Ivy Profiler: A Lightweight Performance Analysis Tool for Multicore Systems

Yang Hun Park, Jae Young Jang, and Jae W. Lee
Sungkyunkwan University
{pyh5431, rhythm2jay, jaewlee}@skku.edu

Abstract

In this paper, we propose Ivy Profiler, a lightweight, platform independent performance analysis tool for multicores. Composed of public APIs and hardware abstract layer (HAL), which cleanly separate platform-dependent and independent portions, Ivy Profiler provides great portability along with a simple, easy-to-use programming interface. Also, it supports per-section instrumentation for fine-grained analysis for parallel regions. Together with visualization tool support, users can easily identify performance bottlenecks of a parallel program. Ivy Profiler enables effective bottleneck analysis and performance tuning on a variety of parallel platforms.

Keywords: profiling, multicore, performance, tuning

1. Introduction

With Moore's Law the number of processing cores on a chip will continue to increase. Manycores with tens or hundreds of available cores have already hit the market (e.g., Intel's 61-core Knights Corner). In such a manycore environment, performance scalability is the key to exploit the abundant hardware resources most effectively. Therefore, performance analysis tools that analyze complex bottlenecks are of great value.

Recently, there have been several performance analysis tools for profiling multicore systems such as Intel VTune [1], Valgrind [2], PAPI [3] and so on. However, these tools are inadequate for fine-grained analysis and platform dependent (i.e. Intel Vtune). While some tools (i.e. Valgrind, PAPI) support platform-independent APIs and instrumentation features, but they are difficult to use due to its heavy and complex module. Moreover, because they simply report low-level performance numbers directly from performance-monitoring unit supported by hardware [4] (e.g. Intel Performance Monitoring Unit), it is hard to catch *program-level* performance bottlenecks such as load imbalance among tasks, which is typical in applications composed of many sub-tasks inside. In this paper, we propose a lightweight, platform-independent, and easily-customizable performance analysis tool for multicores, called *Ivy Profiler*. Ivy Profiler supports simple user-friendly APIs and abstracts away hardware details by employing a hardware abstraction layer (HAL). Also, it supports per-parallel section instrumentations and intuitive performance analysis by integrating a visualization tool. With these features, Ivy Profiler greatly helps accurate analysis and characterization of various parallel frameworks and workloads on multicore systems.

2. Implementation: Ivy Profiler

Ivy Profiler is based on ANSI C and consists of user specific public APIs and platform-dependent hardware abstract layer (HAL). By separating platform-dependent specifications from user APIs, Ivy Profiler effectively eliminates needs for users to understand platform-dependent details (e.g., interface to performance counters). Through 1) passing events into user-specific input files and 2) applying user APIs to pinpoint program regions to profile, users obtain analysis results with minimal efforts.

Additionally, Ivy Profiler supports section per-section instrumentations. By simply passing section IDs to user APIs, profiling results are gathered for each section. With additional visualization tools (i.e. gnuplot-based tools), per-section information can be effectively merged for further performance analysis and users can easily use this to find performance bottlenecks (e.g. load balance among the tasks).

In order to use Ivy Profiler, users first need to set

Table 1 Ivy Profiler: API Table

Function	Description
Public API (platform independent)	
ivyp_set_config(filename)	Parse xml file to get config setting
ivyp_start(key, section)	Mark beginning of a region with key, and section tag
ivyp_end(key, section	Mark end of a region with key and section tag
ivyp_report(void)	Report profiling results
Hardware Abstraction Layer API (platform dependent)	
ivyp_get_time(void)	Get current time in usec
ivyp_get_power(void)	Get cumulative power consumption in watt
ivyp_get_llc_misses(void)	Get cumulative last-level cache misses

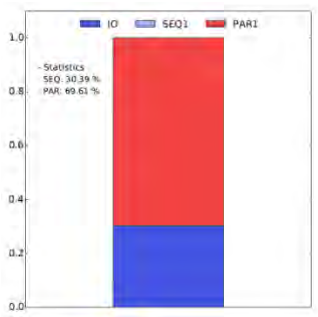

Figure 1 Amdahl's Limit: Blackscholes

configurations by editing an XML input file which consists of flags for profiling events and the visualization tool. By turning on/off these options, profiling events and reporting format are determined.

After input file configuration, users need to apply public APIs to mark sections where they want to profile. Column 3-6 of Table 1 shows public APIs used for instrumentation. First, users should specify the configuration input file with *ivyp_set_config()* API. After, for profiling specific section, users need to write *ivyp_start()* at start of the section and ends by writing *ivyp_end()* API. These APIs have *key* and *section* as parameters. *Key* is used for naming the profiled region which users marked. *Section* parameter is a flag, which indicates either sequential or parallel region. Finally, to report analysis results, *ivyp_report()* API is used. This reports profiling information in log file. If users checked visualization option in configuration file, then *ivyp_report()* makes graphs additionally.

3. Application Examples

In this section, we demonstrate how effectively Ivy Profiler analyzes parallel workloads. We used blackscholes program from PARSEC benchmark suite version 3.0 [5]. We use an Intel 4th Generation Core i7 4770 processor (4-physical, 8-logical core with hyper-threading) for evaluation. The processor has 32KB L1-D and L1-I, 256KB L2, and 8MB L3 caches. The program runs with sixteen threads

Figure 1 shows Amdahl's limit of the application. From this, users obtain the ratio of the parallelizable portion and sequential portion of the entire program intuitively. With this, potential performance benefits can be estimated with fully parallelized program.

Also, it is possible to classify the cause of bottlenecks. Figure 2 shows time chart analysis for categorizing each instrumented section. Each section is categorized by section ID given from the public API. As in the figure, it gives execution time of each section and load balance of the cores in parallel section. Also synchronization overhead would be known.

Figure 3 shows power consumption of parallel section in a program according to the number of threads. As the number of threads increases, power

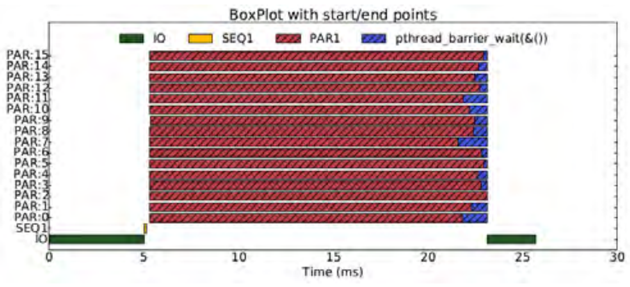

Figure 2 Timeline Analysis: Blackscholes

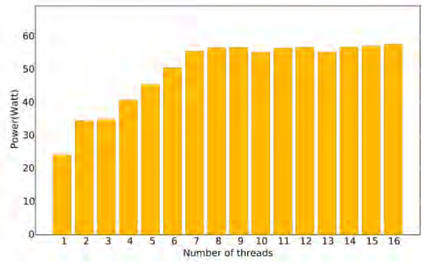

Figure 3 Event Profiling: Power Consumption

consumption of parallel section of the program is also increases. However, power consumption is saturated over 7 threads. With this, users obtain energy efficient number of cores when running programs.

4. Conclusion

We have proposed a lightweight selective profiling tool for parallel frameworks and workloads on multicore systems. It provides profiling information of various bottleneck and users can choose what they want to profile. Current version of Ivy Profiler has low platform coverage and profiling functionality limited to time chart analysis, power events and cache miss events. In the future, we will expand supported platforms and its functionality such as data contention analysis and detailed memory profiling events.

Acknowledgements

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (#14-824-09-011, "Research Project on High Performance and Scalable Manycore Operating System").

References

[1] "Intel Vtune Performance Amplifier," [Online]. Available: https://software.intel.com.

[2] "Valgrind Developers," [Online]. Available: http://valgrind.org.

[3] "PAPI," [Online]. Available: http://icl.cs.utk.edu/papi.

[4] "Intel 64 and IA-32 Architectures Software Developer's Manual," [Online]. Available: http://www.intel.com.

[5] "PARSEC benchmark suite," [Online]. Available: http://parsec.cs.princeton.edu.

[6] M. J.L, "Challenges and Opportunities in Many Core Computing," in *Proceedings of the IEEE*, May 2008.