BRUDEN · OSSG
On-Shore Systems Group

Java performance on OpenVMS

Guy Peleg
Senior Member of the Technical Staff
Director of EMEA Operations
guy.peleg@bruden.com

Breaking the Myth

Java + OpenVMS = Great Performance ??

Sind Sie verrückt?

Yes….but it's not really related

Breaking the Myth

- Performance & Tuning Guidelines
  - **OpenVMS V8.3 running on IA64**
  - **Tune your system**
  - **Java 5**
  - **Tune the Java environment**
  - **Profile your application**

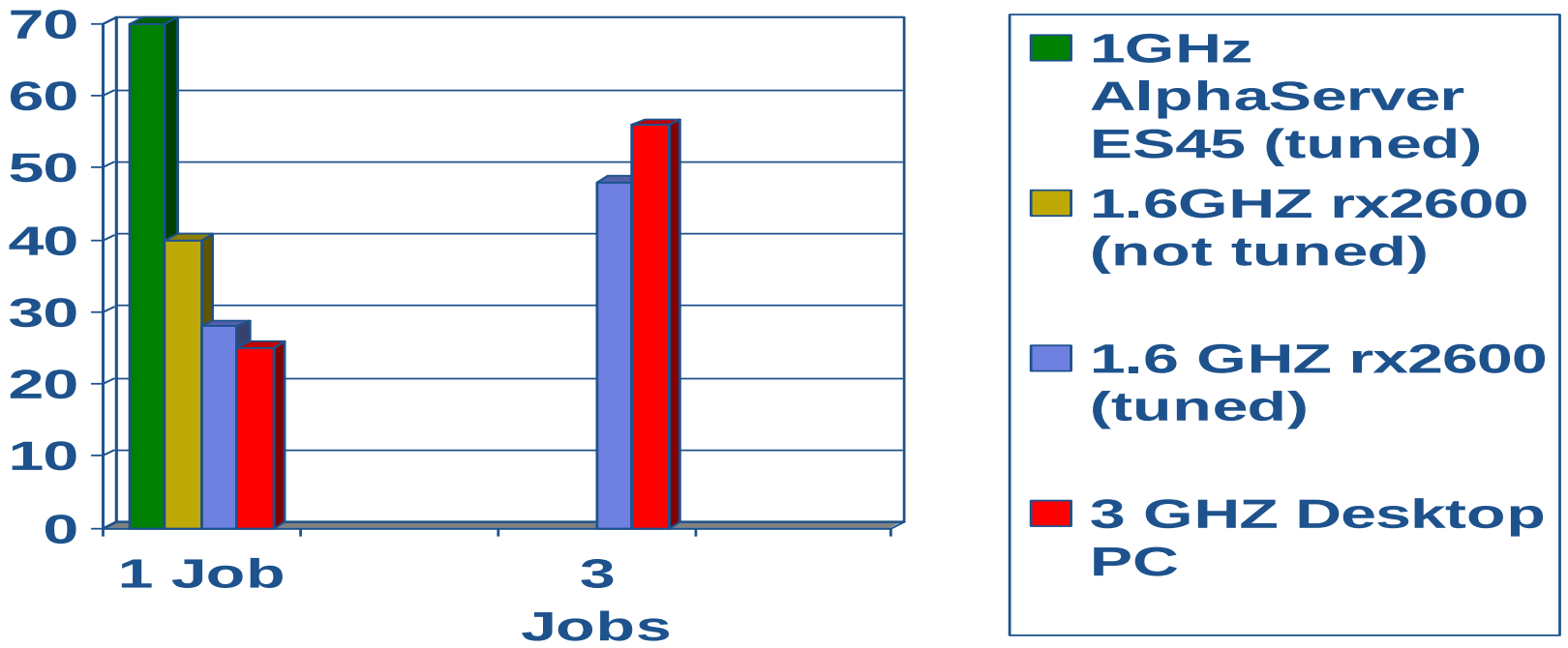
Important Notice

All the tests and benchmarks in this presentation have been executed on an 1.3Ghz rx2600, faster systems available (rx3600, rx6600)

Java Benchmark

- Batch processing application
- 100% pure Java

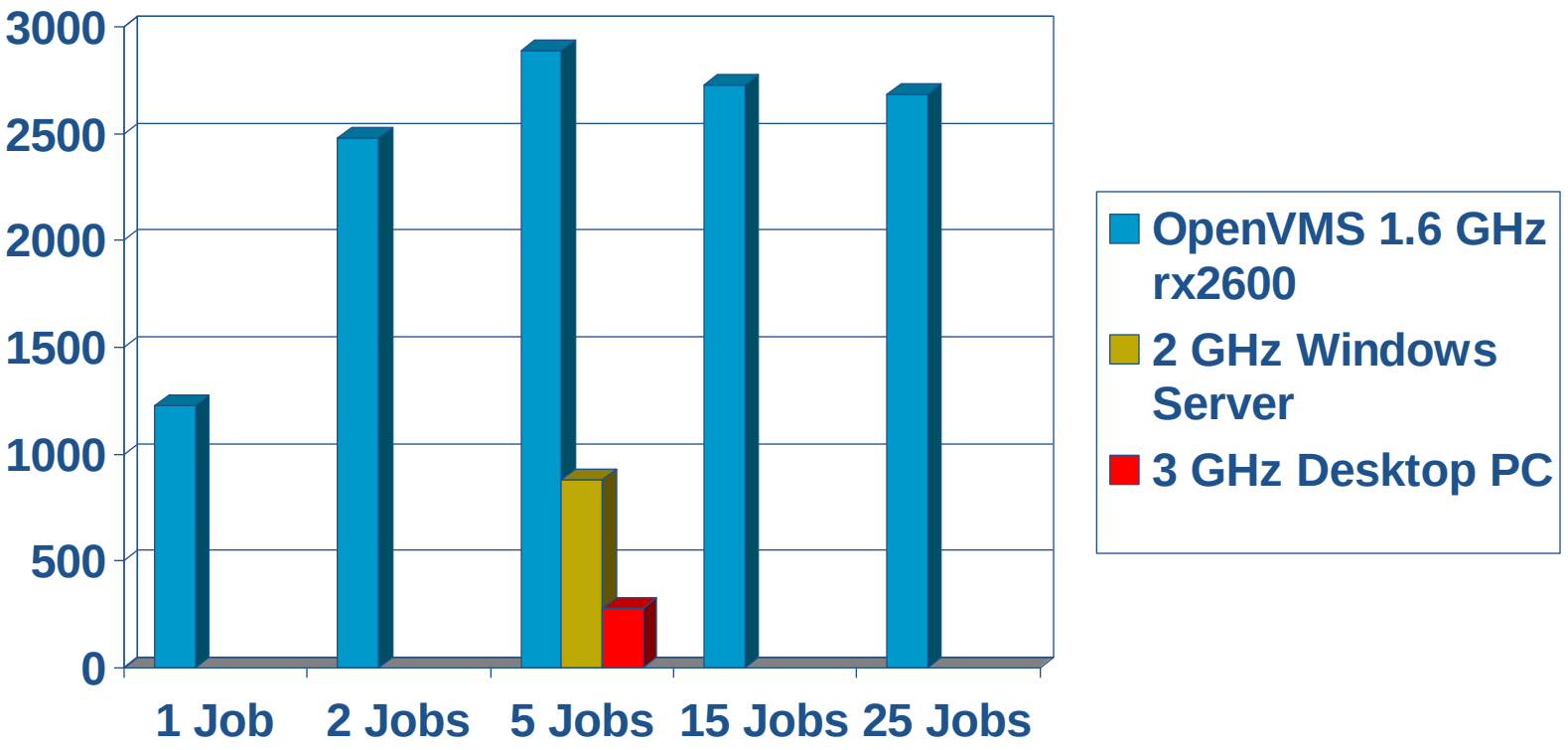
CARFAX VEHICLE HISTORY REPORTS



Seconds to process 60,000 application records (less is better)

Java Benchmark - Stress Test

Number of records processed per second (more is better)



Java on OpenVMS Performance Highlights

- Optimal performance requires:
  - Java 5
    - HotSpot on Itanium
    - FastVM on Alpha
  - OpenVMS V8.3
  - ODS-5
- HotSpot Major Features
  - Fast thread synchronization
  - Adaptive compilation
    - “Hotspots” get dynamically compiled
    - Code changes during lifetime of application
  - Generational garbage collector
- HotSpot on Itanium is **MUCH** better than FastVM on Alpha

Java 5 Performance Highlights

- Creating a JVM takes 3-5 seconds
  - *Not suitable for short & frequent activations*
- CPU bound performance 2-3 times slower than Windows
- Excellent I/O performance
  - NIO package provides same I/O performance as C
- Performance sensitive tasks?
  - Call native code/ O/S services using JNI

CPU bound Benchmarks

- OpenVMS V8.3
- rx2600 1.3 GHz
- CPU intensive / No I/Os performed

Test	C++	Java
Fibonacci	22.72	22.90
Object Instantiation	55.80	1:07.90
String Concatenation	1.44	2.96

Note: Tests do not take full advantage of the HotSpot technology

Tuning Java – 30,000 feet overview

- General system tuning
- Install the main Java images resident. Use RAM disk for main classes, jars and wars.
- Process quotas & system parameters
- Tune JAVA$ logical names
- Tune CRTL logical names
- Heap sizing, garbage collector
- Tune I/O parameters
- Optimize Java code
- Compiled code for performance critical tasks (JNI)

Installation & Setup

- Java for OpenVMS Alpha & I64
  - http://h18012.www1.hp.com/java/alpha/
- Java should be installed on an ODS-5 disk
  - Not necessarily the system disk
  - PRODUCT INSTALL/DESTINATION=ddcn:[foo]
- classpath can be defined using VMS or Unix format
  - VMS Style

    $define java$classpath [], my_jars:[app] ,-

    dka300:[oracle]
  - Unix Style

    $define classpath “/my_jars/app”

Compiling HelloWorld

```
$ ty helloworld.java
class HelloWorld {
        public static void main (String args[]) {
                System.out.println ("Hello World");
        }
}
$
$ javac helloworld
javac: invalid flag: hello
Usage: javac <options> <source files>

$ javac helloworld.java
```

Executing HelloWorld

```
$ java helloworld
Exception in thread "main" java.lang.NoClassDefFoundError: helloworld (wrong name:
    HelloWorld)
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:631)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:124)

$ java HelloWorld
Exception in thread "main" java.lang.NoClassDefFoundError: helloworld (wrong name:
    HelloWorld)
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:631)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:124)

$ java "HelloWorld"
Hello World
$
```

Java is as VERY!! CaSe SenSiTive

Executing HelloWorld

```
$ define decc$argv_parse_style enable
$ define decc$efs_case_preserve enable
$ set proc/parse=exte
$ java HelloWorld
Hello World
```

```
$ java -version
java version "1.5.0"
Java(TM) 2 Runtime Environment, Standard Edition
Java HotSpot(TM) Server VM (build 1.5.0-1 08/05/2006-21:29 IA64, mixed mode)
```

File Format

- Java requires stream_lf file format for .jar, .java, .class and .zip files.
  - Other formats may work but with significant performance reduction (seconds comparing to minutes)
- Convert ASCII file to stream_lf
  - $ convert/fdl=[JAVA$150.COM]STREAM_LF.FDL input output
- Convert binary file to stream lf (.class, .jar, .zip)
  - $ SET FILE/ATTR=(RFM:STMLF,RAT:CR,MRS:0,LRL:32767)
  - jar tf <jarname> to verify format

Resident Images & RAM disks

Resident images

- Candidates to be installed resident
  - [JAVA$150.BIN]JAVA$JAVA.EXE
  - [JAVA$150.JRE.LIB.IA64.HOTSPOT]JAVA$HOTSPOT_SHR.EXE
  - [JAVA$150.JRE.LIB.IA64]JAVA$JAVA_VMS_SHR.EXE
  - [JAVA$150.JRE.LIB.IA64.NATIVE_THREADS]JAVA$HPI_SHR.EXE
  - [JAVA$150.JRE.LIB.IA64]JAVA$VERIFY_SHR.EXE
  - [JAVA$150.JRE.LIB.IA64]JAVA$JAVA_SHR.EXE
  - [JAVA$150.JRE.LIB.IA64]JAVA$ZIP_SHR.EXE
  - [JAVA$150.JRE.LIB.IA64]JAVA$NET_SHR.EXE
  - [JAVA$150.JRE.LIB.IA64]JAVA$NIO_SHR.EXE

Resident images

- Properly size GH region to fit all images
  - GH_RES_CODE
  - GH_RES_DATA
- INSTALL ADD /RESIDENT file_spec
  - /SHARE=ADDRESS where possible
  - VMS83I_INSTALL-V0100 to force resident images back to S0/S1 space

RAM Disks

- DECRAM is integrated with the base O/S
- Included with EOE package or individual PAK
- $ MC MDMANAGER
- CREATE DISK/CAPACITY=NUMBER_OF_BLOCKS
- Initialize & Mount the new device
- Copy key classes / JARs / WARs to the newly created device
- Possibly shadow with local disk
  - The shadow server is smart enough to read from memory

Process Quotas and SYSGEN Parameters

Minimum process quotas

- Fillm – 4096
- WSDEF – 2048 (1MB)
- WSQUO – 4096 (2MB)
- WSEXTENT – 16384 (8MB)
- DIOLM – 500
- BIOLM – 500
- TQELM - 500
- SYSGEN CHANNELCNT – 4096
- SYSGEN WSMAX - 16384

Process Quotas

- BYTLM – 500000
- QUANTUM
- Monitoring Quotas consumption on V8.3
  - ‘Q’ option in SHOW PROC/CONTINUOUS

- [JAVA$150.COM]JAVA$CHECK_ENVIRONMENT

More on Process Quotas later....

Logical Names

Setting up Logical Names

- The run-time behavior of Java is controlled by a set of logical names
- @[JAVA$150.COM]JAVA$CONFIG_WIZARD
  - Generates JAVA$CONFIG_SETUP
  - MUST be executed after JAVA$150_SETUP to avoid overriding selection

Logical Names of Interest

- JAVA$FILENAME_CONTROLS
  - Controls UNIX-Style mapping algorithms
  - To enable all options – set logical to -1
  - To disable all options – set logical to 0
    - Best option for performance
  - Filenames with multiple dots require setting DECC$EFS_CHARSET
- JAVA$SHOW_FILENAME_MAPPING
  - See file mapping as they occur
- Avoid setting JAVA$FILE_OPEN_MODE
  - Results in unnecessary synchronous I/Os
  - If required set DECC$FILE_SHARING to ENABLED

Logical names of interest

- JAVA$DISABLE_MULTIDOT_DIRECTORY_STAT
  - Avoid secondary stat() call when class can't be found in a certain path
  - Only valid if you really do not have multi-dot directories
- JAVA$CACHING_INTERVAL <interval in seconds>
  - stat() information is cached
  - Some information about the files may be answered from cache
  - Cache is invalidated at the end of interval or when a write to the file occurs

Logical names of interest

- JAVA$DAEMONIZE_MAIN_THREAD
  - Avoid starving the main Java thread for CPU time when heavy use of ASTs is made.
  - New thread is created to run the main Java thread.

- JAVA$READDIR_CASE_DISABLE
  - Turns off the case sensitivity filename extraction feature
  - Feature not required on ODS-5 if you ensure filenames are created/named with the correct case
  - Improves performance of scanning directories for .class files

Logical Names of interest

- JAVA$OMIT_CASE_CHECK
  - Increases performance of JAVAC and JAR commands
- Displaying Graphics Intensive applications
  - -Dsun.java2d.pmoffscreen=false
  - Do not store images in pixmaps by default
- JAVA$TIMED_READ_USE_QIO
  - Use QIO & ASTs instead of select()
  - Better performance
  - Look at tcpip show comm/mem search for the number of SELECT structure

Logical Names of interest

- Put files with largest number of classes first in JAVA$CLASSPATH

- JAVA$FORK_PIPE_STYLE
  - 1 Default mailbox driver
  - 2 Uses sockets for buffering data (requires TCP/IP) between heap and parent process

- DECC$ENABLE_GETENV_CACHE
  - CRTL will cache entries returned by getenv()

Logical Names of interest

- DECC$LOCALE_CACHE_SIZE
  - Memory in bytes for caching locale date
  - Default is 0
- DECC$TZ_CACHE_SIZE
  - Number of time zones that can be held in memory
  - Default is 2
  - VMS83I_ACRTL-V0100

Java Memory Management

(Heap & GC)

Java Memory Management

- Java creates a large heap (pre-allocated memory pool) for all memory management related activities
- Similar to a call to LIB$GET_VM
- Objects allocated from the heap
- There is no way to release unused/unneeded memory back to the heap
- C (malloc & free), C++ (new & delete)
- Java (alloc & Garbage Collection)

Java Memory Management

- Garbage Collection (GC) is the process of reclaiming heap space
- Memory reclaimed from dead objects (objects no longer needed)
- Typical GC
  - Stall the application
  - Scan the heap for dead objects
  - Remove dead objects
  - Resume the application
- Process repeated as needed
- Poorly tuned applications may spend significant amount of time performing GC
  - Significant impact on throughput & response time

Heap & GC

- The heap is arranged in three generations
  - Young
  - Tenured
  - Perm
- Three types of Garbage Collectors
  - Serial Collector
  - Throughput Collector
  - Concurrent low pause Collector

Java Memory Management

- Inspire to fit the entire heap into physical memory
  - Increase WS*
    - WSINC
  - Large heap and small working set will cause excessive paging activity
  - Do not oversize the working set
    - Java will use what is given to it regardless of actual requirements
    - Start with 200,000 to 400,000 pagelets (depending on number of users and available memory)
  - Size your pagefiles

- PGFLQUOTA – 2097152
  - Matches unix default supports upto 512MB heap
- Set PGFLQUOTA to twice the heap size

  *pgflquota = (2 * HEAP_IN_MB * 1024 * 1024) / 512*

Heap & GC

- Check for GC - run the application with –verbose:gc
  - -XX:+PrintGCDetails
- Frequent GC's are sign for small heap
- Preferably avoid GC completely
  - increase the heap and process quotas if needed

- Total heap size
  - Bounded below by –Xms and above –Xmx
  - For best performance –Xms == -Xmx
  - (today) 1500MB is hard coded limit for OpenVMS

GC example

```
$  java -Xms64m -Xmx64m -XX:+PrintGCDetails objinst 100000000
false
true
false
true
false
[GC [PSYoungGen: 16448K->120K(19136K)] 16448K->120K(62848K), 0.0030000 secs]
[GC [PSYoungGen: 16568K->128K(19136K)] 16568K->128K(62848K), 0.0010000 secs]
[GC [PSYoungGen: 16576K->120K(19136K)] 16576K->120K(62848K), 0.0010000 secs]
[GC [PSYoungGen: 16568K->120K(19136K)] 16568K->120K(62848K), 0.0010000 secs]
[GC [PSYoungGen: 16568K->136K(19136K)] 16568K->136K(62848K), 0.0010000 secs]
[GC [PSYoungGen: 16584K->120K(19136K)] 16584K->120K(62848K), 0.0010000 secs]
[GC [PSYoungGen: 16568K->120K(19136K)] 16568K->120K(62848K), 0.0010000 secs]
[GC [PSYoungGen: 16568K->120K(16384K)] 16568K->120K(60096K), 0.0010000 secs]
```

GC

- Parallel (throughput) collector
  - -XX:+UseParallelGC
  - Intended for applications with large number of processors
  - Uses multiple threads to execute minor collection
  - -XX:MaxGCPauseMillis=<nnn >
- Concurrent collector
  - -XX:+UseConcMarkSweepGC
  - For applications that would benefit from shorter GC and can afford to share processor resources with the GC when the application is running
  - Optimal results for applications with tenured generations of a modest size
- Serial collector
  - -XX:+UseSerialGC
  - Smaller applications and systems

Heap & GC

- Rules of thumb for sizing the Young Generation:
  - Decide the total amount of memory you can afford to give the JVM
  - Unless you find programs with excessive major collection or pause times, grant plenty of memory to the young generation.
  - Increasing the young generation becomes counterproductive at half the total heap size
  - Be sure the increase the young generation as you increase the number of processors, since allocation can be parallelized.

- Young generation should be sized 3/8th of heap size
  - -XNewSize and –XMaxNewSize bound the young generation size (or –XNewRatio)

GC tips

- Reduce object life time by code inspection
  - Remove references when not required
  - Can do this explicitly with

    objectref = null;

- -XX:+UseAgressiveHeap
  - Requires min of 256MB RAM
  - Overall heap will be around 3850MB
  - GC deferred as long as possible
  - Not suited to multi-app servers
  - Not supported by OpenVMS

Tuning I/O

Tuning I/O

- I/O's are native in Java and eventually reach RMS
- I/O performance comparable to C's capabilities

Optimizing your Java code

Java Vs. Compiler

- Why would I optimize my code?
  - Java is an interpreter
  - Java is not a compiler
  - Java will not shield you from yourself
- If you'll write stupid code…you will execute stupid code

Compiler Optimization

```
1       1 void main (){
        1       2
        1       3           for (int i=0; i<1000000; i++)
        2       4           {
        2       5                   int x = 5;
        2       6                   int y = x * x +300;
        1       7           }
        1       8 }
```

No machine code was generated for lines 2-7…this will not be the case with Java

```
__MAIN:                                                                 // 000001
                        { .mmi
0120000A0380    0000            mov     r14 = 80 ;;
010028E183C0    0001            sub     r15 = sp, r14                   // r15 = r12, r14
000008000000    0002            nop.i   0 ;;
                        }
. . .

 { .mfb
012000002200    0100            mov     r8 = 1                                          // 000008
000008000000    0101            nop.f   0
000108001100    0102            br.ret.sptk.many rp ;;                  // br0
                        }
                                .endp   MAIN
```

Optimize your code

- MONITOR SYSTEM
- MONITOR PAGE
- MONITOR MODES
  - 100% user mode for compute bound applications
  - CPU speed is critical
  - Memory latency
  - Montecito HyperThreads
- JAVAC –”O”
  - Inline certain method calls
  - Only method declared private, static or final can be considered for inlining.

Optimize your code

- Use the shift operator to multiply and divide integers by the power of 2 (X>=0)
  - X >> 2 instead of x/4
  - X << 1 instead of X*2

- Common sub expression elimination
  - Replace

```
double x = d * (lim/max) * sz;
double y = d * (lim/max) * sy;
```

With

```
double depth = d * (lim/max);
double x = depth * sx;
double y = depth * sy;
```

Optimize your code

- ArrayList.contains() requires a linear search
- TreeSet.contains() performs hashed lookup
  - Much! Faster
  - 20 seconds comparing to 3 minutes

- The fastest way to invert boolean is to XOR it
  - Bool ^= true

Optimize your code

- Comparing to zero is faster
  - Doesn't require N to be loaded
  - for (i=0; i<N; i++) should be replaced with

    for (i=N; --I >=0)

- Buffer, vector, map, table, heap, set….all support an initial capacity parameter in their constructors

Strings Vs. StringBuffer

```
public class test1
{
        public static void main (String[] args)
        {
                String s = "[";

                for (int i=1; i<=2000; i++)
                {
                        if (i>1)
                                s += ",";
                        s += Integer.toString (i,10);
                }
                s += "]";

                        System.out.println(s);
        }
```

String Vs. StringBuffer

```
public class test2
{
        public static void main (String[] args)
        {
                StringBuffer sb = new StringBuffer ();

                 sb.append ("[");

                for (int i=1; i<=2000; i++)
                {
                        if (i>1)
                                sb.append  (",");
                        sb.append ( Integer.toString (i,10));
                }
                sb.append("]");
                String s = sb.toString();;

                        System.out.println(s);
```

Profiling

- Engineering resources are limited
  - Where should I invest my tuning time?
- Profile, profile and profile some more…
  - java –prof myClass
    - Look at java.prof
  - java -Xprof

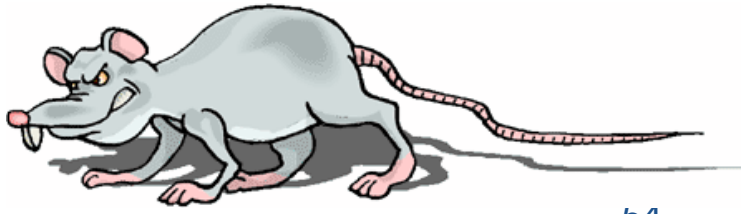
Partial output from Java.prof

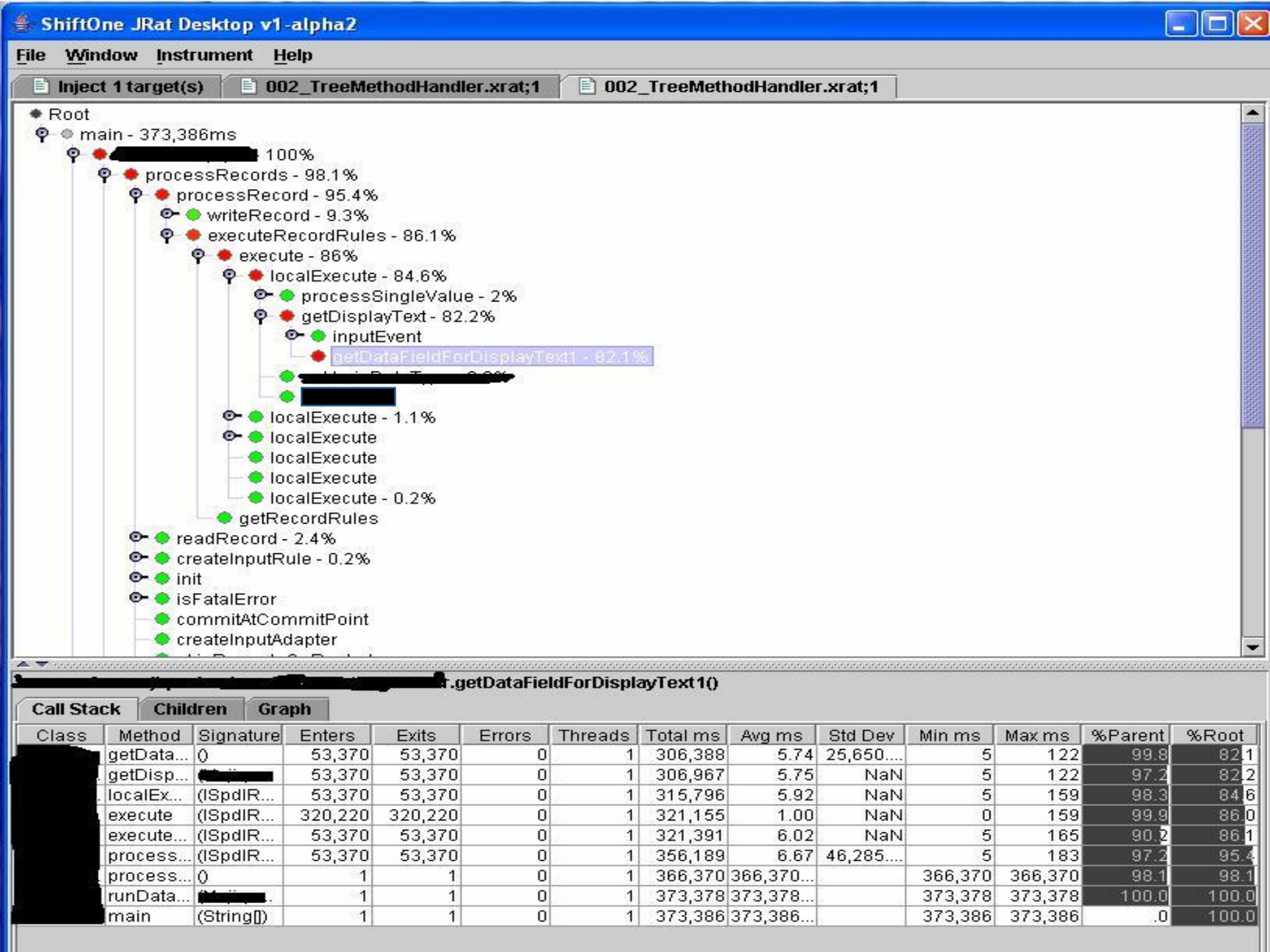
```
count callee caller time
4041 java.lang.String.length()I
     java.lang.AbstractStringBuilder.append(Ljava/lang/String;)Ljava/lang/AbstractStringBuilder; 41
4039 java.lang.String.getChars(II[CI)V
     java.lang.AbstractStringBuilder.append(Ljava/lang/String;)Ljava/lang/AbstractStringBuilder; `
4018 java.lang.AbstractStringBuilder.append(Ljava/lang/String;)Ljava/lang/AbstractStringBuilder;
     java.lang.StringBuffer.append(Ljav`
4001 java.lang.StringBuffer.append(Ljava/lang/String;)Ljava/lang/StringBuffer; test2.main([Ljava/lang/String;)V 306
2000 java.lang.Integer.toString(II)Ljava/lang/String; test2.main([Ljava/lang/String;)V 200
2000 java.lang.Integer.toString(I)Ljava/lang/String; java.lang.Integer.toString(II)Ljava/lang/String; 148
2000 java.lang.String.<init>(II[C)V java.lang.Integer.toString(I)Ljava/lang/String; 21
2000 java.lang.Integer.getChars(II[C)V java.lang.Integer.toString(I)Ljava/lang/String; 24
2000 java.lang.Integer.stringSize(I)I java.lang.Integer.toString(I)Ljava/lang/String; 26
242 java.lang.AbstractStringBuilder.append(C)Ljava/lang/AbstractStringBuilder;
     java.lang.StringBuilder.append(C)Ljava/lang/StringBu`
237 java.lang.String.length()I sun.net.www.ParseUtil.decode(Ljava/lang/String;)Ljava/lang/String; 5
232 java.lang.String.charAt(I)C java.io.UnixFileSystem.normalize(Ljava/lang/String;)Ljava/lang/String; 2
231 java.lang.String.charAt(I)C sun.net.www.ParseUtil.decode(Ljava/lang/String;)Ljava/lang/String; 5
231 java.lang.StringBuilder.append(C)Ljava/lang/StringBuilder;
     sun.net.www.ParseUtil.decode(Ljava/lang/String;)Ljava/lang/String; 4
33 java.lang.String.indexOf(II)I java.lang.String.indexOf(I)I 0
32 java.lang.String.startsWith(Ljava/lang/String;I)Z java.lang.String.endsWith(Ljava/lang/String;)Z 1
28 java.lang.AbstractStringBuilder.expandCapacity(I)V
     java.lang.AbstractStringBuilder.append(Ljava/lang/String;)Ljava/lang/Abstract`
25 sun.misc.VM.allowArraySyntax()Z java.lang.ClassLoader.checkName(Ljava/lang/String;)Z 0
25 java.lang.String.indexOf(I)I java.lang.ClassLoader.checkName(Ljava/lang/String;)Z 1
25 java.lang.String.length()I java.lang.ClassLoader.checkName(Ljava/lang/String;)Z 1
```

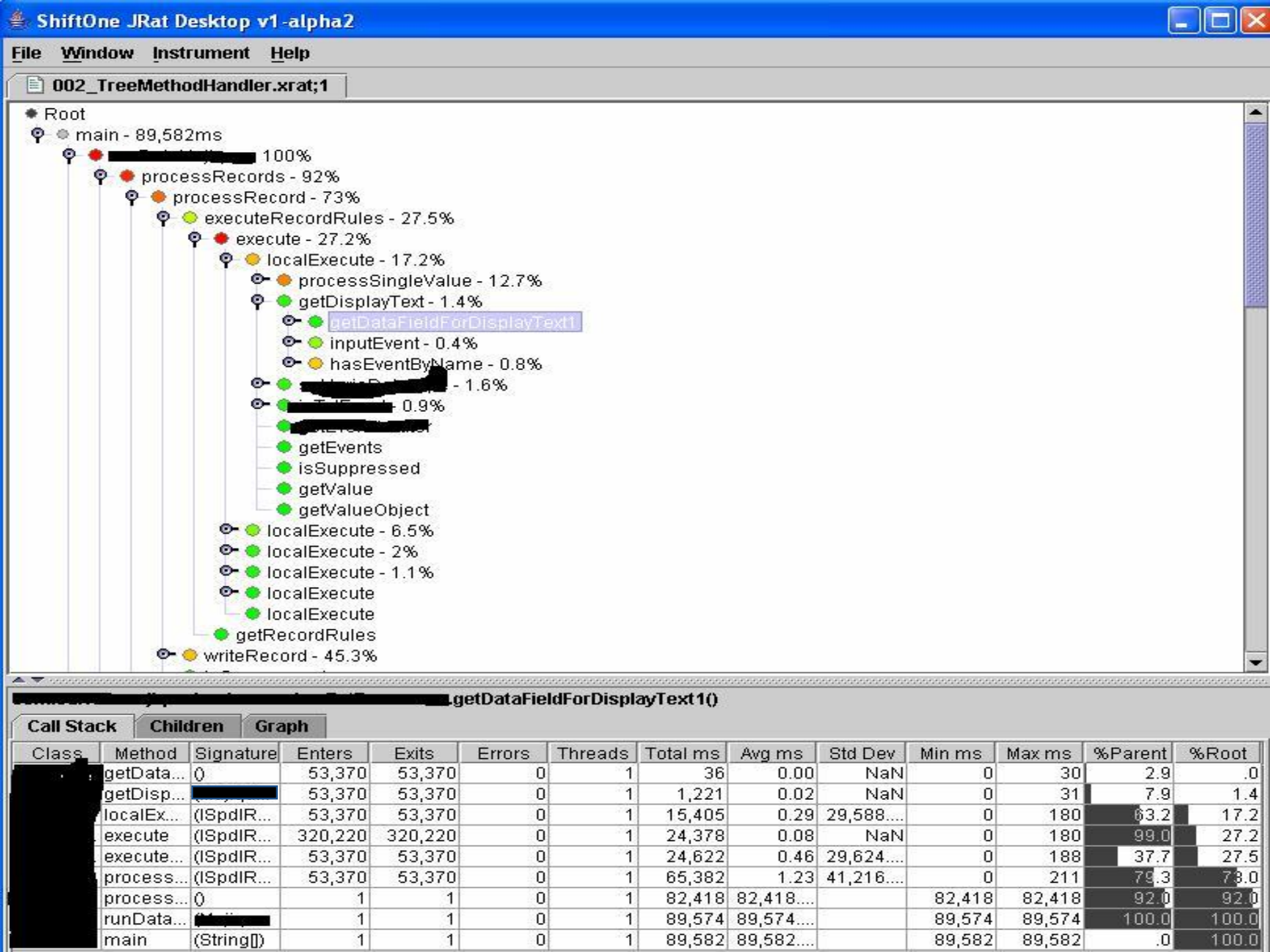
JRat

- *JRat is the Java™ Runtime Analysis Toolkit. Its purpose is to enable developers to better understand the runtime behavior of their Java™ programs. The term "behavior" includes, but is not limited to performance profiling.*
- JRat can:
  - accumulate timing statistics (a few ways)
  - create trace logging
  - track rate methods are called over time
  - track the response time of methods over time

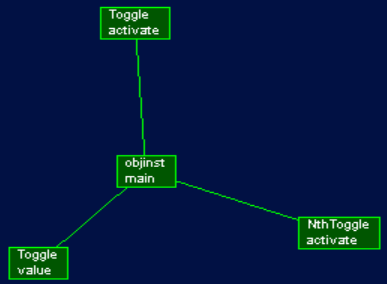
http://jrat.sourceforge.net/




ShiftOne JRat Desktop v1-alpha2
File
Window
Instrument
Help
Inject 1 target(s)
002_TreeMethodHandler.xrat;1
002_TreeMethodHandler.xrat;1
Root
main - 373,386ms
[redacted] 100%
processRecords - 98.1%
processRecord - 95.4%
writeRecord - 9.3%
executeRecordRules - 86.1%
execute - 86%
localExecute - 84.6%
processSingleValue - 2%
getDisplayText - 82.2%
inputEvent
getDataFieldForDisplayText1 - 82.1%
[redacted]
[redacted]
localExecute - 1.1%
localExecute
localExecute
localExecute
localExecute - 0.2%
getRecordRules
readRecord - 2.4%
createInputRule - 0.2%
init
isFatalError
commitAtCommitPoint
createInputAdapter
[redacted].getDataFieldForDisplayText1()
Call Stack
Children
Graph
Class
Method
Signature
Enters
Exits
Errors
Threads
Total ms
Avg ms
Std Dev
Min ms
Max ms
%Parent
%Root
[redacted] getData... () 53,370 53,370 0 1 306,388 5.74 25,650.... 5 122 99.8 82.1
getDisp... [redacted] 53,370 53,370 0 1 306,967 5.75 NaN 5 122 97.2 82.2
localEx... (ISpdlR... 53,370 53,370 0 1 315,796 5.92 NaN 5 159 98.3 84.6
execute (ISpdlR... 320,220 320,220 0 1 321,155 1.00 NaN 0 159 99.9 86.0
execute... (ISpdlR... 53,370 53,370 0 1 321,391 6.02 NaN 5 165 90.2 86.1
process... (ISpdlR... 53,370 53,370 0 1 356,189 6.67 46,285.... 5 183 97.2 95.4
process... () 1 1 0 1 366,370 366,370... 366,370 366,370 98.1 98.1
runData... [redacted] 1 1 0 1 373,378 373,378... 373,378 373,378 100.0 100.0
main (String[]) 1 1 0 1 373,386 373,386... 373,386 373,386 .0 100.0


ShiftOne JRat Desktop v1-alpha2
File Window Instrument Help
002_TreeMethodHandler.xrat;1
Root
main - 89,582ms
[redacted] 100%
processRecords - 92%
processRecord - 73%
executeRecordRules - 27.5%
execute - 27.2%
localExecute - 17.2%
processSingleValue - 12.7%
getDisplayText - 1.4%
getDataFieldForDisplayText1
inputEvent - 0.4%
hasEventByName - 0.8%
[redacted] - 1.6%
[redacted] 0.9%
[redacted]
getEvents
isSuppressed
getValue
getValueObject
localExecute - 6.5%
localExecute - 2%
localExecute - 1.1%
localExecute
localExecute
getRecordRules
writeRecord - 45.3%
[redacted].getDataFieldForDisplayText1()
Call Stack
Children
Graph
Class
Method
Signature
Enters
Exits
Errors
Threads
Total ms
Avg ms
Std Dev
Min ms
Max ms
%Parent
%Root
[redacted] getData... () 53,370 53,370 0 1 36 0.00 NaN 0 30 2.9 .0
[redacted] getDisp... [redacted] 53,370 53,370 0 1 1,221 0.02 NaN 0 31 7.9 1.4
[redacted] localEx... (ISpdIR... 53,370 53,370 0 1 15,405 0.29 29,588.... 0 180 63.2 17.2
[redacted] execute (ISpdIR... 320,220 320,220 0 1 24,378 0.08 NaN 0 180 99.0 27.2
[redacted] execute... (ISpdIR... 53,370 53,370 0 1 24,622 0.46 29,624.... 0 188 37.7 27.5
[redacted] process... (ISpdIR... 53,370 53,370 0 1 65,382 1.23 41,216.... 0 211 79.3 73.0
[redacted] process... () 1 1 0 1 82,418 82,418.... 82,418 82,418 92.0 92.0
[redacted] runData... [redacted] 1 1 0 1 89,574 89,574.... 89,574 89,574 100.0 100.0
[redacted] main (String[]) 1 1 0 1 89,582 89,582.... 89,582 89,582 .0 100.0

objinst.main(String[])



Zoom Rotate Hyperbolic Right-click nodes and background for more options

Using JNI Overview

The Need for JNI on OpenVMS

- Java directly meets most programming needs with the exception of:
  - Run-time Library Calls
  - System Service Calls
  - Specialized RMS calls such as indexed file access
  - Specialized high performance routine
- To support these interfaces you can use the Java Native Interface (JNI) mechanism.
  - You can use JNI to call C routines for OpenVMS support
- You may want to call Java from C, for info on this see

  http://h18012.www1.hp.com/java/documentation/1.5.0/ovms/docs/user_guide.html

Questions?

BRUDEN-OSSG thanks you for attending this session.

See us at www.BRUDENOSSG.com for:

- *Performance analysis*
  - *(Performance Results Or No Expense)*
- *Porting assistance*
- *Special OPS (OpenVMS Programming Services)*
- *System Management Support*
- *O/S Support and Crash Dump Analysis*