ORACLE DATA SHEET

Oracle Developer Studio Performance Analyzer

The Oracle Developer 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 performing optimally? Are you taking full advantage of your high-throughput multi-core systems? The Performance Analyzer is a powerful market-leading profiler for optimizing application performance and scalability. It provides both source-level and machine-specific analysis that enables you to drill-down on application behavior, allowing you to quickly isolate hotspots and eliminate bottlenecks.

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 and Scala that supersedes other tools' accuracy and machine-level observability. Identification of a hot call path is streamlined with an interactive, graphical call tree called a flame graph. After a hot function is identified, navigating call paths while viewing relevant source is performed with single mouse-clicks. At the source level, bottlenecks are quickly identified by the performance metrics shown at each line of source, Java bytecode, and machine disassembly. For deep drill-down, Analyzer describes which optimizations have been applied to lines of C/C++/Fortran, and shows how Java was transformed by Hotspot compilation. In mixed Java/native environments, Performance Analyzer seamlessly handles calls to and from native code.

The Performance Analyzer can be used to profile fully optimized single-threaded and multi-threaded applications written using pthreads, Oracle Solaris threads, and OpenMP. As systems evolve to support hundreds or thousands of hardware threads, scalability and resource contention are emerging as the critical factor. The Performance Analyzer highlights how threads are being used and how expensive it is to synchronize them, helping users to understand which loops can be parallelized.

In addition, on Oracle Solaris, the Performance Analyzer also profile system wide activity, including the kernel impact on the system as a whole.

Advanced Performance Metrics

KEY FEATURES

- Low overhead for fast and accurate results
- Advanced profiling of serial and parallel applications
- Rich set of performance data and metrics
- Easy to use GUI
- Remote analysis to simplify cloud development
- Supports C, C++, Java, Fortran and Scala

KEY BENEFITS

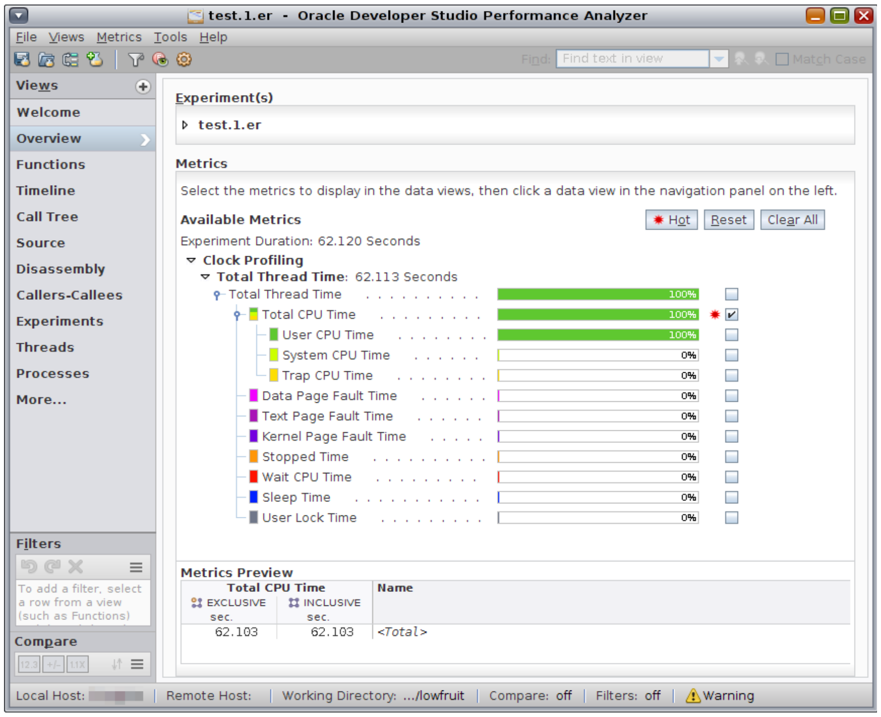
- Maximize application performance
- Improve system utilization and software quality
- Increase developer productivity

The Performance Analyzer provides a rich set of data and metrics for diagnosing a wide variety of performance problems and 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 timestamp. These metrics are then attributed to functions, source lines or (disassembled) instructions:

- **Clock-based Profiling:** Statistical in nature, this metric collects information on kernel accounting microstates and supports the user CPU time metric.
- **Hardware Counter (HWC) Overflow Profiling:** Also statistical in nature, this metric 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 multi-core systems contain 200+ hardware counters and are fairly extensive.
- **Dataspace and Memoryspace Extensions to HWC Profiling:** Dataspace profiling enables users to determine which data structures are responsible for cache misses. Memoryspace profiling allows users to find hot cache lines, and understand which data addresses, threads, and instructions are making them hot. On machines such as SPARC T7, with precise HW counter interrupts for memory counters, no special compilation is needed, and memoryspace profiling allows the detection of false sharing of cache lines.
- **Thread Synchronization Delay Tracing:** Provides an understanding of the efficiency of multithreading in a program.
- **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. User processes can also be simultaneously profiled along with the kernel. Kernel profiles can be based on either clock-ticks or HWC-overflow events.
- **Java Profiling:** Java programs often contain Java code mixed with C/C++ native code as well as Hotspot compiled methods. This metric seamlessly combines and reconciles two callstacks, the Java stack and the machine stack, into a single view.
- **Heap Tracing:** Tracks Heap (malloc, calloc, free) usage and identifies memory leaks or inefficient allocations in user code.
- **I/O Tracing:** Allows you to identify I/O patterns in your application and quickly pinpoint I/O bottlenecks that may be impacting application performance.
- **Cycles per Instruction (CPI) / Instructions per Cycle (IPC) Tracing:** Provides metrics to track CPI and IPC to help you identify where your application is running efficiently or inefficiently. These metrics are available when you perform hardware counter profiling on your application and specify the counters for cycles and instructions.

Intuitive and 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 by also providing an easy-to-use GUI to tune for optimal performance. The GUI provides advanced sorting, filtering and timeline visualization capabilities for rapid identification of performance bottlenecks. The Performance Analyzer GUI is organized around data views that each show a different perspective of the performance metrics for your profiled application and allow you to compare experiment data collected from different runs.



Select the metrics to display in the data views, then click a data view in the navigation panel on the left.

Experiment Duration: 62.120 Seconds

Clock Profiling

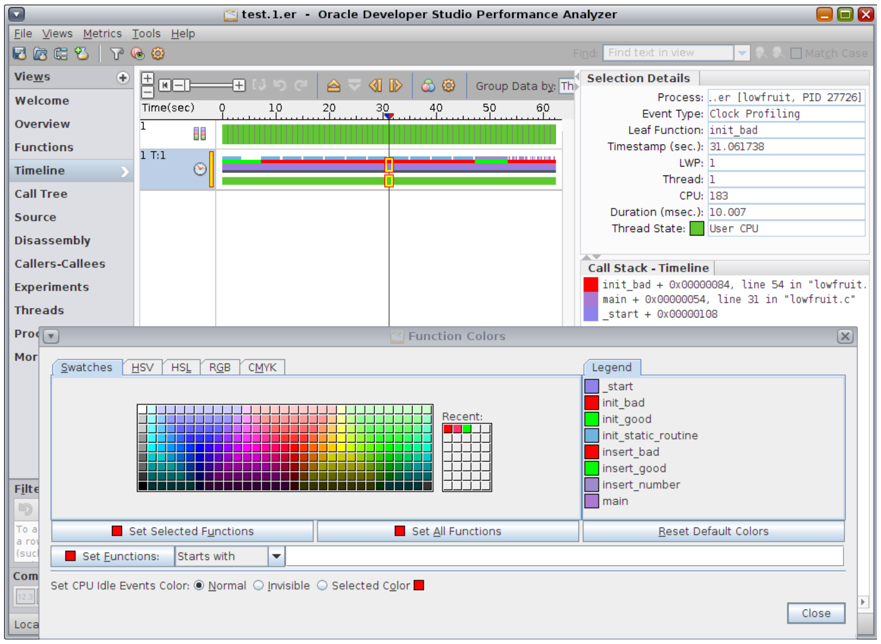
Total Thread Time: 62.113 Seconds

Metric		Selected
Total Thread Time	100%	☐
Total CPU Time	100%	✓
User CPU Time	100%	☐
System CPU Time	0%	☐
Trap CPU Time	0%	☐
Data Page Fault Time	0%	☐
Text Page Fault Time	0%	☐
Kernel Page Fault Time	0%	☐
Stopped Time	0%	☐
Wait CPU Time	0%	☐
Sleep Time	0%	☐
User Lock Time	0%	☐

Total CPU Time		Name
EXCLUSIVE sec.	INCLUSIVE sec.	
62.103	62.103	<Total>

Local Host: | Remote Host: | Working Directory: .../lowfruit | Compare: off | Filters: off | Warning

Figure 1. Overview Screen shows a summary of performance metrics for the experiment



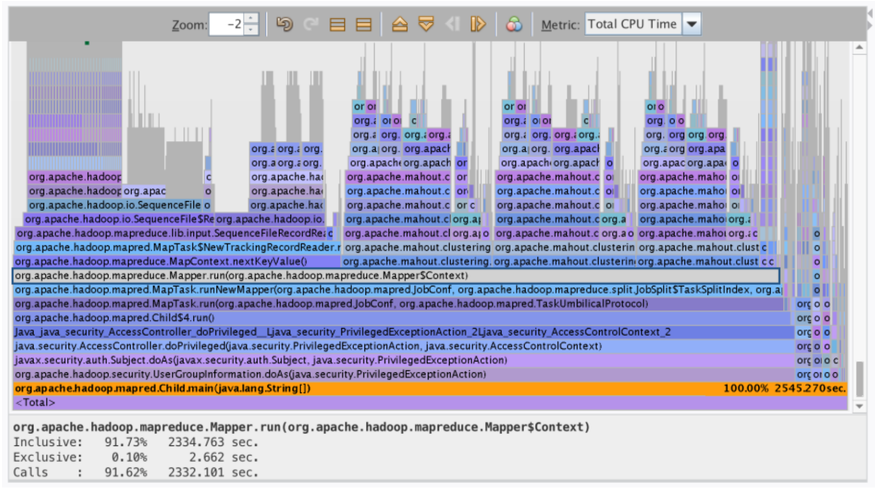
Process:	..er [lowfruit, PID 27726]
Event Type:	Clock Profiling
Leaf Function:	init_bad
Timestamp (sec.):	31.061738
LWP:	1
Thread:	1
CPU:	183
Duration (msec.):	10.007
Thread State:	User CPU

```
init_bad + 0x00000084, line 54 in "lowfruit.
main + 0x00000054, line 31 in "lowfruit.c"
_start + 0x00000108
```

Legend: _start, init_bad, init_good, init_static_routine, insert_bad, insert_good, insert_number, main

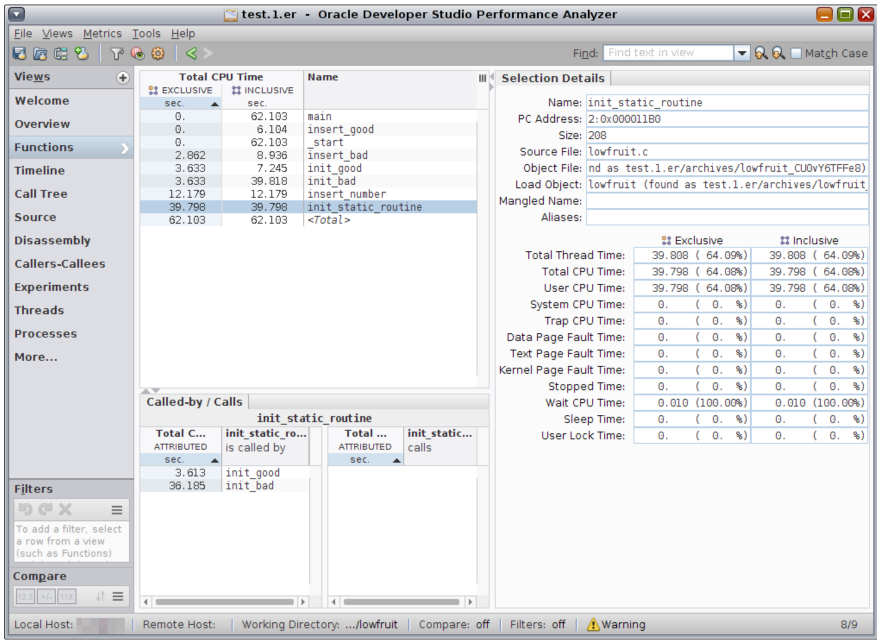
Set CPU Idle Events Color: [✓ Normal] Invisible Selected Color

Figure 2. Timeline View allows you to visualize performance hotspots, zoom in and filter



```
org.apache.hadoop.mapreduce.Mapper.run(org.apache.hadoop.mapreduce.Mapper$Context)
Inclusive:   91.73%   2334.763 sec.
Exclusive:    0.10%      2.662 sec.
Calls    :   91.62%   2332.101 sec.
```

Figure 3. Flame Graph View provides visualization and identification of the most frequent code paths



Total CPU Time EXCLUSIVE sec.	Total CPU Time INCLUSIVE sec.	Name
0.	62.103	main
0.	6.104	insert_good
0.	62.103	_start
2.862	8.936	insert_bad
3.633	7.245	init_good
3.633	39.818	init_bad
12.179	12.179	insert_number
39.798	39.798	init_static_routine
62.103	62.103	<Total>

Called-by / Calls

init_static_routine

Total C... ATTRIBUTED sec.	init_static_ro... is called by
3.613	init_good
36.185	init_bad

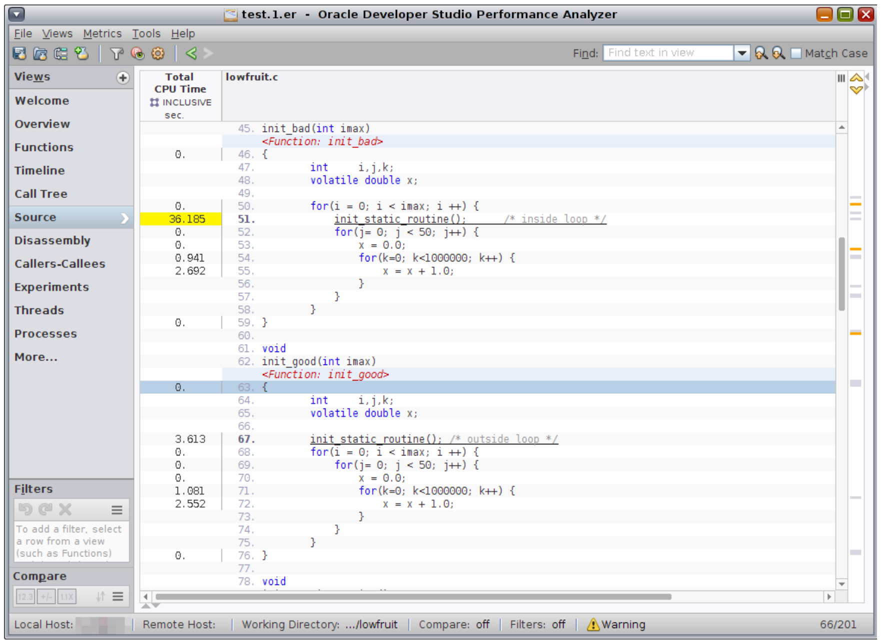
Total ... ATTRIBUTED sec.	init_static... calls

Selection Details

Name:	init_static_routine
PC Address:	2:0x000011B0
Size:	208
Source File:	lowfruit.c
Object File:	nd as test.1.er/archives/lowfruit_CU0vY6TFFe8)
Load Object:	lowfruit (found as test.1.er/archives/lowfruit
Mangled Name:	
Aliases:	

	Exclusive	Inclusive
Total Thread Time:	39.808 (64.09%)	39.808 (64.09%)
Total CPU Time:	39.798 (64.08%)	39.798 (64.08%)
User CPU Time:	39.798 (64.08%)	39.798 (64.08%)
System CPU Time:	0. (0. %)	0. (0. %)
Trap CPU Time:	0. (0. %)	0. (0. %)
Data Page Fault Time:	0. (0. %)	0. (0. %)
Text Page Fault Time:	0. (0. %)	0. (0. %)
Kernel Page Fault Time:	0. (0. %)	0. (0. %)
Stopped Time:	0. (0. %)	0. (0. %)
Wait CPU Time:	0.010 (100.00%)	0.010 (100.00%)
Sleep Time:	0. (0. %)	0. (0. %)
User Lock Time:	0. (0. %)	0. (0. %)

Figure 4. Functions View helps you understand which functions are taking the most time



```
  Total       lowfruit.c
 CPU Time
INCLUSIVE
   sec.
              45. init_bad(int imax)
                  <Function: init_bad>
     0.       46. {
              47.         int     i,j,k;
              48.         volatile double x;
              49.
     0.       50.         for(i = 0; i < imax; i ++) {
 36.185       51.             init_static_routine();      /* inside loop */
     0.       52.             for(j= 0; j < 50; j++) {
     0.       53.                 x = 0.0;
     0.941    54.                 for(k=0; k<1000000; k++) {
     2.692    55.                     x = x + 1.0;
              56.                 }
              57.             }
              58.         }
     0.       59. }
              60.
              61. void
              62. init_good(int imax)
                  <Function: init_good>
     0.       63. {
              64.         int     i,j,k;
              65.         volatile double x;
              66.
     3.613    67.         init_static_routine(); /* outside loop */
     0.       68.         for(i = 0; i < imax; i ++) {
     0.       69.             for(j= 0; j < 50; j++) {
     0.       70.                 x = 0.0;
     1.081    71.                 for(k=0; k<1000000; k++) {
     2.552    72.                     x = x + 1.0;
              73.                 }
              74.             }
              75.         }
     0.       76. }
              77.
              78. void
```

Local Host: | Remote Host: | Working Directory: .../lowfruit | Compare: off | Filters: off | Warning 66/201

Figure 5. Source View allows you to profile data on your source code, also includes syntax highlighting based on the source language and hyperlinks for the caller and callee functions

Remote and Cross-Platform Analysis

The Performance Analyzer provides support for remote analysis, allowing you to profile applications, collect performance data, and view experiments on a remote server from a Linux, Windows, MAC or Solaris client environment. In addition, ssh tunneling support allows remote performance analysis of applications on Compute VMs in cloud-based development environments. You can also read experiments recorded on any platform with cross-architecture analysis support. The Performance Analyzer is a powerful and robust application profiling tool with industry-leading functionality to help you efficiently optimize enterprise applications for maximum performance and scalability.

CONTACT US

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

CONNECT WITH US

- blogs.oracle.com/oracle
- facebook.com/oracle
- twitter.com/oracle
- oracle.com

Integrated Cloud Applications & Platform Services