

ORACLE SOLARIS STUDIO PERFORMANCE ANALYZER

KEY FEATURES

- Low overhead for fast and accurate results
- Advanced profiling of single-threaded and multithreaded applications
- Support for multiprocess and system-wide profiling
- Ability to analyze MPI applications
- Support for C, C++, Fortran, and Java code
- Optimized for the latest Oracle systems

BENEFITS

- Maximizes application performance
- Improves system utilization and software quality
- Increases developer productivity

The Oracle Solaris Studio Performance Analyzer provides unparalleled insight into the behavior of your application, allowing you to identify bottlenecks and improve performance by orders of magnitude. Bring high-performance, high-quality, enterprise applications to market faster and obtain maximum utilization from today's highly complex systems.

Introduction

Is your application not performing optimally? Are you taking full advantage of your high-throughput multicore systems? Understanding the performance of enterprise applications is not a simple task. Enterprise systems are created for extreme scalability and high performance for demanding and consolidated enterprise workloads. Creating optimally tuned applications in these complex environments is a daunting task. Oracle Solaris Studio provides compilers and libraries that can tune an application for the best code sequences, but understanding performance characteristics of complex, long-running, interdependent applications is challenging. The Performance Analyzer is a powerful market-leading performance analysis tool for developers who need to optimize application performance and scalability. It provides in-depth analysis that enables you to quickly understand your application's behavior, allowing you to easily eliminate hotspots, bottlenecks, and areas of high resource consumption.

Analyze Serial and Parallel Applications

The Performance Analyzer is optimized for a variety of programming scenarios and helps pinpoint the offending code with the fewest number of clicks. In addition to supporting C, C++, and Fortran applications, the Performance Analyzer also includes support for Java code. All data is converted into function source, lines of code, or disassembly code with caller/callee values. Additionally, compiler transformations are interpreted to show how the optimizer transforms source code. For Java programs, the Performance Analyzer highlights which methods are most expensive, how they are affected by Hotspot compilation, and how memory allocation and garbage collection affect behavior. The Performance Analyzer also handles C++ and Java mangled names.

The Performance Analyzer can be used to profile single-threaded as well as multithreaded applications written using PThreads, Oracle Solaris threads, and OpenMP. These multithreading models as very common as underlying chips offer an increasingly larger numbers of cores and hardware threads. Thus, application performance is emerging as a critical factor, with bad performance increasingly considered to be program failure.

The Performance Analyzer highlights how threads are being used and how expensive it is to synchronize them, enabling users to understand which loops can be parallelized and what the obstacles are to parallelizing them.

The Performance Analyzer also handles multiple concurrent processes to collect system-wide performance data, providing application insights down to the OS kernel, presenting a graphical identification of bottlenecks and helping fully optimize application performance.

Tune Multinode (MPI) Applications

The Performance Analyzer also provides the ability for developers to visualize the performance of distributed MPI-based applications via intuitive graphs that reveal performance hotspots in clustered environments. Basic application profiling data is available, as are various MPI message statistics, such as message length, internode message count, and message latency. In addition, the Performance Analyzer supports hybrid OpenMP/MPI applications by collecting and analyzing data seamlessly on MPI processes that are themselves multithreaded using OpenMP.

Advanced Performance Metrics

The Performance Analyzer probes applications with very low overhead. It records a variety of important metrics as a series of events. Each metric includes event-specific data and a call stack, thread ID, CPU ID, and high-resolution time stamp. These metrics are then attributed to functions, source lines, or (disassembled) instructions.

- **Clock-Based Profiling:** Statistical in nature and collects information on kernel accounting microstates and supports the user CPU time metric.
- **Hardware Counter (HWC) Overflow Profiling:** Also statistical in nature and is supported with special registers in hardware that count specific hardware events. The most useful counters indicate the behavior of the memory (cache) subsystems. The latest multicore systems contain 200+ hardware counters and are fairly extensive.
- **Dataspace Extensions to HWC Profiling:** Enables users to determine which data structures are responsible for cache misses.
- **Thread Synchronization Delay Tracing:** Provides an understanding of the efficiency of multithreading in a program.
- **Heap Tracing:** Identifies memory leaks or inefficient allocations in user code.
- **MPI: Tracing:** Identifies causes of communication delays in MPI applications.
- **Samples and Execution Statistics:** Allows users to sample their application at specific points in time. The execution statistics provide a global view of the program including microstate accounting data, time stamps, and kernel monitors.
- **Kernel Profiling:** Provides information on how the operating system kernel interacts with the application, using Oracle Solaris DTrace technology.
- **Java Profiling:** Seamlessly combines and reconciles two call stacks (the Java stack and the machine stack) into a single view. Java programs often contain Java code mixed with C/C++ native code as well as Hotspot compiled methods.

Easy-to-Use Graphical Interface

The Performance Analyzer identifies application performance bottlenecks by specifying not only which functions, code segments, and source lines are having an impact on performance but also by providing an easy-to-use GUI to tune for optimal performance. From annotated compiler commentary listings, in which the compiler indicates a range of information, to optimization status and runtime thread performance, users can visualize performance hotspots with the feature-rich GUI.

The GUI provides the following views to help you dig deeper into your applications behavior, helping you understand any use-case and workflow: Functions, Callers-Callees, Call Tree, Source, Disassembly, and Timeline.

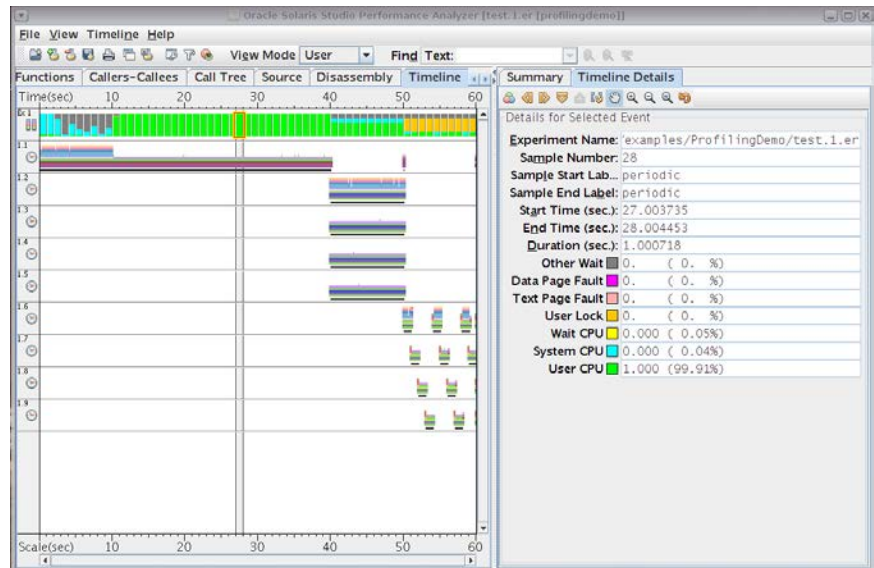


Figure 1: Visualize multithreading application performance in the Timeline view.

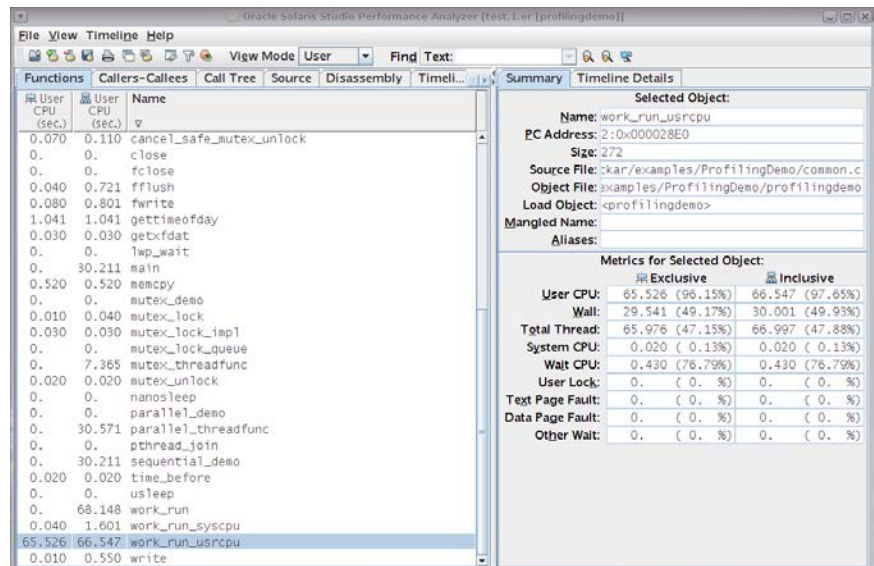


Figure 2: Understand which functions are taking the most time in the Functions view.

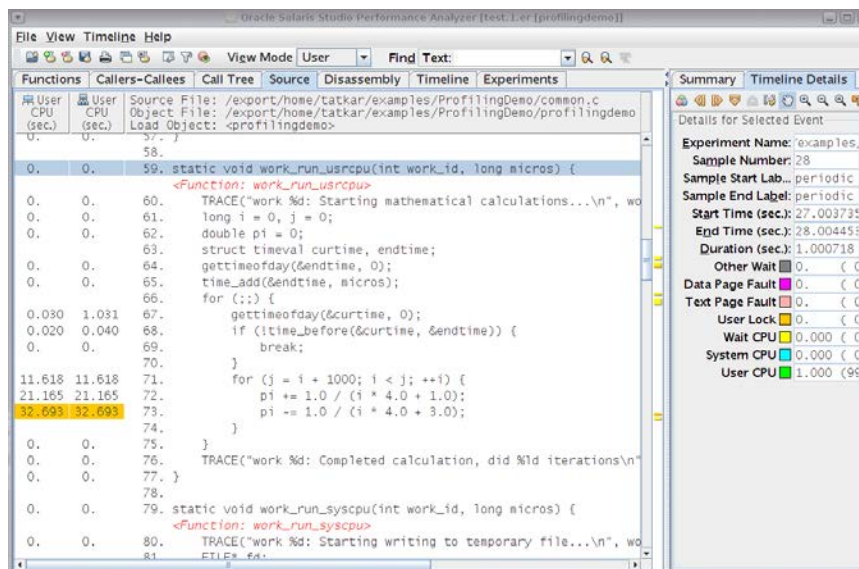


Figure 3: View profile data on your source code in the Source view, and also view profile data on your assembly code.

Since its introduction in the mid-1990s, the Performance Analyzer has been delivering leading-edge functionality and has been the first to market with many features, including microstate accounting, microstate profiling, dataspace profiling, mixed Java/C++ and Hotspot-optimized methods in one profile, MPI state profiling, and user-model call stack and OMP state tracing in OpenMP profiling.

Contact Us

For more information about Oracle Solaris Studio, visit oracle.com/goto/solarisstudio or call +1.800.ORACLE1 to speak to an Oracle representative.

 Oracle is committed to developing practices and products that help protect the environment

Copyright © 2005, 2010, 2011 Oracle and/or its affiliates. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark licensed through X/Open Company, Ltd. 0611

Hardware and Software, Engineered to Work Together