

## **Analyzing Java Performance Using Hardware Performance Counters**

Gary Frost  
AMD Java Labs  
gary.frost@amd.com

## Agenda

AMD Java Labs

Hardware Performance Counters

A Workload To Profile

Profiling using hprof

Profiling with CodeAnalyst™

Profiling with CodeSleuth

Demo

Demo closure

Q&A

## AMD Java Labs

- An engineering team within AMD dedicated to improving Java performance on AMD processors
- Provide feedback/assistance in optimization work to JVM vendors
  - Create/Evaluate Java performance related tools
  - Provide benchmark analysis (SPEC member)
  - Routing community feedback to silicon designers to improve future products
  - Participate in Java and open source communities

## What Are Hardware Performance Counters?

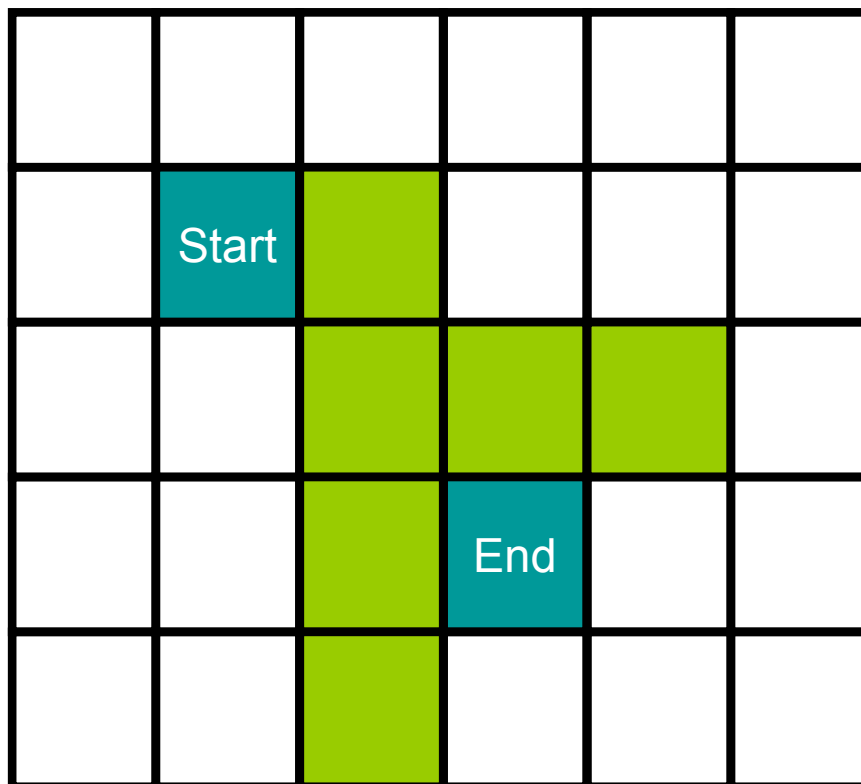
- Programmable counters inside the processor
- Accessed via device drivers (not from user mode)
- Can generate interrupts when trigger value is reached
- System-wide

## How Performance Counters Can Help Java Developers

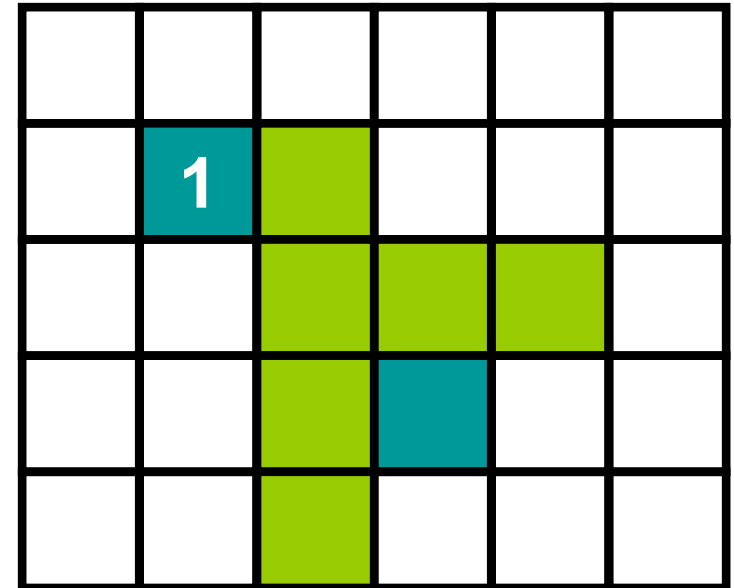
- Understand what the processor is really doing with your code
- Help isolate cause of bottlenecks
- Provide detail that JVM based profilers can not
  - ◆ hprof takes you to the method, but not **into** the method
- No need to inject instrumentation
- Help developers make informed code and algorithm choices
- Measure and analyze as you develop code

## A Workload To Profile

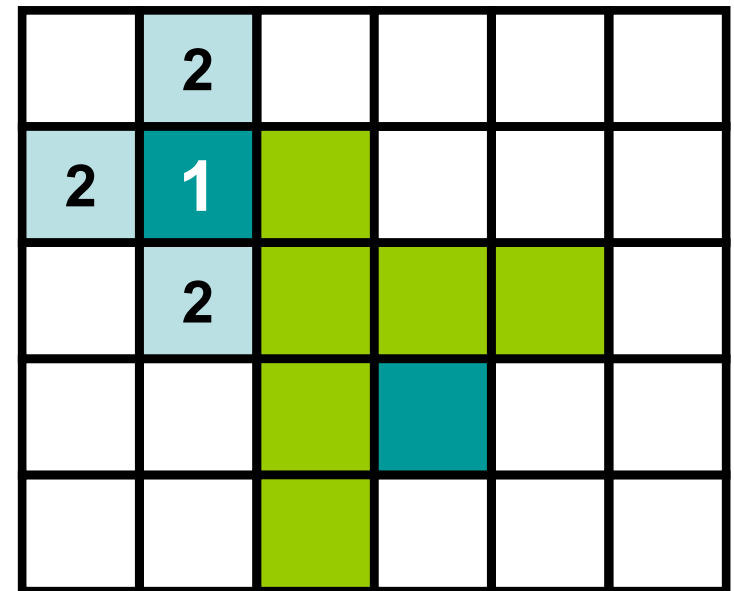
Find path between  
**start** and **end**  
avoiding **obstacles**  
using 'Lee routing  
algorithm'



Mark starting cell with 1



for each cell with number 'n'  
mark adjacent empty cells 'n+1'  
until end marked | no empty cells



Eventually...

Step onto end cell  
 while (n!=1)  
     mark as part of path  
     step on adjacent cell (n-1)

3	2	3	4	5	6
2	1		5	6	7
3	2				8
4	3		11	10	9
5	4				10

3	2	3	4	5	6
2	1		5	6	7
3	2				8
4	3		11	10	9
5	4				10

Voila !

	2	3	4	5	6
	1				7
					8
			11	10	9

## Profiling with hprof

Shipped with the JDK easy to use command line tool

```
java -agentlib:hprof=cpu=samples -cp lee.jar lee.Main
```

Creates a report

```
rank self accum method
  1 93%  93%   lee.Grid.flood
  2  2%  95%   java.lang.AbstractStringBuilder.append
  3  2%  97%   java.util.Collections.unmodifiableList
  4  2% 100%   sun.nio.cs.UTF_8.newDecoder
```

## hprof

- Reports how much time was spent on various call paths (stacks traces) to a method

### But

- Does not tell us where within the method the time was spent
- Does not expose what happens in code emitted by the JIT (compiled methods)

hprof is representative of many Java profilers in that it is limited by the information that can be extracted from the JVM.

## How AMD's CodeAnalyst™ Can Help

CodeAnalyst™ is a profiling tool which can access hardware performance counters

- Timer- or Event-based Profiling
- Linux 32- and 64-bit and Microsoft® Windows® 32- and 64-bit OSs
- Free! (<http://developer.amd.com/tools>)
- Maps performance data to address and source (C++/C/Java)
- Includes code emitted by the JIT

## CodeAnalyst From The Command Line

- Coordinate the launching of an executable

```
caprofile /s /l java -cp my.jar -agentlib:CAJVMTIA32 MyApp
```

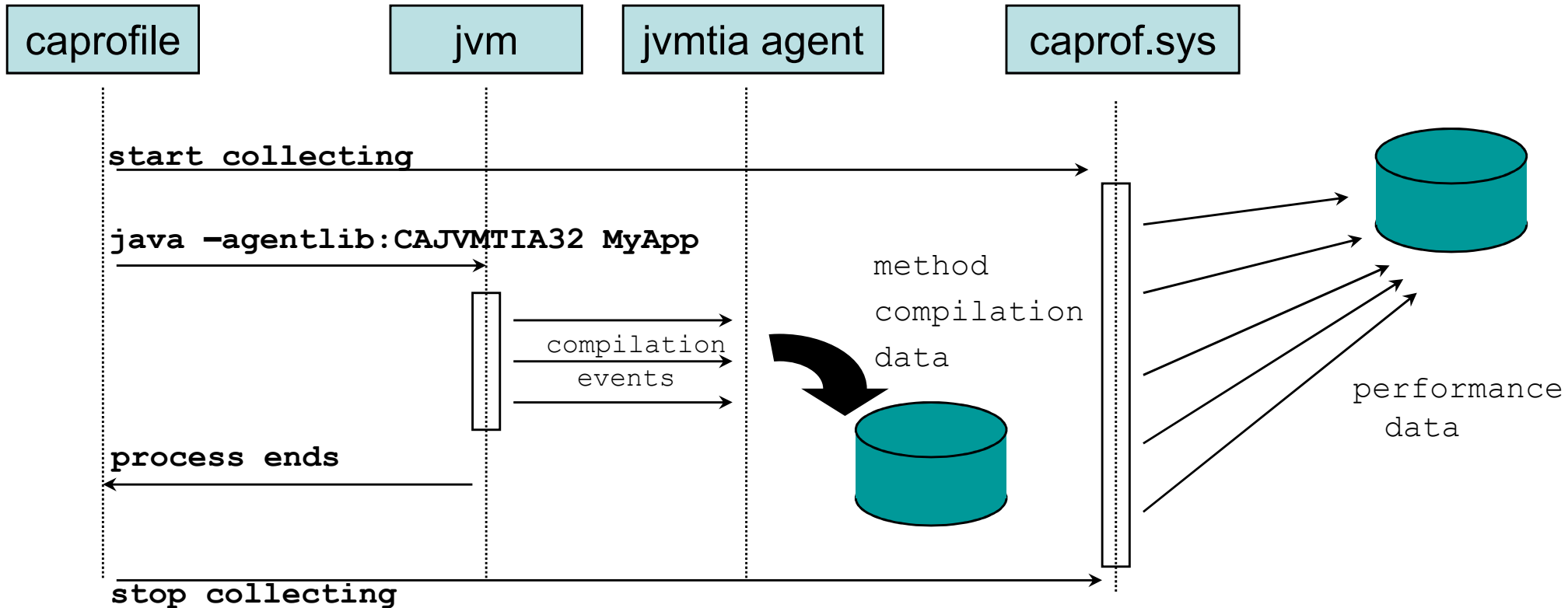
- Profiler (`caprofile`) collects event data
- JVMTI agent (`CAJVMTIA32`) collects JIT data
- Correlates and creates reports

```
cadataanalyze
```

```
careport /i profile.tbp.dir/profile.tbp /m MyApp
```

# CodeAnalyst

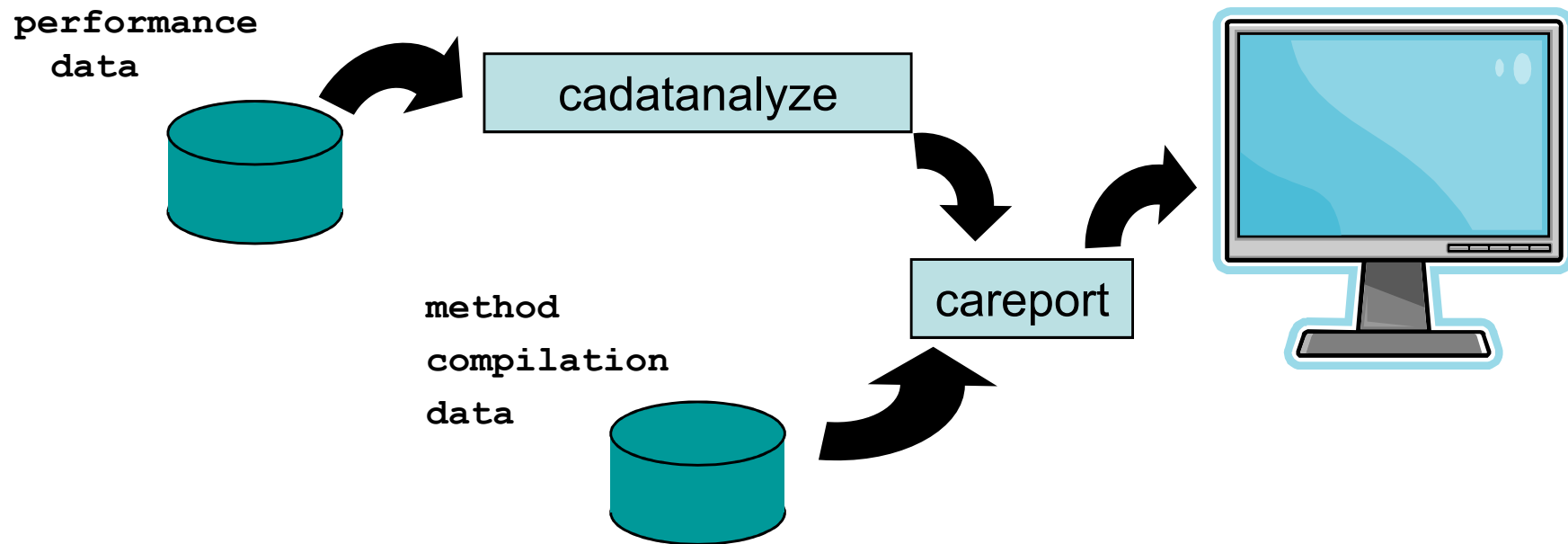
caprofile /s // java -cp my.jar -agentlib:CAJVMTIA32 MyApp



## CodeAnalyst

**cadataanalyze**

**careport /i profile.tbp.dir/profile.tbp /m MyApp**

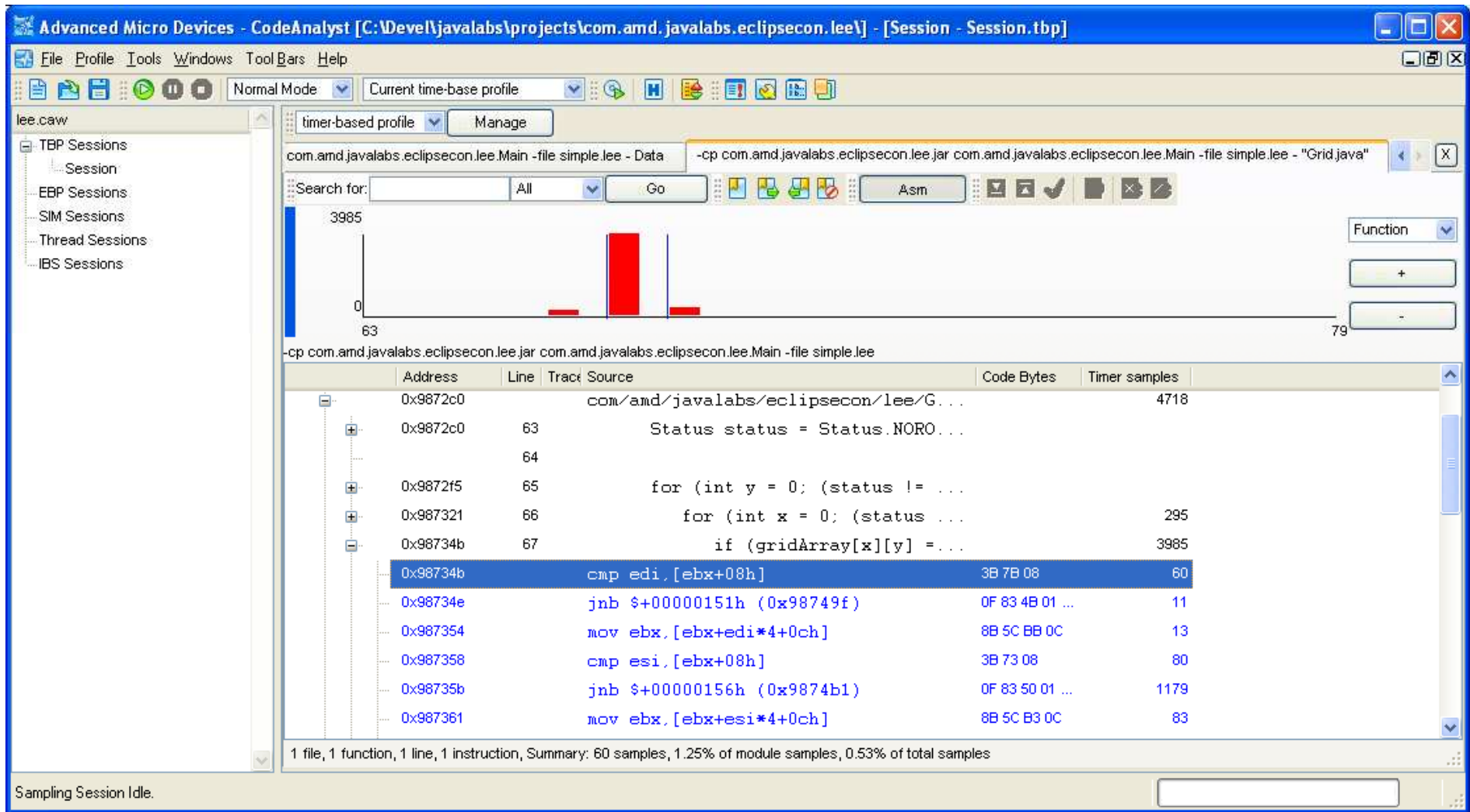


## CodeAnalyst

- Event data, addresses, and methods

vma	Timer	% total	Name
0x9877c0	24269	56.00	lee/Grid::flood
0x98781b	1	0.00	
...			
0x98784d	6132	14.28	
0x987853	474	1.10	
0x987857	13135	30.58	
0x987859	402	0.94	
...			

# CodeAnalyst UI



The screenshot shows the CodeAnalyst interface with a timer-based profile. The main window displays a bar chart and a table of assembly instructions.

**Timer-based profile:**

- Y-axis: 0 to 3985
- X-axis: 63 to 79
- Bar at address 0x98734b (line 67) has a height of 3985.

**Assembly Table:**

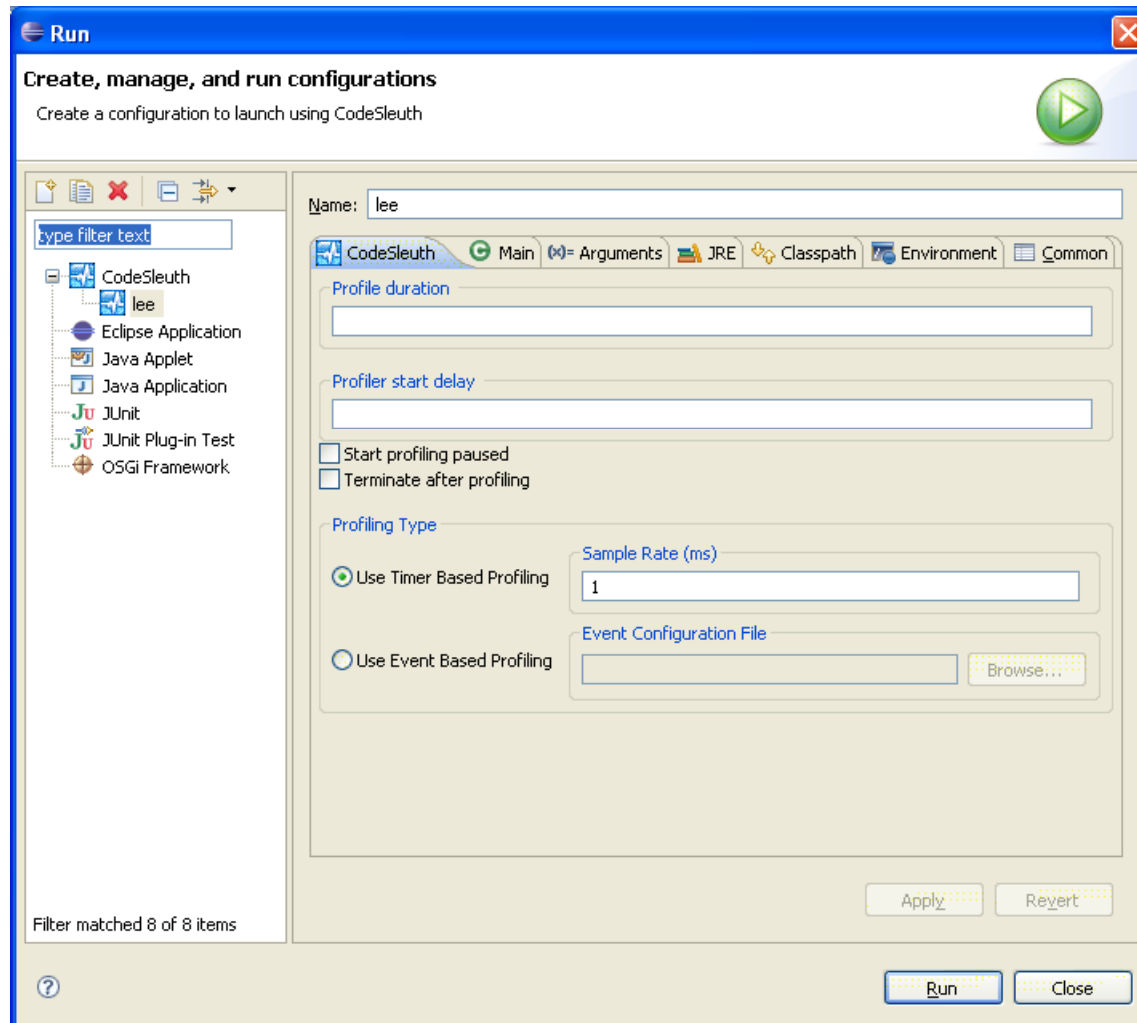
Address	Line	Tract	Source	Code Bytes	Timer samples
0x9872c0			com/amd/javab.../lee/G...		4718
0x9872c0	63		Status status = Status.NORO...		
	64				
0x9872f5	65		for (int y = 0; (status != ...		
0x987321	66		for (int x = 0; (status ...		295
0x98734b	67		if (gridArray[x][y] = ...		3985
0x98734b			cmp edi, [ebx+08h]	3B 7B 08	60
0x98734e			jnb \$+00000151h (0x98749f)	0F 83 4B 01 ...	11
0x987354			mov ebx, [ebx+edi*4+0ch]	8B 5C BB 0C	13
0x987358			cmp esi, [ebx+08h]	3B 73 08	80
0x98735b			jnb \$+00000156h (0x9874b1)	0F 83 50 01 ...	1179
0x987361			mov ebx, [ebx+esi*4+0ch]	8B 5C B3 0C	83

Summary: 1 file, 1 function, 1 line, 1 instruction, Summary: 60 samples, 1.25% of module samples, 0.53% of total samples

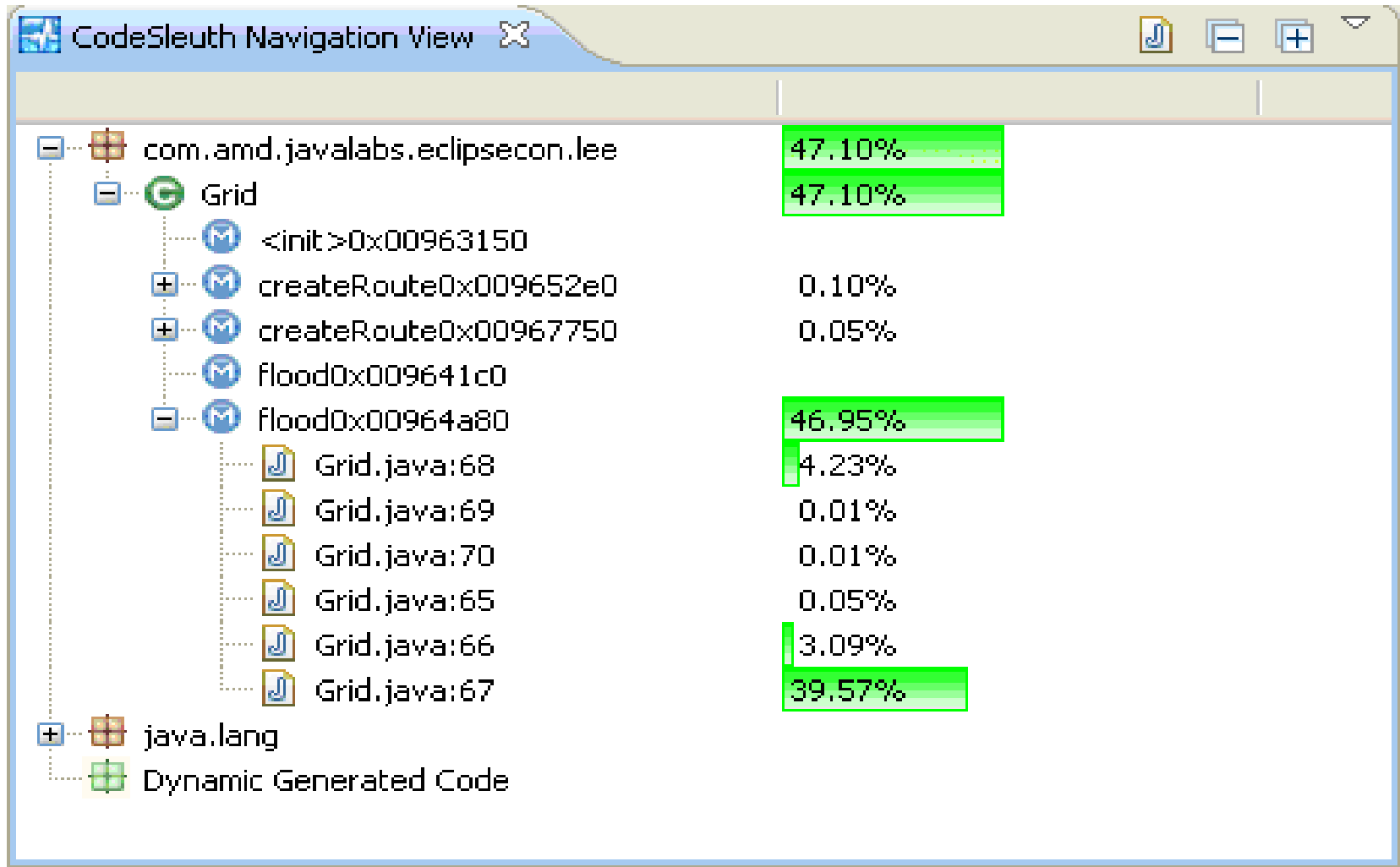
## CodeSleuth

- Open Source (EPL)
- Brings various CodeAnalyst ‘views’ into Eclipse
- More natural for Java developers
  - ◆ Same IDE we are used to
  - ◆ SourceView is your Java editor
  - ◆ ‘Markers’ indicate event data we can expand
  - ◆ Configure CodeAnalyst™ parameters in Launch Configuration
  - ◆ Cross reference architecture documentation from within IDE
- Makes Java analysis easier

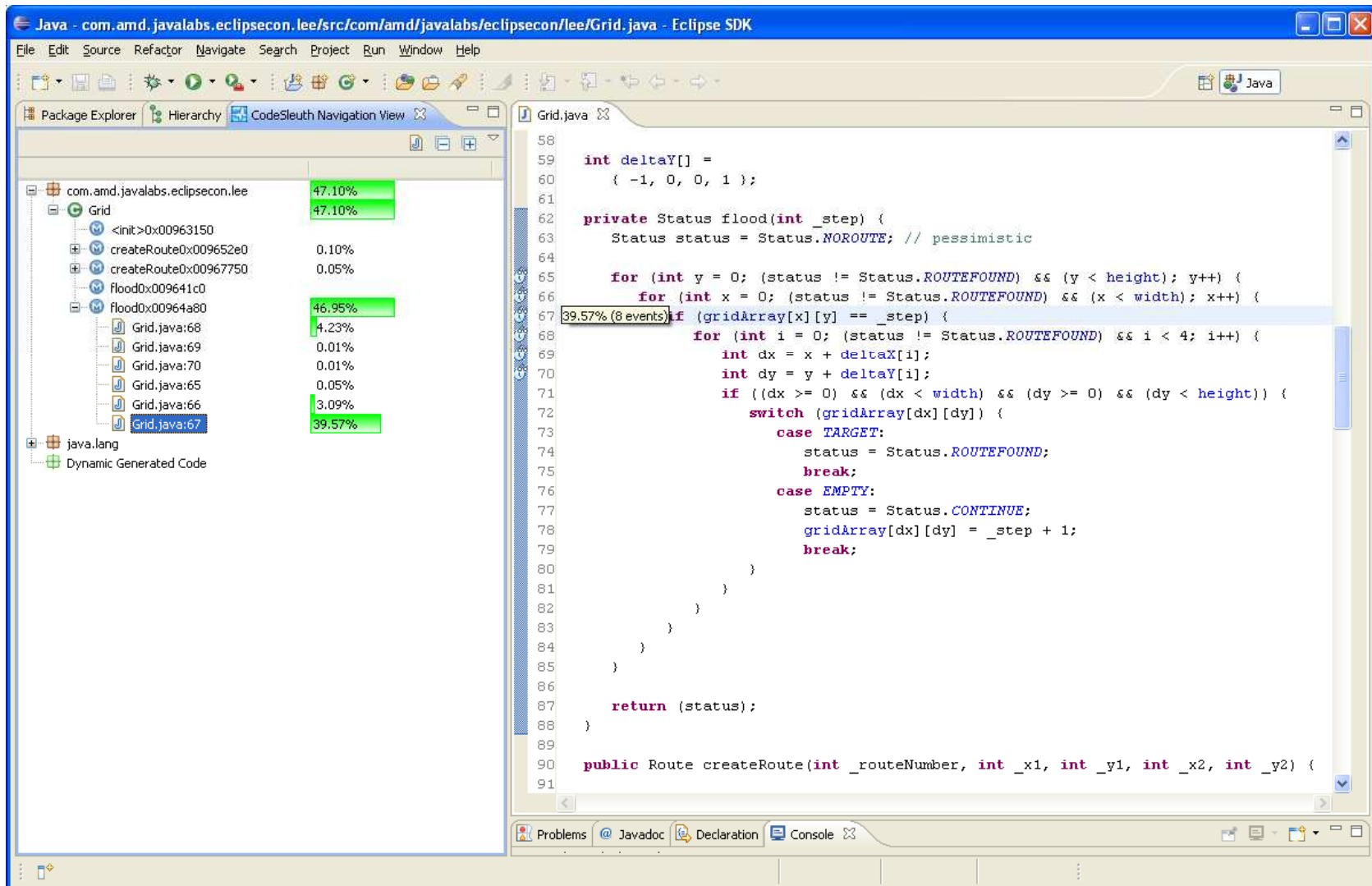
# CodeSleuth Run Configuration



# CodeSleuth Navigation View



# Linking views to Java source



# CodeSleuth Disassembled Code View

Event Data	Address	Hex Bytes	Mnemonic	Operands
	00964ae3	90	nop	
0.98%	00964ae4	bb a8 0f 97 26	mov	%ebx 0x26970fa8
0.02%	00964ae9	8b 9b 58 01 00 00	mov	%ebx [%ebx+0x158]
0.14%	00964aef	3b c3	cmp	%eax %ebx
0.86%	00964af1	0f 84 3e 01 00 00	je	0x00964c35
0.03%	00964af7	8b 59 1c	mov	%ebx [%ecx+0x1c]
0.07%	00964afa	85 05 00 01 3f 00	test	[+0x3f0100 %eax
0.97%	00964b00	3b fb	cmp	%edi %ebx
0.02%	00964b02	0f 8d 2d 01 00 00	jge	0x00964c35
	00964b08	8b 59 08	mov	%ebx [%ecx+0x8]
0.91%	00964b0b	3b 7b 08	cmp	%edi [%ebx+0x8]
0.06%	00964b0e	0f 83 4b 01 00 00	jae	0x00964c5f
0.02%	00964b14	8b 5c bb 0c	mov	%ebx [%ebx+%edi*4+0xc]
0.79%	00964b18	3b 73 08	cmp	%esi [%ebx+0x8]
11.82%	00964b1b	0f 83 50 01 00 00	jae	0x00964c71
0.59%	00964b21	8b 5c b3 0c	mov	%ebx [%ebx+%esi*4+0xc]
24.65%	00964b25	3b da	cmp	%ebx %edx
0.73%	00964b27	0f 85 ea 00 00 00	jne	0x00964c17
	00964b2d	bb 00 00 00 00	mov	%ebx 0x0
	00964b32	90	nop	
	00964b33	90	nop	
	00964b34	ba a8 0f 97 26	mov	%edx 0x26970fa8
	00964b39	8b 92 58 01 00 00	mov	%edx [%edx+0x158]
	00964b3f	3b c2	cmp	%eax %edx
	00964b41	0f 84 d0 00 00 00	je	0x00964c17
	00964b47	85 05 00 01 3f 00	test	[+0x3f0100 %eax

# Documentation Views

**1 Instruction Formats**

The format of an instruction encodes its operation, as well as the locations of the instruction's operands and the result of the operation. This section describes the general format and parameters by all instructions. For information on the specific format(s) for each instruction, see:

- Chapter 3, "General-Purpose Instruction Reference."
- Chapter 4, "System Instruction Reference."
- "128-Bit Media Instruction Reference" in Volume 4.
- "64-Bit Media Instruction Reference" in Volume 5.
- "x87 Floating-Point Instruction Reference" in Volume 5.

**1.1 Instruction Byte Order**

An instruction can be between one and 15 bytes in length. Figure 1-1 shows the byte order of instruction format.

**Figure 1-1. Instruction Byte-Order**

# Demo CodeSleuth

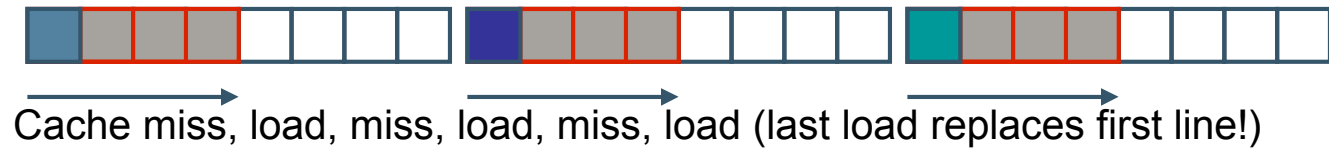
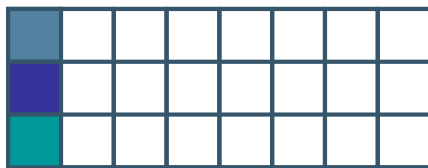
## Demo Closure

- Code uses a large two dimensional array
- The order 'x by y' or 'y by x' for traversing this array is important when arrays are large (cache implications)
- The original scan order does not take advantage of the cache, in fact it 'pollutes' its own cache
- By reordering the scan we can decrease cache pollution and thus improve performance

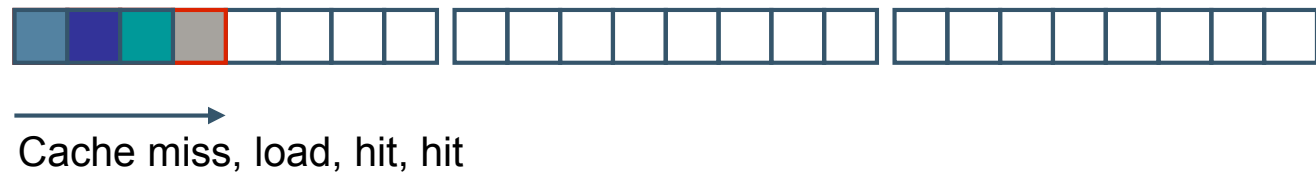
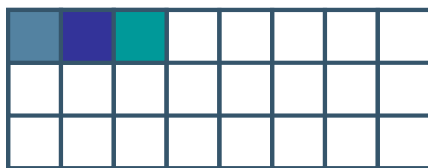
# Demo Closure

- If we had 2 cache lines, each of which can hold 4 cells

Y by X (Column Major)



X by Y (Row Major)



## Conclusions

- Hardware performance counters can help reveal performance issues
- CodeAnalyst enables you to analyze CPU level performance of Java applications
- CodeSleuth helps bring the power of this tool into your favorite IDE
- Promote the inclusion of performance analysis as part of your development/test cycle

## CodeSleuth Links

### Site

<http://developer.amd.com/codesleuth>

### Source

<http://codesleuth.sourceforge.net>

### Update Site

[http://developer.amd.com/codesleuth\\_1\\_0\\_0](http://developer.amd.com/codesleuth_1_0_0)

## Links

- <http://developer.amd.com>
  - ◆ Software downloads
  - ◆ AMD Java Labs blog
  - ◆ Forum discussions
- Lee, C.Y., "An Algorithm for Path Connections and Its Applications", IRE Transactions on Electronic Computers, vol. EC-10, number 2, pp. 364-365, 1961