ARM® DS-5

Version 5.4

Using ARM Streamline

ARM®

ARM DUI 0482B (ID012411)

ARM DS-5
Using ARM Streamline

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
September 2010	A	Non-Confidential	ARM Streamline Performance Analyzer 1.0
January 2011	B	Non-Confidential	Update for DS-5 version 5.4

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
Using ARM Streamline

Chapter 1
Conventions and feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions

The following typographical conventions are used:

`monospace` Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

`mono`space Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

`monospace italic`

Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- your name and company

- the serial number of the product
- details of the release you are using
- details of the platform you are using, such as the hardware platform, operating system type and version
- a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem
- the version string of the tools, including the version number and build numbers.

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DUI 0482B
- if viewing online, the topic names to which your comments apply
- if viewing a PDF version of a document, the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and *Frequently Asked Questions* (FAQs).

Other information

- ARM Information Center, http://infocenter.arm.com/help/index.jsp
- ARM Technical Support Knowledge Articles, http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html
- ARM Support and Maintenance , http://www.arm.com/support/services/support-maintenance.php.

Chapter 2
Getting Started with ARM Streamline

ARM Streamline Performance Analyzer is a system-wide visualizer and profiler for systems running ARM Linux or Android native applications and libraries. Combining an ARM Linux kernel driver, target daemon, and a graphical user interface, it transforms system trace and sampling data into reports that present the data in both visual and statistical forms. Streamline leverages hardware performance counters with kernel metrics to provide an accurate representation of system resources. Streamline supports Cortex™-A8, Cortex-A9 UP, Cortex-A9 SMP, ARM9™, and ARM11™ processors running ARM Linux.

The following topics describe how to set up you target and run Streamline, then gives you a overview of the available views:

2.1 Setting up the Gator driver and daemon

The Gator daemon and driver collect target metrics and then send them to your host machine.

To enable profiling, load the Gator daemon and driver on your target.

2.1.1 Prerequisites

You need the following tools on your host to build the Linux kernel and the Gator driver:

- Linux kernel source code for the target platform. Note that Streamline supports only Linux kernel versions 2.6.32 and above.
- Either the cross compiler for building the Linux kernel or the ARM Linux GCC that comes with DS-5.

2.1.2 Setting up the Gator driver and daemon

To set up the Gator driver and daemon, first prepare and build your kernel by following these steps:

1. Navigate to the root source directory of the Linux kernel.
2. Enter the following command in your shell:

   make ARCH=arm CROSS_COMPILE=${CROSS_TOOLS}/bin/arm-none-linux-gnueabi-*Your_SoC*_defconfig
3. Enter the following command in your shell:

   make ARCH=arm CROSS_COMPILE=${CROSS_TOOLS}/bin/arm-none-linux-gnueabi-menuconfig
4. Activate the Profiling Support option in General Setup: **CONFIG_PROFILING=y**.
5. Under Kernel hacking, activate the **Tracers** and **Trace process context switches and events** options: **CONFIG_FTRACE=y** and **CONFIG_ENABLE_DEFAULT_TRACERS=y**.
6. Enter the following command in your shell to build the kernel:

   make -j5 ARCH=arm CROSS_COMPILE=${CROSS_TOOLS}/bin/arm-none-linux-gnueabi- uImage

2.1.3 Building the Gator module

To use Streamline with your ARM target, you need to build the Gator driver, `gator.ko` and place it in the same directory as the Gator daemon, `gatord`, on the target file system. Assuming that you have all of the required tools for building kernel modules, enter the following command to create the `gator.ko` module:

make -C *kernel_build_dir* M=`pwd` ARCH=arm CROSS_COMPILE=<...> modules

2.1.4 Running the Gator daemon on your target

Now that you have all of the necessary files in place, it is time to start the Gator daemon. To run `gatord`:

1. Load the kernel onto the target

2. Copy `gatord` and `gator.ko` into the file system on the target

   Note

   `gatord` is located in *`installdir`*`/arm/armv5t/` on your host. `gatord` must be placed in the same directory as `gator.ko` on the target.

3. To ensure `gatord` has execute permission, enter the following command:

   chmod +x gatord

4. After making sure that you have root privileges, enter the following to execute the Gator daemon:

   ./gatord &

By default, `gatord` uses port 8080 for communication with the host, but you can adjust this by launching `gatord` with the port number as a parameter and changing the **Port** option in the Capture Options dialog box. To open the Capture Options dialog box, click **Change capture options** in the ARM Streamline Data view.

2.1.5 See also

Tasks

- *Setting capture options* on page 2-4
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Other capture options* on page 2-6
- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-14
- *The Code view* on page 2-16

2.2 Setting capture options

If you have not already set up an Eclipse workspace, either create a new one or use the default location.

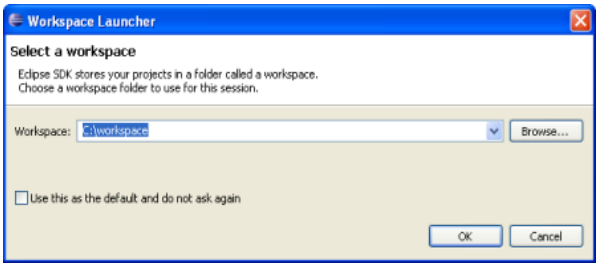

Figure 2-1 The workspace launcher

If this is your first time opening the workspace, Eclipse for DS-5 displays the welcome screen. To get started, click on the **Go to Workbench** link. If you have used this workspace before, the workbench opens directly.

To edit capture settings and connect to the target:

1. Make sure that you have the ARM Streamline Data view open.

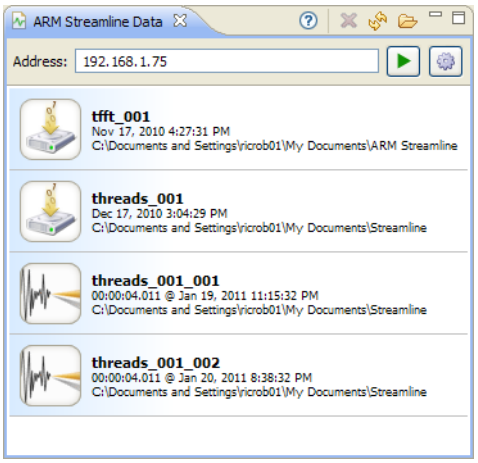

Figure 2-2 The ARM Streamline data view

   To open the ARM Streamline Data view, select **Window** → **Show View** → **Other...**, and choose it from the ARM Streamline folder.

2. Click **Change capture options**, located in the upper right of the ARM Streamline Data view.

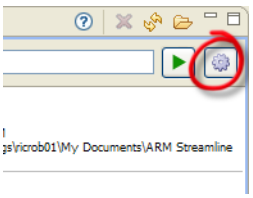

Figure 2-3 The Change capture options button

3. Enter a name for the captured settings in the **Name** field:

Figure 2-4 Naming the configuration

4. Enter the name or IP address of your target in the **Address** field.
5. Click **Add Program...** to add any program images.
6. Click **Apply** to save the settings.
7. Close the Capture Options dialog box.

2.2.1 See also

Tasks

- *Setting up the Gator driver and daemon* on page 2-2
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Other capture options* on page 2-6
- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-14
- *The Code view* on page 2-16

2.3 Other capture options

The Capture Options dialog box presents various options you can use to customize a profiling session.

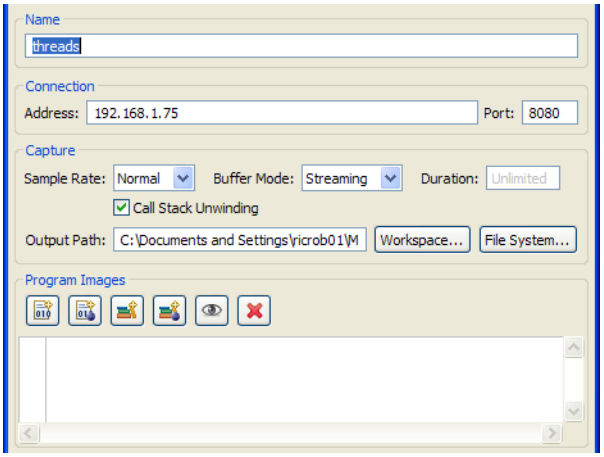

Figure 2-5 Other configuration options

It contains the following settings:

Address The network address of the target. You can also enter the network name of your target here.

Port The network port of the target on which the Gator daemon is listening.

Sample Rate

The target generates periodic measurement interrupts according to the following settings: **Normal**=1kHz, **Low**=100Hz. The **Normal** setting works well for most instances. **Low** is recommended if you have a slow target, or if the target is heavily loaded.

Buffer Mode

The default setting is unbounded streaming of target data directly to your host using a 1MB buffer. You can also use one of the following store-and-forward buffers:

Large 16MB

Normal 4MB

Small 1MB

If you select one of these sizes the capture ends when the buffer is full. This prevents the latency caused by streaming data from the target to the host.

Duration

The desired length of the capture session, in seconds. For example, enter 1:05 for 1 minute and 5 seconds. If you do not provide a value here, the capture session continues until you stop it manually.

Call stack unwinding

Select this checkbox to ensure that Streamline records call stacks. This greatly improves your visibility into the behavior of your target. Make sure to compile your EABI images and libraries with frame pointers using the `-fno-omit-frame-pointer` compilation option.

Output path

Use this field to define the directory location and name of the file generated by the capture and analysis session. By default, the file is saved to a results directory defined by an install variable and given the name `@F_@N.apd`. `@F` is a variable for the given configuration name, while `@N` is a sequential number.

Program images

Use this area to explore your file system and define all of the images and libraries you want to profile. Attach libraries to a Program so that statistics are only recorded in that context.

Click **Add Program...** to add images.

Click **Add Library...** button to attach libraries to programs.

Note

When compiling images, make sure to set the `-g` compilation option to enable debug symbols. Disabling inlining with the `-fno-inline` compiler setting substantially improves the call path quality.

2.3.1 See also

Tasks

- *Setting up the Gator driver and daemon* on page 2-2
- *Setting capture options* on page 2-4
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-14
- *The Code view* on page 2-16

2.4 Capturing and analyzing data for the threads example

After you have configured your capture options, you can capture, analyze, and view a report in Streamline.

To capture data:

1. Click **Start capture**, located next to the **Address** field in the ARM Streamline data view, to start the profiling session. If you have defined your target correctly, an analysis file appears in the ARM Streamline Data view with a **Stop** button.

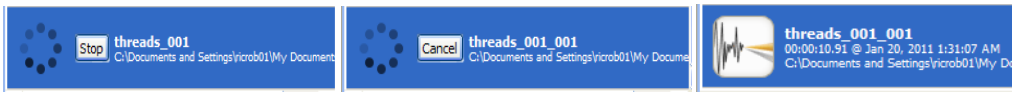

Figure 2-6 ARM Streamline Data View: From working .apc file to .apd file

Note

`.apc` files contain all of the raw data collected on the target during the capture session. `.apd` files are created on the host from the information contained in these capture files. `.apd` files are not compatible across different versions of Streamline, but `.apc` files are. If you have upgraded Streamline and rendered your .apd files incompatible, re-run the `.apc` files to create new, compatible `.apd` files.

2. During the collection of profiling data, switch to your target and open a terminal.
3. Navigate to the `threads` directory
4. Enter `./threads` on command line to execute the `threads` example on your target.
5. After you have run threads, return to your host machine and click the **Stop** button. This action processes the data and makes it ready for report viewing.

2.4.1 Viewing the .apd report and analyzing the captured data

When the capture completes, Streamline automatically opens the profiling report. To open the report again, double-click on `threads_001_001` in the Streamline Data view.

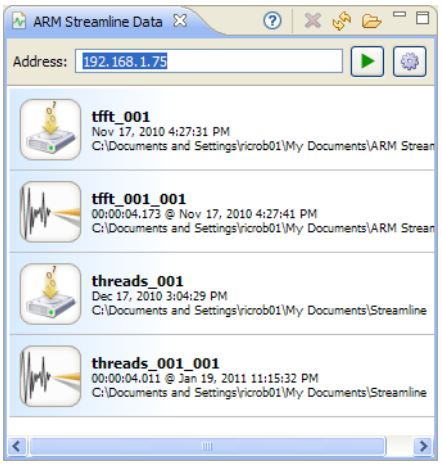

Figure 2-7 ARM Streamline Data view

To analyze the captured .apc data again with other settings, double-click on the captured data icon.

You can use the resulting settings dialog box to add or remove images and libraries. Doing so ensures that your final reports reflect only the execution on which you want to focus.

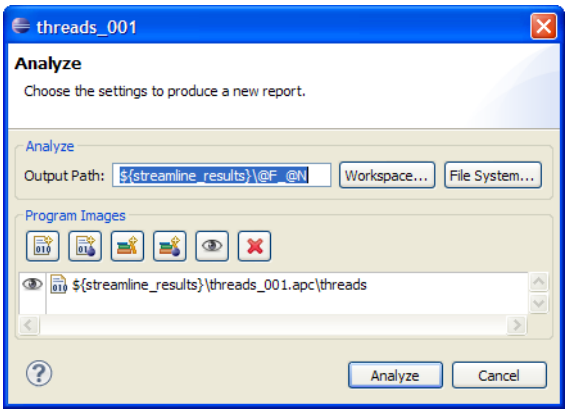

Figure 2-8 Changing the settings for a new report

2.4.2 See also

Tasks

- *Setting up the Gator driver and daemon* on page 2-2
- *Setting capture options* on page 2-4

Reference

- *Other capture options* on page 2-6
- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-14
- *The Code view* on page 2-16

2.5 The Timeline view

After you have successfully generated a report, Streamline opens it automatically and displays the Timeline view.

Figure 2-9 The Timeline view

The Timeline view breaks up its data into bins, a unit of time defined by the unit drop down menu at the top of the view. For example, if the unit is set to 100ms, every point in the charts and every color-coded bin in the processes section represents trace data captured during a 100ms window.

The Timeline view contains calipers that you can use to set the specific window of time on which you want to focus. Streamline updates each report based on where the in and out marker are set, so if you wanted to focus on the second or so `threads` executed, drag the left caliper to the left boundary of threads activity and the right caliper to the right boundary. To locate this period of threads execution, study the processes section of the Timeline view. The **threads** process is bright red during the time it ran.

2.5.1 Charts

Streamline collects data for the charts from hardware and software performance counter resources. The data is dependent on the type of system you use. For SMP systems, disclosure arrows indicate that you can expand the data to show collection per core.

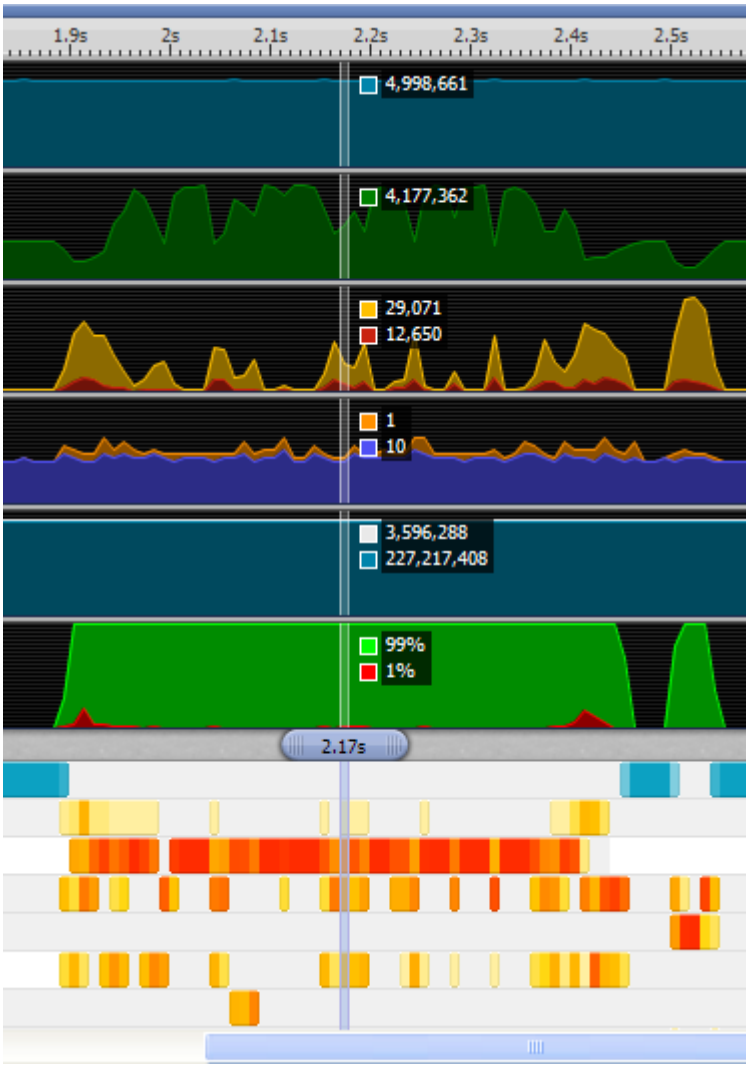

Figure 2-10 Timeline marker

2.5.2 Processes

The Processes section of the Timeline view shows you the active processes in each bin. The entries are derived from process/thread trace data from the Linux kernel scheduler. Weighted colors reflect approximate allocation.

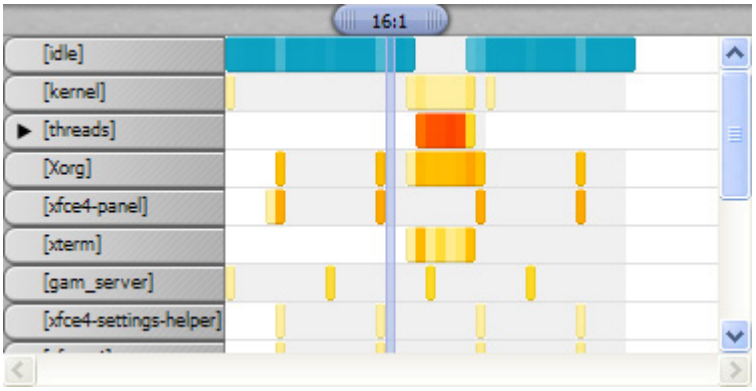

Figure 2-11 Process bars

White Inactive

Gray Active

Yellow to red

Responsible for some percentage of total instructions during this bin. Red indicates a higher percentage.

Notice that in the `threads` example, all bins are white until the process starts. When `threads` has started, the bins turn bright red.

2.5.3 Detail bars

The detail bars show the sampling points in the selected trace bin. Selecting a bar jumps you into the relevant context in the Call Paths view.

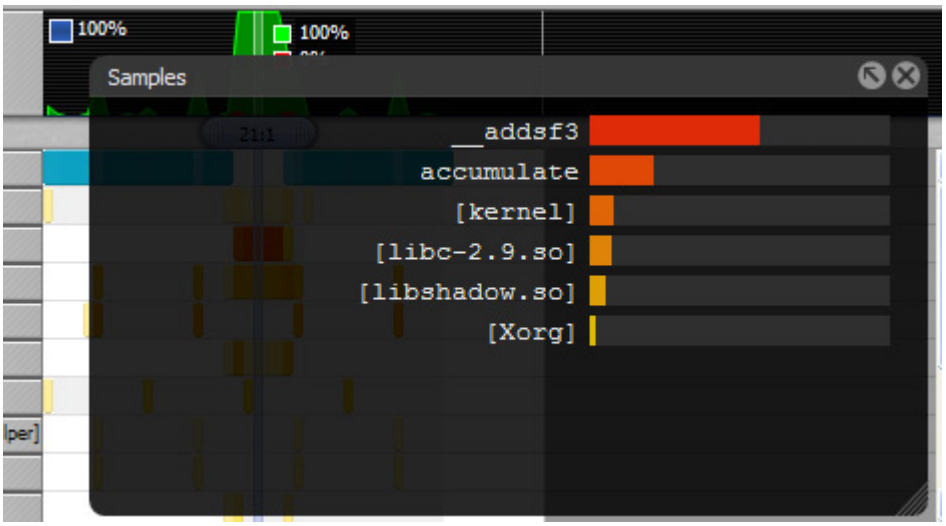

Figure 2-12 Timeline Detail Bar

2.5.4 X-Ray mode

X-Ray mode changes the process trace from an intensity map of time, to a mode that highlights core affinity. In this mode, the bars show the mapping of software threads to processor cores.

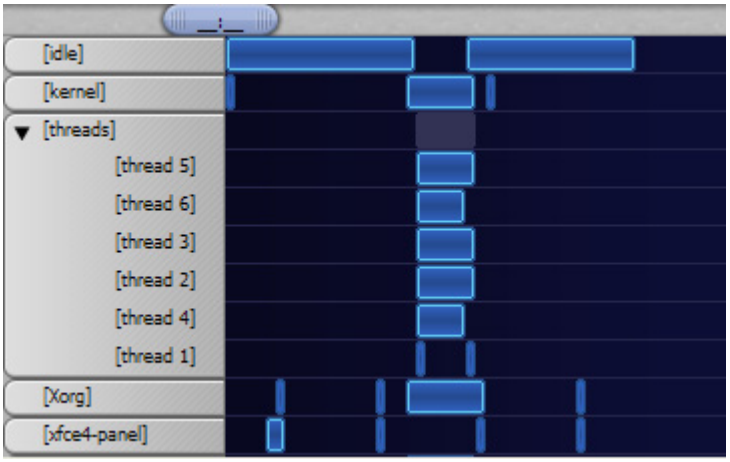

Figure 2-13 X-Ray mode

2.5.5 See also

Tasks

- *Setting up the Gator driver and daemon* on page 2-2
- *Setting capture options* on page 2-4
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Other capture options* on page 2-6
- *The Call Paths view* on page 2-14
- *The Code view* on page 2-16

2.6 The Call Paths view

Use the Call Paths view to see statistics for the whole system. The report presents data hierarchically. Expand the Processes and Threads to expose the Function children.

To get the best results from the Call Paths view, compile your programs and libraries with frame pointers using the `-fno-omit-frame-pointer` compiler flag, and check the call stack unwinding checkbox in the Capture Options dialog box.

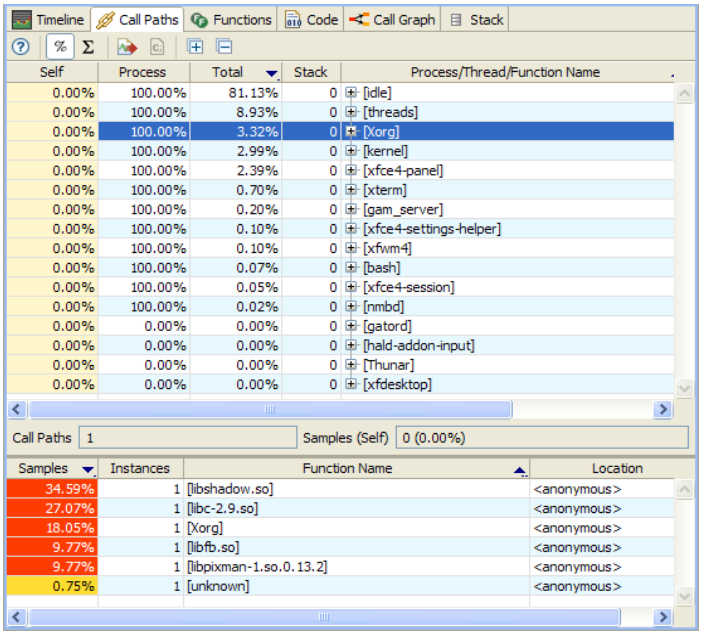

Figure 2-14 The Call Paths view

The Call Paths view shows you statistics for each process, thread, or function in the following categories:

Self Streamline measures time when it records the program callstack on interrupt events. On each interrupt, it attributes the sample to the function that was active. It calculates percentages against the process total.

Process Process time works the same as Self time, only the values here represent the time spent in the function, thread or process and its children. Percentages are calculated against the process total.

Total Total time works the same as Process time, only the values here represent percentages calculated against all collected samples.

Stack The number of bytes used by the stack after the call of this function.

Process/Thread/Function Name

The name field also includes the disclosure control. Click the plus button to open a thread, process or function up to expose its children.

Location The **Location** field lists the name of the source file that contains the function next to a line number of the function start.

Double-click on any item in the Call Paths view to open the functions view with the relevant function selected.

2.6.1 See also

Tasks

- *Setting up the Gator driver and daemon* on page 2-2
- *Setting capture options* on page 2-4
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Other capture options* on page 2-6
- *The Timeline view* on page 2-10
- *The Code view* on page 2-16

2.7 The Code view

The Code view helps with the discovery of function-level hot spots. It flattens statistics and displays them at the source and disassembly levels.

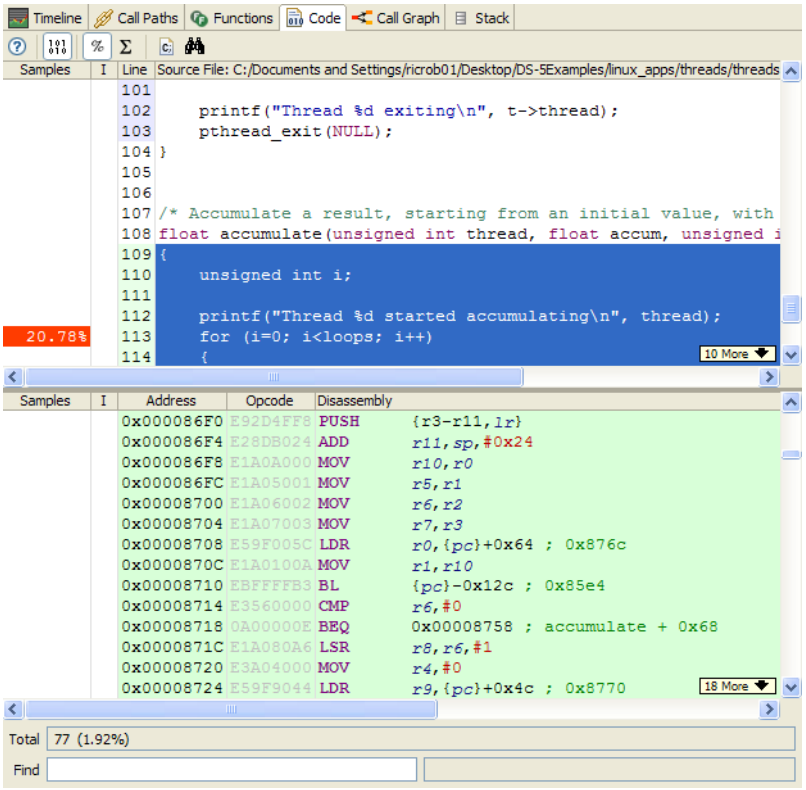

Figure 2-15 The Code view

The Code view presents the percentage each source line or disassembly entry contributed to the total samples collected for the function.

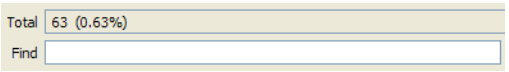

Figure 2-16 The Totals panel

2.7.1 See also

Tasks

- *Setting up the Gator driver and daemon* on page 2-2
- *Setting capture options* on page 2-4
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Other capture options* on page 2-6
- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-14

Chapter 3
Troubleshooting

If you are having trouble running Streamline, consult this collection of solutions to common problems.

The following topics describe how to troubleshoot common Streamline issues:

3.1 Target connection issues

Each of the error messages provided by Streamline on a connection failure indicates a different issue:

Symptom You receive the following error message: `Unable to connect to the gator daemon at` *`your_target_address`*`. Please verify you have installed the gator daemon on your target and it is running. Installation instructions can be found in:` *`DS-5 root`*`/arm/README_Streamline.txt`

Solutions Make sure the Gator daemon is running on your target. Enter the following command in the shell of your target:

```
ps -d | grep gatord
```

If this command returns no results, `gatord` is not active. Start it by navigating to the directory that contains gatord and entering the following command:

```
sudo ./gatord &
```

Re-try connecting to the target.

If `gator` is active and you still receive the above error message, try disabling any firewalls on your host machine that might be interfering with communication between it and the target.

Symptom You receive the following error message: `Unknown host`

Solution Make sure that you have correctly entered the name or IP address of the target in **Address** field. If you have entered a name, try an IP address instead.

Symptom You receive the following error message: `Unable to launch. Only one instance of Streamline may be running on this machine. Please close all instances of Eclipse and try again.`

Solution Stop any other running Streamline session and re-try connecting to the target. If you cannot find another session, try closing Eclipse for DS-5, then re-starting it.

3.1.1 See also

Reference

- *Report issues* on page 3-3

3.2 Report issues

If the data in your reports seems incomplete, you might not have included a compilation option essential to Streamline. Common report problems and solutions are:

Symptom Streamline does not show any source code in the Code view.

Solution Make sure that you used the `-g` option during compilation. Streamline must have debug symbols turned on to match instructions to source code.

Symptom Streamline does not show source code for shared libraries.

Solution Add the libraries using the Capture options dialog box. Click **Add Library...** in the images section, navigate to your shared library, and then add it.

Symptom The data in the call paths view is flat. The presented table is a list, rather than a hierarchy.

Solution Use the `-fno-omit-frame-pointer` option during compilation and be sure to check the **Call Stack Unwinding** option in the capture options dialog box.

Note

Streamline does not walk the stack for kernels or statically linked drivers. Both generate flat data in the call paths view.

Symptom Functions that you know are highly used are missing from the reports. Other functions might seem artificially large.

Solution This can be due to code inlining done by the compiler. To turn inlining off, add `-fno-inline` as an option during compilation.

3.2.1 See also

Reference

- *Target connection issues* on page 3-2