Metrics for Success: Performance Analysis 101

February 21, 2008

Kuldip Oberoi
Developer Tools
Sun Microsystems, Inc.

Agenda

- Application Performance
- Compiling for performance
- Profiling for performance
- Closing remarks

Sun Studio software

C/C++/Fortran tooling for the multi-core era

- Adoption increased 100+% over 2 years. Large footprint in enterprise/technical accts & growth in open src usage
- #1 IDE in the Evans survey for Performance of Resulting Applications category
- Intel, AMD, Sun & Fujitsu Partnerships

SUN STUDIO SHINES

Sun Studio software has emerged as a leading compiler and tool for application development on Solaris and Linux operating systems. Now, it's rising to a new level of capability and performance.

The latest release of Sun Studio 12 software accelerates application performance while reducing time-to-market through parallelizing C, C++, and Fortran compilers and tools for Solaris and Linux platforms running on x86 and x64 platforms. A free application, this software uses the processing power in AMD processor-based systems to provide enhanced performance for applications – thus realizing outstanding results from multithreaded applications.

Sun Studio 12 software uses innovative optimization techniques on x86 systems to provide up to 25 percent better performance on compute-intensive tasks and up to 80 percent better performance for memory-intensive, industry-recognized benchmarks than open source alternatives.

- Parallelism – feature-rich toolchain (auto-parallelizing compilers, thread analysis / debugging / profiling, OpenMP support, ...) & MPI support via Sun HPC ClusterTools
- Performance – dozens of industry benchmark records in the past year over Intel, AMD, Sun, & Fujitsu architectures
- Productivity – NetBeans-based IDE, code & memory debuggers, application profiler
- Platforms – Simplified dev across architectures & OSs (Solaris OS, OpenSolaris OS, Linux)

Sun Studio Software Overview

FREE

Integrated Toolchain

- Record-setting parallelizing C/C++/Fortran Compilers with autopar
- NetBeans-based IDE
- Stable, Scriptable, Multilingual Debugger (dbx)
- Memory Debugger- leak, access, usage (RTC)
- Application Profiling Tools (Performance Analyzer)
- Multi-core Optimizations, Multithreaded High Performance Libraries
- OpenMP API Support
- Multithreading Tools- Thread Analysis

http://developers.sun.com/sunstudio

Agenda

- Application Performance
- Compiling for performance
- Profiling for performance
- Closing remarks

Why Care about Performance?

- Because your company cares
  - Faster code => greater productivity => lower cost
- Because your peers and customers care
  - Better performance => less HW => lower cost
- Because it's interesting and suprising
  - Coding is based on assumptions about behavior
  - Performance problems arise from disconnects
- Because it's easy to do
  - Simple runs, automated runs

What's a Performance Problem?

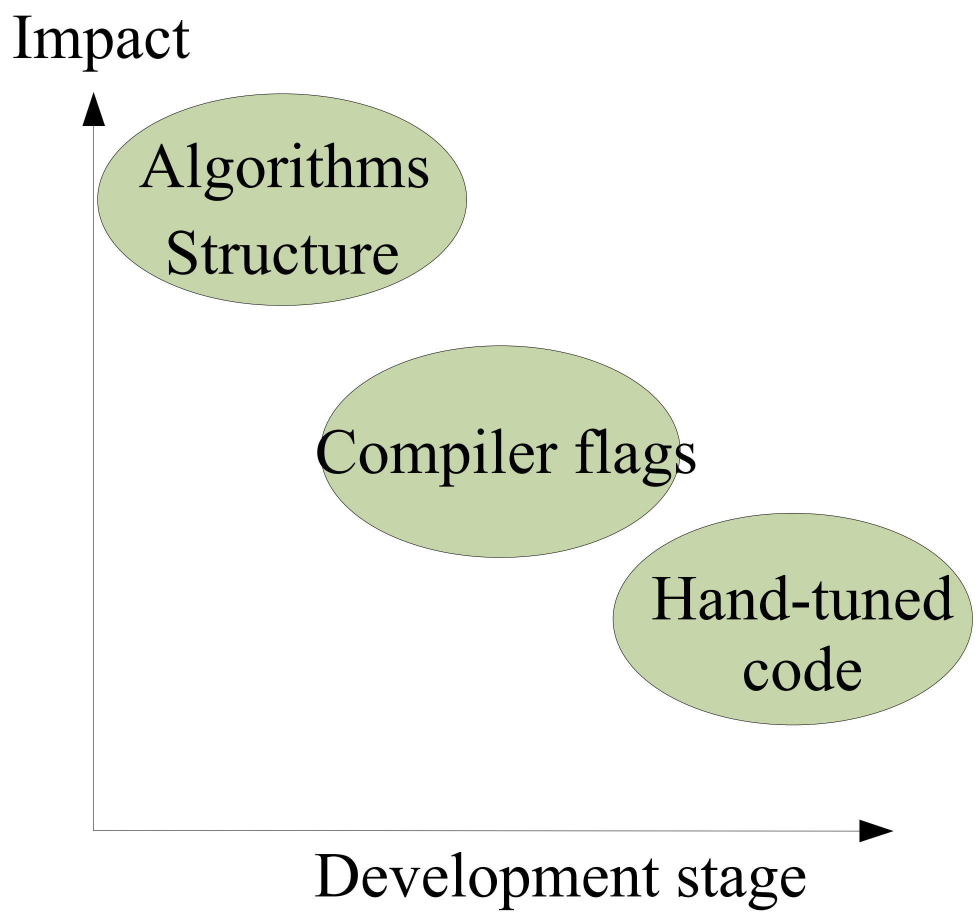
- Subjective criteria:
  - > It takes too long to finish
  - > It responds too slowly
- Objective critera:
  - > It can't handle the required load
  - > It consumes too many resources to do its work
- Is it worth fixing?
  - > Cost of fixing vs. aggregate cost of problem
- Most untuned codes have low-hanging fruit!

Where does performance come from?

- Not the real question...
  - Every application has a maximum theoretical performance
  - Design decisions and implementation can deliver lower actual performance
- Alternative: *Where has performance been lost?*
  - Need to identify where the time is spent
  - Then determine what can be done about it

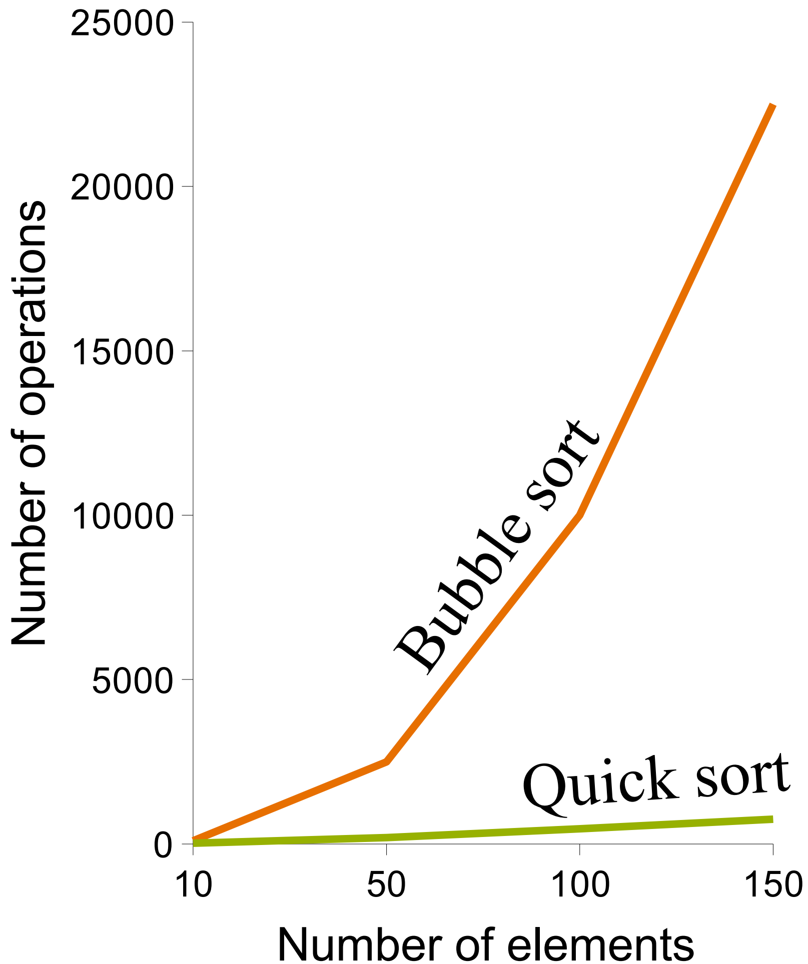
Where performance goes to

- Performance opportunities:
  - > Algorithms
  - > Structure
  - > Compiler flags
  - > Hand-tuned code



Algorithmic complexity

- How many operations?
- Classic example:
  - > Bubble sort $O(n^2)$
  - > Quick sort O(nlog(n))
- Sun Studio libraries have optimized code
  - > perflib (BLAS, FFTs, etc)
  - > medialib (images, codecs)
  - > Optimized maths libraries



Compiler flags

- The compiler's job:
  - > Produce the best code
  - > Given little knowledge of developer's intent
  - > Best code regardless of coding style
- Increasing optimization
  - > Leads to improved performance
  - > Relies on standard conforming source code
- Does a better job with
  - > Visibility of more of the code
  - > More information about the intent of the developer

Hand-tuned code

- Examples:
  - Special case code for the common situation
  - Assembly language versions of key routines
- Cons:
  - Time consuming
  - Inflexible (e.g. workload specific)
  - Platform specific
- Good examples:
  - Use of language standards to provide compiler with more information (e.g. `restrict` keyword)

Perspective on optimization

- Algorithms, structures, and compiler flags
  - High-level change
  - Useful for all platforms
  - Improve the application for all workloads
- Tweaks, hand-code
  - Low-level (localised) change
  - Platform specific
  - Often already done by the compiler
- So:
  - Focus on the high-level
  - Unless the low-level gains are clear

Methodology / Tools Used

Ensure Best Builds:
Latest Compiler
Optimization flags
Profile feedback
Insert #pragmas

Check Libraries Used:
optimized math libs
libsunperf
medialib
Write special routines?

Identify Hot Spots:
gprof(function timings)
tcov(line counts)
analyzer (many stats)

Get Execution Stats:
cputrack(perf counters)
locstat(lock containment)
trapstat(traps)

Study and rewrite Source as appropriate

Study and rewrite assembly as appropriate

Agenda

- Application Performance
- Compiling for performance
- Profiling for performance
- Closing remarks

Choosing compiler flags

- Use the flags
  - > That you understand
  - > That you need (i.e. make a difference)
- Don't use flags
  - > That you don't understand
  - > That don't have an impact

Optimization

- Rule:
  - No optimization flags means no optimization
- Suggestions:
  - Use at least **-O**
  - Try **-fast**
- Notes:
  - Compile and link with the same flags

Debug information

- **Always** generate debug information
  - `-g` for C/Fortran
  - `-g0` for C++
- Also useful for profiling
- No/minimal performance impact

Exploring -fast

- `-fast` is a macro-flag:
  - Enables a number of potentially useful optimisations
  - May not be suitable for all situations
- Assumes build machine = run machine
  - Use `-xtarget=` to specify otherwise
- Enables floating point simplification
  - Use `-fsimple=0 -fns=no` otherwise
- Assumes basic pointer types do not alias
  - Use `-xalias_level=any` otherwise
- Flags are parsed from left to right
  - Override by placing flags on right

Target hardware

- Not all processors implement the same instructions
  > Application will not run if instructions are not implemented
- If build machine is machine that will run binary:
  > **`-xtarget=native`**
- For binaries that will run on a wide range of machines:
  > SPARC: **`-xtarget=generic -xarch=sparcvis2`**

  > x86: **`-xtarget=generic -xarch=sse2`**

Instruction set extensions (SSE/x86)

- Instruction set extensions on x86
- Single Instruction Multiple Data (SIMD)
  - e.g. two parallel add operations in a single instruction
- Enable generation with:
  - **`-xtarget=generic -xarch=sse2 -xvector=simd`**

Target hardware 32-bit or 64-bit

- 32-bit (`–m32`) can address 4GB of memory
- 64-bit (`–m64`) can address >>4GB of memory
- 64-bit: pointers and longs are 8 bytes
  - => larger memory footprint
  - => slower
- x86 64-bit has
  - More registers, Better ABI
  - => faster
  - For x86 64-bit faster except when dominated by increased memory footprint

Inlining and cross-file optimisation

- Inlining:
  - Avoids call overhead
  - Provides more opportunities to code optimisation
  - Increases code size
- Within file inlining at **`-xO4`**
- Crossfile inlining at **`-xipo`**
  - Inlines between source files
  - Reduces impact of source code structuring
  - Can increase compile time

Profile feedback

- Profile feedback enables the compiler
  - To see the runtime behavior of the application
  - To make better code layout decision
  - To make the right inlining decisions
- Three step process:
  - Compile with **`-xprofile=collect:/dir/profile`**
  - Run with training workload
  - Compile with **`-xprofile=use:/dir/profile`**
- Very useful for applications containing lots of decision logic

Agenda

- Application Performance
- Compiling for performance
- Profiling for performance
- Closing remarks

Good practices

- **Always** profile your application
  - > Is the time being spent in the important code?
  - > Are there obvious hot-spots to improve?
  - > How does the profile change with the workload?
- Amdahl's law
  - > Limit on performance gain is the time spent in the slow code
- Fix performance issues
  - > But make the fixes at the highest possible level of abstraction

Why use Sun Studio Tools? (I)

- The work for production code, production runs
  - Runs from tens of seconds through hours
- They measure real behavior
  - Fully optimized and parallelized applications
  - Java HotSpot enabled
- They have minimal dilation and distortion
  - ~5% for typical apps, ~10% for Java apps
- Supports code compile with Sun Studio & GNU compilers
- They're ***FREE*** for Solaris & Linux

Why use Sun Studio Tools? (II)

- They make things as simple as possible
  - Show data in the user's source model
  - Including OpenMP, MPI, Java, threads, etc.
- ..., but no simpler
  - Show exactly what the compiler did
    - Inlines, outlines, clones, parallel routines
  - Show what JVM did
    - Interpreted methods, HotSpot-compiled methods
    - GC & HotSpot-compiler activities

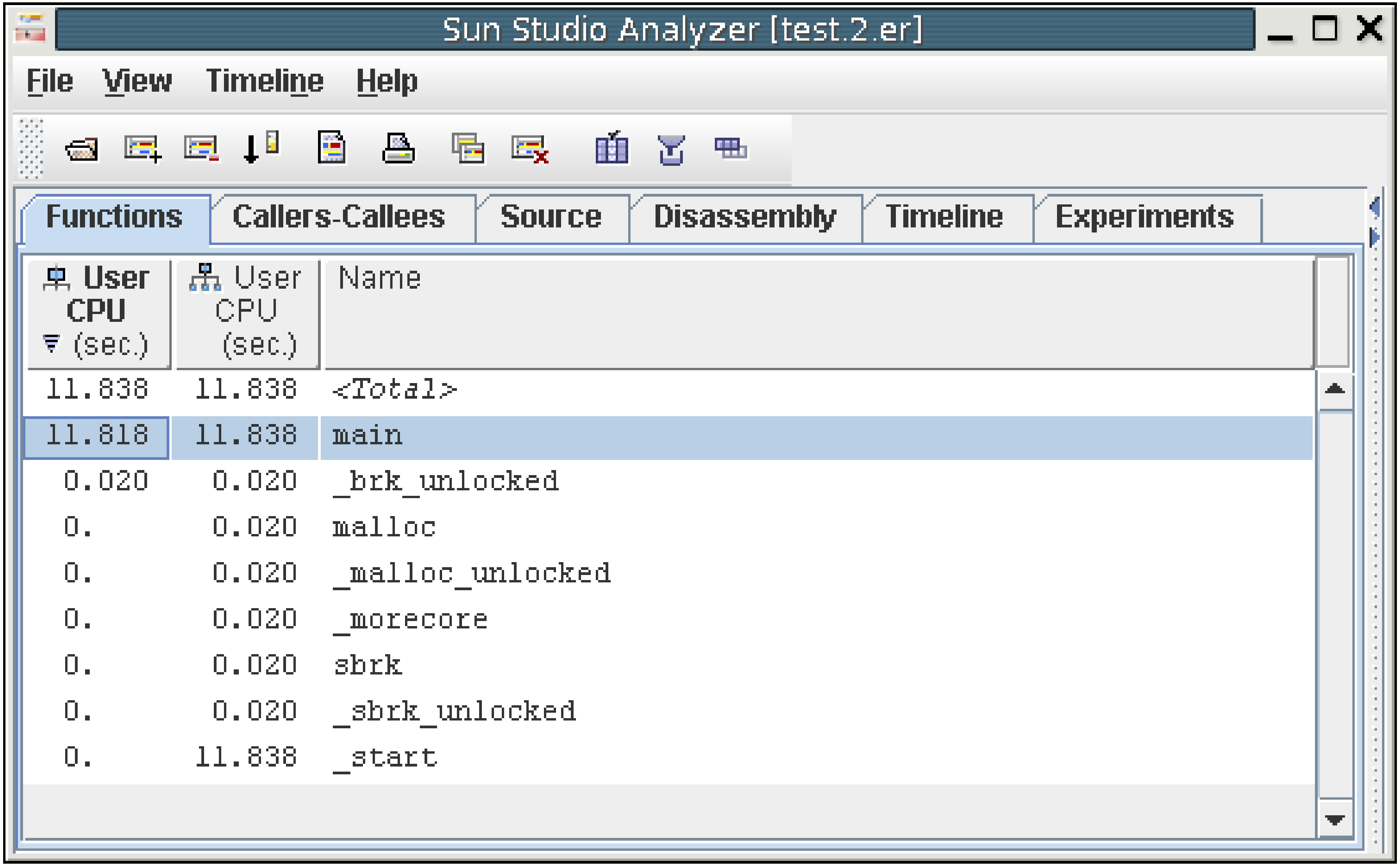
Questions

- What can I change to improve perforance?
- Which resources are being used?
- Where are they being used?
- Single-threaded
  - > Is the CPU being used efficiently?
  - > Memory subsystem delays? (TLBs, caches)
  - > I/O subsystem problems? (disk, network, paging)
- Multi-threaded
  - > Similar to single-threaded &
  - > Load unbalanced? Lock contention? memory/cache contention?

Gathering profiles

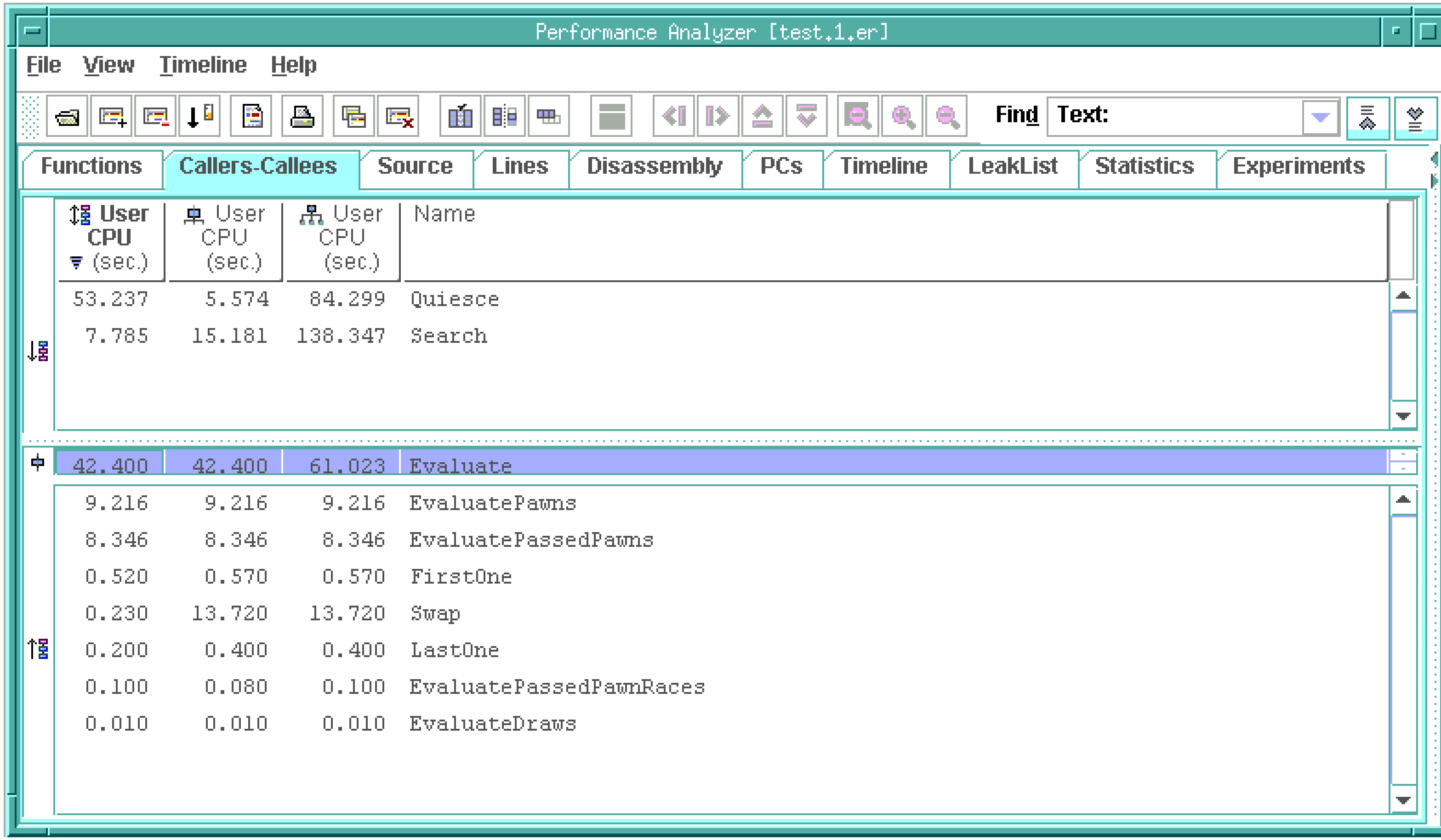
- Use **-g**/**-g0** for attribution of time to source line
- Gather profile with:
  - > **collect** <app> <params>
  - > **collect -P** <pid>
- Analyse profile with:
  - > **analyzer** test.<N>.er
  - > **er_print** test.<N>.er

Application profile

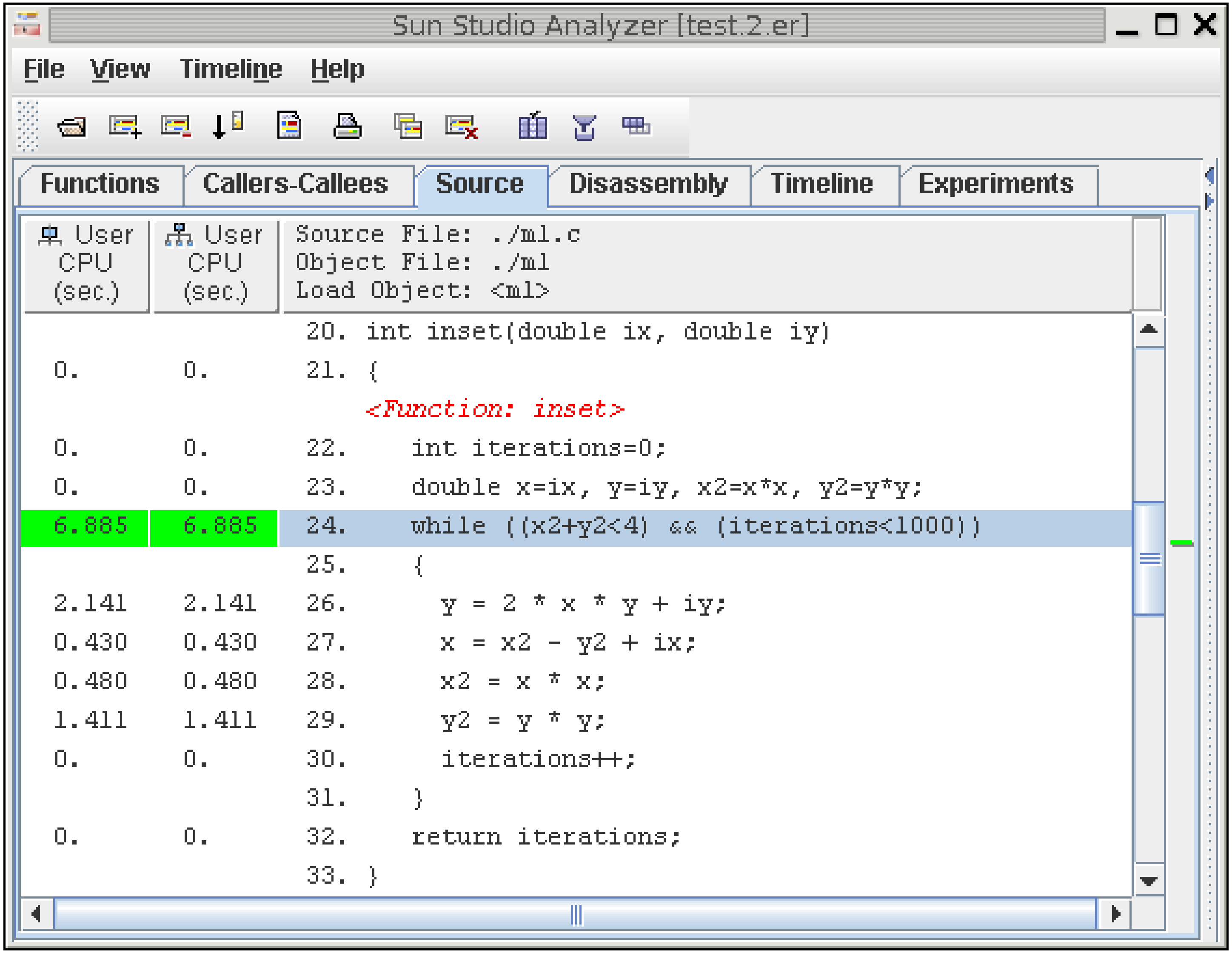


User CPU (sec.)	User CPU (sec.)	Name
11.838	11.838	<Total>
11.818	11.838	main
0.020	0.020	_brk_unlocked
0.	0.020	malloc
0.	0.020	_malloc_unlocked
0.	0.020	_morecore
0.	0.020	sbrk
0.	0.020	_sbrk_unlocked
0.	11.838	_start

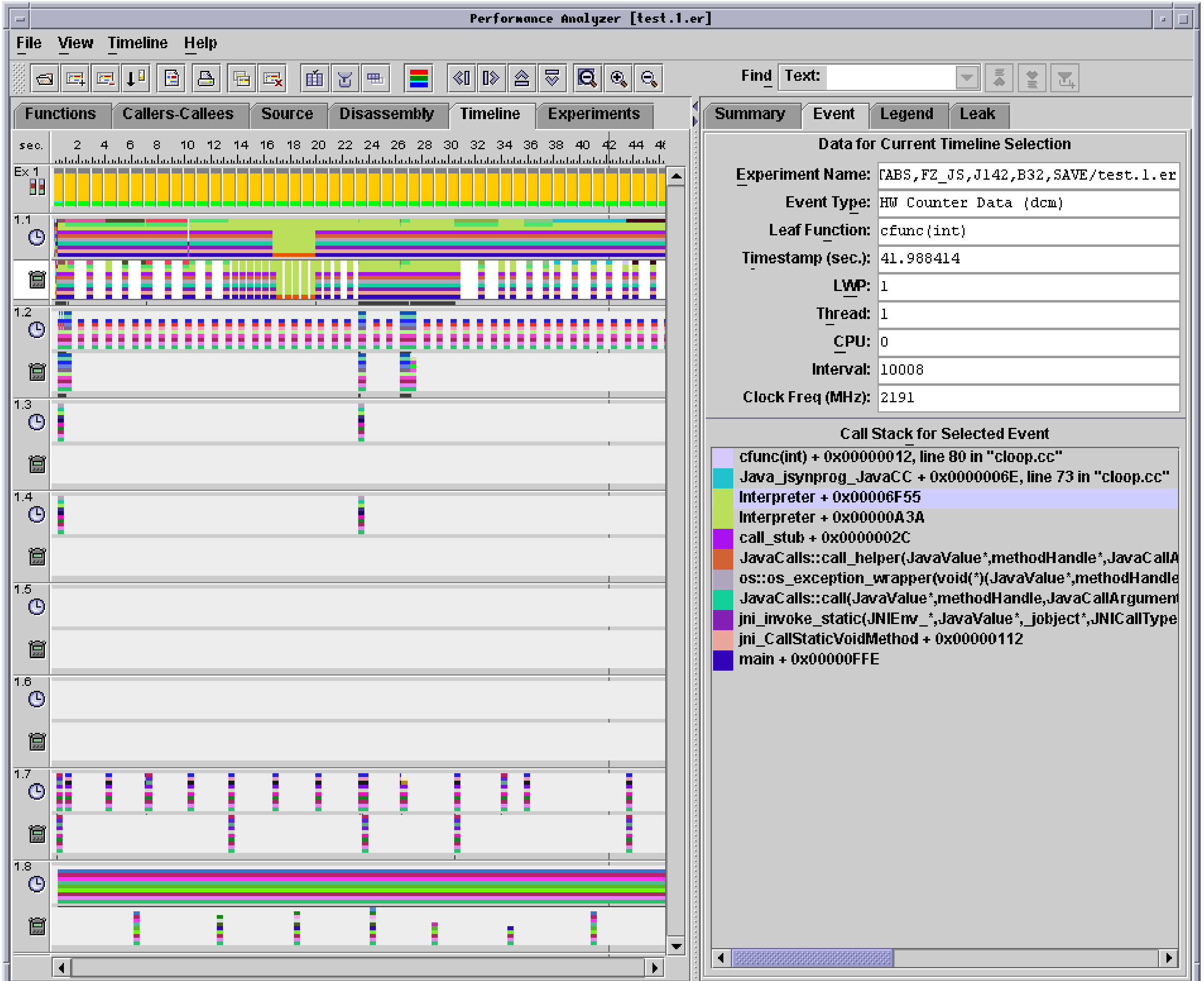
Caller-Callee



Source level profile



Performance Analyzer - Timeline



DEMO

Agenda

- Application Performance
- Compiling for performance
- Profiling for performance
- Closing remarks

The checklist

- Build with optimization
  - At least **-O**
- Build with debug enabled
  - **-g** (**-g0** for C++)
- Profile
  - **collect** `<app> <params>`

Optimization: Increasing optimisation

- Increased optimization (**`-fast`**)
  - Typically improved performance
  - Be aware of the optimisations enabled
- Use crossfile optimization (**`-xipo`**)
  - Typically good for all codes

Optimization: Increasing information

- Profile feedback to give more information
  - > `-xprofile=[collect:|use:]`
  - > Good for all codes
  - > Particularly helpful for inlining and branches

Optimization: Leveraging libraries

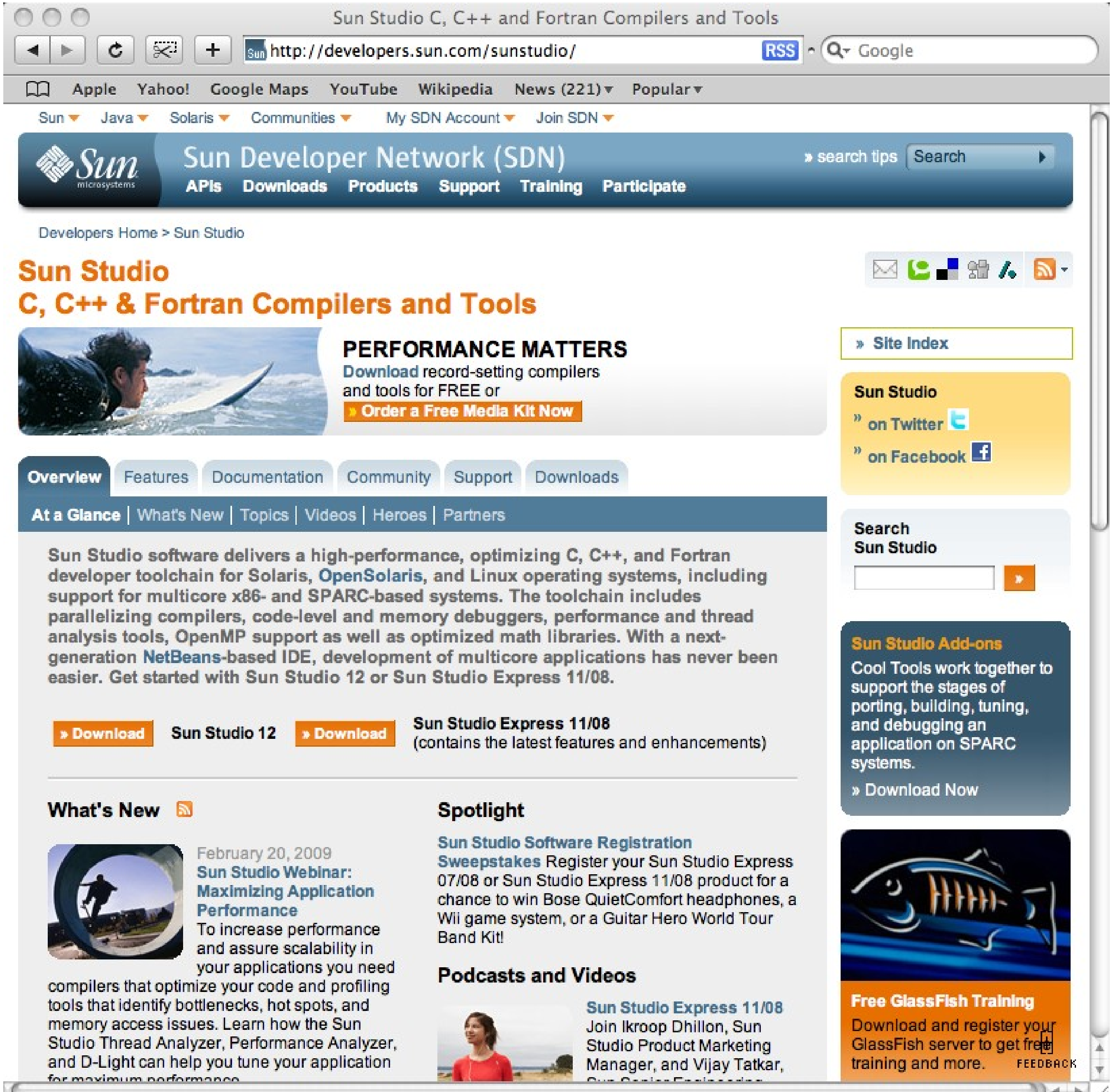
- If time is spent in library code:
  - Supplied optimized maths functions
    `-xlibmil -xlibmopt`
  - Optimized STL for C++
    `-library=stlport4`
  - The performance library
    `-library=sunperf`

Summary

- Always profile
- Always use optimization

Gains from Tuning Categories

Tuning Category	Typical Range of Gain
Source Change	*25-100%*
Compiler Flags	*5-20%*
Use of libraries	*25-200%*
Assembly coding / tweaking	*5-20%*
Manual prefetching	*5-30%*
TLB thrashing/cache	*20-100%*
Using vis/inlines/micro-vectorization	*100-200%*


Sun Studio C, C++ and Fortran Compilers and Tools
http://developers.sun.com/sunstudio/
RSS
Google
Apple Yahoo! Google Maps YouTube Wikipedia News (221) Popular
Sun Java Solaris Communities My SDN Account Join SDN
Sun Developer Network (SDN)
APIs Downloads Products Support Training Participate
» search tips Search
Developers Home > Sun Studio
Sun Studio
C, C++ & Fortran Compilers and Tools
PERFORMANCE MATTERS
Download record-setting compilers and tools for FREE or
» Order a Free Media Kit Now
» Site Index
Sun Studio
» on Twitter
» on Facebook
Overview Features Documentation Community Support Downloads
At a Glance | What's New | Topics | Videos | Heroes | Partners
Sun Studio software delivers a high-performance, optimizing C, C++, and Fortran developer toolchain for Solaris, OpenSolaris, and Linux operating systems, including support for multicore x86- and SPARC-based systems. The toolchain includes parallelizing compilers, code-level and memory debuggers, performance and thread analysis tools, OpenMP support as well as optimized math libraries. With a next-generation NetBeans-based IDE, development of multicore applications has never been easier. Get started with Sun Studio 12 or Sun Studio Express 11/08.
» Download Sun Studio 12
» Download Sun Studio Express 11/08
(contains the latest features and enhancements)
Search Sun Studio
Sun Studio Add-ons
Cool Tools work together to support the stages of porting, building, tuning, and debugging an application on SPARC systems.
» Download Now
What's New
February 20, 2009
Sun Studio Webinar: Maximizing Application Performance
To increase performance and assure scalability in your applications you need compilers that optimize your code and profiling tools that identify bottlenecks, hot spots, and memory access issues. Learn how the Sun Studio Thread Analyzer, Performance Analyzer, and D-Light can help you tune your application
Spotlight
Sun Studio Software Registration Sweepstakes Register your Sun Studio Express 07/08 or Sun Studio Express 11/08 product for a chance to win Bose QuietComfort headphones, a Wii game system, or a Guitar Hero World Tour Band Kit!
Podcasts and Videos
Sun Studio Express 11/08
Join Ikroop Dhillon, Sun Studio Product Marketing Manager, and Vijay Tatkar,
Free GlassFish Training
Download and register your GlassFish server to get free training and more.
FEEDBACK

Metrics for Success: Performance Analysis 101

Thanks!!

Kuldip Oberoi
koberoi@sun.com
http://koberoi.com