Task Profiling through OpenMP Runtime API and Tool Support

Ahmad Qawasmeh, Abid Malik, Deepak Eachempati, Barbara Chapman
University of Houston
Dept. of Computer Science
Houston, TX 77004, USA
{arqawasm,ammalik3,dreachem,chapman}@cs.uh.edu

ABSTRACT

The introduction of tasks in the OpenMP programming model brings a new level of parallelism. This also creates new challenges with respect to its meanings and applicability through an event-based performance profiling. The OpenMP Architecture Review Board (ARB) approved an interface specification known as the "OpenMP Runtime API for Profiling" to enable performance tools to collect performance data for OpenMP programs. In this paper, we propose new extensions to the OpenMP Runtime API for profiling task level parallelism. We present an efficient method to distinguish individual task instances in order to track their associated events. We implement the proposed extensions in the OpenUH compiler which is an open-source OpenMP compiler. With negligible overheads, we are able to capture important events like task creation, execution, suspension, and exiting. These events help in identifying overheads associated with the OpenMP tasking model, e.g., task waiting until a task starts execution or task cleanup etc. These events also help in constructing important parent-child relationships that define tasks' call paths. Additionally, we integrate our ORA into the performance tool TAU to visualize task measurements at the construct level. The proposed extensions are in line with the newest specifications recently proposed by the OpenMP tools committee for task profiling.

Keywords

OpenMP, OpenMP Runtime API for Profiling, Task Profiling, Open Source

1. INTRODUCTION

OpenMP is a standard directive-based API for shared memory programming. The lack of standards in the runtime layer has hampered the development of third-party tools to support OpenMP application development. The OpenMP Runtime API (ORA) for profiling OpenMP applications was presented in [3]. The ORA was accepted by the tools committee of the ARB. The API is designed to permit a tool, known as a collector, to gather information about an OpenMP program from the runtime system in such a manner that neither the collector nor the runtime system needs to know any details about each other. These tools should maintain information about the OpenMP execution model in order to trouble-shoot OpenMP specific performance problems. The ORA does not require modifications to an application's source code. Consequently, compiler analysis and optimizations will not be affected. Hence, the performance measurements of an application are more accurate.

Applications exhibiting irregular parallelism were not taken care of before the introduction of tasking in the OpenMP model. Tasking has added a new dimension of concurrency to OpenMP applications. The task construct allows a developer to dynamically create asynchronous units of work to be scheduled at runtime. Two types of tasks have been introduced in the OpenMP specification; **1) Tied tasks 2) Untied tasks**. Tied tasks can be suspended at specific scheduling points. Untied tasks can be suspended at any point in an OpenMP program according to OpenMP 3.1 specifications. Moreover, a tied task is linked to one thread only while an untied task can be resumed by any thread in the team. The new ORA extenstions we propose in the runtime of OpenUH [4] allows developers to:

- Distinguish the individual task instances in the same task construct by assigning a distinct ID to each task instance.
- Track creation, switching, suspension, resumption, exiting, and completion of task instances as well as task constructs.
- Allow collector tools to maintain performance measurements associated with the aforementioned events.

Our implementation supports C/C++ and Fortran programs with tied tasks and untied tasks. Furthermore, our implementation is in line with the task profiling specification proposed by the OpenMP ARB tools committee [2]. To the best of our knowledge, the work reported here is the first open-source implementation of the ORA with extensions to support tasks. Our implementation is explained next followed by an evaluation, conclusion, and future work hints.

2. IMPLEMENTATION

The OpenUH compiler supports OpenMP 3.0 tasking on the IA-64, IA-32, x86_64, and Opteron Linux ABI platforms. This includes the front-end support ported from the GNU C/C++ compiler, back-end translation implemented by the HPCTool group at the University of Houston jointly with the Tsinghua University, and an efficient task scheduling infrastructure developed by the HPCTools group.

2.1 OpenMP Task Implementation in OpenUH

The OpenMP tasking directives are translated into the OpenMP runtime routines.[1] We use the OpenMP runtime routines to capture the OpenMP events and states related to the OpenMP task, such as task creation, task waiting in the task pool, task switching from a create state to a suspend state etc. These states and events are captured by simply modifying the OpenMP runtime routines, without modifying the OpenMP translation of the source code.

2.2 OpenMP Task Profiling API

The ORA interface consists of a single routine that takes the form: *int __omp_collector_api (void *arg).* The **arg** parameter is a pointer to a byte array that can be used by a collector tool to pass one or more requests for information from the runtime. The collector requests notification of a specific event by passing the name of the event to be tracked as well as a callback routine to be invoked by the OpenMP runtime each time the event occurs. The interaction between the tool and the OpenMP runtime library is obtained via the ORA through a set of implemented requests, events, and states.

- Task Creation Events and States: Capture the start and completion of a task creation.
- Task Suspension Events and States: Capture the start and completion of a task suspension. This suspension occurs when a taskwait construct is encountered.
- Task Execution/Exiting Events and States: Capture the start and completion of task execution/exiting. Once a task is created, it may start executing immediately or with a delay depending on threads availability.
- Task IDs and Parent Task IDs: A distinct ID is assigned to each task instance. Each time a new task is created, the task ID is incremented atomically to ensure that only one thread can modify this field at any instance of time. The parent task ID is obtained by having a pointer to the parent task. Two requests are defined to enable the collector tool to obtain these IDs at any given point of the program execution.

3. EVALUATION

We evaluated our implementation in the OpenUH compiler. We performed the following analyses;

- We measured the overheads introduced by the inclusion of our implementation in the runtime.
- We tracked the newly developed task IDs, states, and events through our employed requests at the task-instance level. This part was achieved by developing a prototype OpenMP task profiler.
- We integrated our ORA into the performance tool TAU [5]. We aggregate the task instances called from the same calling context to visualize task measurements at the construct level.

[1] All tables and figures can be found in our poster.

We used the Barcelona OpenMP Task Suite (BOTS) kernels [1] as benchmark applications. The experiments were done using the x86_64 Linux system with four 2.2 GHz 12-core AMD Opteron processor (48 cores total). The results show that the overhead associated with our implementation is insignificant. The absolute overhead percentage ranges from 0% to 5% of the execution time. The average overhead percentage obtained is less than 1%.

4. CONCLUSIONS

We have presented our experiences in implementing a new API for OpenMP task profiling. The OpenMP Runtime API for profiling (ORA) was approved by the OpenMP tool committee to create a standardized tool interface for OpenMP programs. We have extended the ORA to support profiling for OpenMP tasks at both task-instance level and construct level. We have implemented our extensions using the OpenUH open-source compiler. Our extensions to the ORA allow the execution and scheduling of tied and untied OpenMP tasks to be tracked by a tool to collect performance measurements. These measurements assist application developers to gain more insight into the dynamic behavior of OpenMP based applications. A visualization tool support for the ORA was developed using TAU. Our extensions adhere to the proposal recently suggested by the OpenMP tool committee for task profiling. Moreover, Our experiments show that the overheads associated with our implementation are negligible.

5. REFERENCES

[1] A. Duran, X. Teruel, R. Ferrer, X. Martorell, and E. Ayguade. Barcelona OpenMP tasks suite: A set of benchmarks targeting the exploitation of task parallelism in OpenMP. In *Parallel Processing, 2009. ICPP'09. International Conference on*, pages 124–131. IEEE, 2009.

[2] A. Eichenberger, J. Mellor-Crummey, M. Schulz, N. Copty, J. DelSignore, R. Dietrich, X. Liu, E. Loh, D. Lorenz, and other members of the OpenMP Tools Working Group. OMPT and OMPD: OpenMP tools application programming interfaces for performance analysis and debugging. (OpenMP 4.0 draft proposal). 2013.

[3] M. Itzkowitz, O. Mazurov, N. Copty, and Y. Lin. An OpenMP runtime API for profiling. *OpenMP ARB as an official ARB White Paper available online at http://www. compunity. org/futures/omp-api. html*, 314:181–190, 2007.

[4] A. Qawasmeh, B. Chapman, and A. Banerjee. A compiler-based tool for array analysis in HPC applications. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 454–463. IEEE, 2012.

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