, the debugger, via OMPD, must ultimately be able to access the address space of the device and fetch information on all of the OpenMP threads spawned within the target region.

While the current OMPD specification supports up to OpenMP 3.1, one can conceive various ways to extend it for the target construct. A simplest approach would be to extend the current scheme whereby OMPD will return to the debugger a handle for each OpenMP thread. However, this approach breaks down on devices that utilize many threads. Current GPU devices illustrate this issue. For example, an NVIDIA device is capable of hosting up to 2^{15} resident threads [1]. Representing each of these threads with a thread handle will incur a very large overhead, see figure 2b.

As an alternative, we propose to represent teams and threads in target regions by a target region handle and two unsigned integers, see figure 2c. The OpenMP 4.0 specification requires that teams in target regions are identified by a set of consecutive integers starting at zero and ending at one less than the number of teams. Similarly, threads in a team are identified by integers in the range of zero to one less than the number of threads in the team. Therefore, any thread in any target region can be identified by a target-region handle and two unsigned integers. This thread representation allows for consecutive threads in consecutive teams, common query patterns, to be compactly reported as ranges instead of individual thread identifiers. Perhaps more importantly, the compact thread handles can be constructed by the debugger on demand. We refer to this thread representation scheme as Construction of Online Lightweight Thread handle, COLT.

3. PROTOTYPE IMPLEMENTATION
We propose a set of extensions to OMPD to enable it to be used scalably for OpenMP 4.0 target devices, see attached poster for details. To evaluate its feasibility and performance, we implemented a prototype of the proposed API extension on top of our previous work that implements the current OMPD specification. We built our extension prototype for OpenMP programs compiled with Clang [5] as well as linked with both the Intel OpenMP Runtime [4] and the Clang libomptarget runtime [2]. We integrated the library into the cuda-gdb debugger from NVIDIA [6].

Our experience is that OMPD/COLT can be implemented for this environment, provided that other basic OMPD functionality such as OpenMP state tracking is added to li-
Figure 1: OMPD architecture overview.

Figure 2: Examples of the hierachical OpenMP objects accessible via the OMPD interface.
bomptarget, a device neutral OpenMP offloading library. Further our preliminary evaluation on performance and scalability suggests that COLT can significantly reduce the memory and performance overhead of the debugger debugging target regions.

4. ACKNOWLEDGMENTS
This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-ABS-675382). The authors acknowledge the contributions of John DelSignore and Ariel Burton of Rogue Wave Software who reviewed our work and provided insightful feedback on it.

5. REFERENCES