Chapter 6

CHUD Tools

Sampler and `gprof` can help you find the slow spots in your code. But to learn more about your program's performance, use the Computer Hardware Understanding Developer (CHUD) Tools. The CHUD Tools contain both graphical and command-line tools to measure your program's performance and provide tips to make your code run faster. This chapter covers the most important CHUD Tools: Shark, Saturn, `amber`, `simg4`, `simg5`, and `acid`.

Shark

If you're going to learn to use only one performance tool, you should learn Shark. Shark is a source line profiler, telling you the amount of time your program spends in each function and the amount of time your program spends in each line of code. Some additional things Shark can do include.

- Recording performance events like cache hits, cache misses, virtual memory page-ins, memory reads, and memory writes.
- Providing advice to fix performance problems in your code.
- Breaking your code down to its assembly language statements.
- Profiling Java programs.

In version 4.0 Shark began to incorporate many of Sampler's capabilities, including the ability to profile selected functions and the ability to sample during memory allocations.

Like Sampler, Shark periodically takes a sample and records the functions on the call stack. Additionally, Shark keeps track of the amount of time your program spends in each function and can keep track of performance events.

Shark is a powerful program with many options, which can make learning Shark and reading this chapter difficult. As you read this chapter, remember that you don't have to use every option Shark offers. You can start by using Shark as a normal profiler to find where your program spends the most time. As you grow more comfortable with Shark, you can explore the options and harness the power the options provide.

Configuring Shark

If all you care about measuring is where your program spends its time, running Shark is easy. Choose a program to sample and click the Start button to start profiling. But if you configure Shark before profiling you can measure a lot more. If you don't want to be overwhelmed with configuration options right now, you can skip ahead to the section "Profiling Your Program" and start using Shark. Java programmers should read the "Java Traces" section before skipping ahead.

When you launch Shark, the main window opens. This window can have two or three pop-up menus. The leftmost pop-up menu contains the list of available profiles. Choose Edit Config from this pop-up menu to open Shark's configuration window.

Shark comes with 18 profiles, which I detail in upcoming sections. You can create customized profiles by selecting the profile closest to your needs and clicking the Duplicate button. Double-click the copy to change its name.

A Brief PowerPC Processor History

Because the CHUD Tools report performance problems on specific PowerPC processors, you should know a little bit about the processors. The earliest Macs capable of running Mac OS X use the PowerPC 750 processor, also known as the G3. Macs with a G3 processor include the early iMacs without a flat-panel monitor, early iBooks, G3 Powerbooks, and blue and white towers. If your program runs well on the PowerPC 750, it will run well on any Mac that can run Mac OS X.

In 1999 Apple started shipping G4 Macs. The first G4 Macs used the PowerPC 7400 chip. The biggest improvement in the 7400 from the 750 is the AltiVec unit, also known as the Velocity Engine by Apple and the vector unit by IBM. The AltiVec unit uses 128-bit registers that can store four 32-bit values or eight 16-bit values. AltiVec instructions work on these 128-bit registers, which means a single instruction works on multiple data (SIMD). By having one instruction for all the data in the register instead of one instruction for each piece of data, AltiVec code can run four to eight times faster.

Shortly after the first G4 Macs appeared, the PowerPC 7410 replaced the 7400. The 7410 is a more energy efficient version of the 7400. To keep up with Intel and AMD's processors, Motorola developed the PowerPC 7450 chip, which is an improved version of the 7410. What distinguishes the 7450 from other PowerPC chips is Level 3 cache, which stores recently used instructions and data. The 7450 began to appear in high-end G4 towers in 2001 and started to appear in all G4 towers in September 2002. G4 Macs can have one of three processor architectures: the PowerPC 7400, 7410, and 7450.

In 2003 Apple started shipping G5 Macs, which use the PowerPC 970 processor. IBM developed the PowerPC 970FX, which is a more energy efficient version of the 970, in 2004. All Macs with a G5 processor, including the pizza box iMac, use either the PowerPC 970 or 970FX processor. The PowerPC 970 is the first 64-bit processor to appear in Macs. The G3 and G4 Macs have 32-bit processors, which limit the memory address space to 4 GB. 64-bit processors have a theoretical limit of (2 ^ 64) bytes of memory, which means G5 Macs can have more than 4GB of memory installed on them. The 970 also runs at higher clock speeds, has a faster system bus, has faster memory, and can execute more instructions simultaneously.

Time Profiles

As its name suggests, the time profile measures the amount of time your program spends in each function. Only C, C++, and Objective C programs can use the time profile. Shark has its own profiles for Java programs.

The all thread states time profile works the same way as watching for timed samples in Sampler, which I covered in Chapter 4. Shark takes a sample at a time interval that you specify. The advantage of using the all thread states time profile is simplicity. All you have to tell Shark is the sampling interval.

The all thread states time profile's simplicity doesn't mean the regular time profile is difficult to use. There are more settings you can customize in the regular time profile, giving you greater flexibility at the cost of slightly higher complexity. The regular time profile has the following advantages:

- Higher sampling rates, up to 20000 samples per second. The all thread states time profile has a limit of 1000 samples per second.
- The ability to record low-level performance events like cache misses.
- The ability to set a limit on how long Shark profiles your program.
- The ability to profile more than one program at a time.

Function Trace

In the function trace profile, Shark records a sample when your program calls one of the functions you tell Shark to look for. The function trace profile is the equivalent of watching for function calls in Sampler, which I covered in Chapter 4. To add a function to the watch list, type the name of the function in the text box next to the Add button and click the Add button.

Java Traces

The Java traces profile Java programs. There are three built-in Java traces. Java alloc trace records a sample when the program allocates memory. Java method trace records the entry into each method your Java program calls. Java time trace records a sample at regular time intervals.

To profile a Java program in Shark, the program must have the virtual machine option `–XrunShark` set. To set the virtual machine option in Xcode:

1) Double-click the target name in the Groups and Files area of the project window to open the target settings window.
2) In the Info.plist Entries section of the target settings window, select Pure Java Specific.
3) Add the flag `–XrunShark` in the Additional VM Options text field.

Shark can profile only running Java programs. If you try to launch a Java program from Shark, Shark will launch the program, but it won't do any profiling. To save yourself some aggravation, make sure your Java program is running before profiling it with Shark. There are two ways to ensure your program is running.

- Launch your program before you start profiling with Shark.
- Launch your Java program from Xcode by choosing Debug > Launch Using Performance Tool > Shark. When you launch from Xcode, your program launches before Shark starts profiling.

Data Cache Miss Profiles

Shark comes with six data cache miss profiles, which tell Shark to take a sample when there's a data cache miss. A data cache miss occurs when the CPU is unable to find the data it needs in the cache. Three of the profiles are for L1 data cache misses and the other three are for L2 data cache misses.

The data cache miss profiles are also separated by processor. G4 profiles measure data cache misses on PowerPC 7400 and 7410 processors. G4+ profiles measure data cache misses on PowerPC 7450 processors. G5 profiles measure data cache misses on PowerPC 970 and 970FX processors.

Malloc Trace Profile

When you use the malloc trace profile, Shark records a sample every time your program makes a memory allocation. The malloc trace profile does what watching for memory allocations does in Sampler, which I covered in Chapter 4. Why would you want to use the malloc trace profile instead of using Sampler? The malloc trace profile also lets you detect memory leaks, which allows Shark to act as a combination of the Sampler and MallocDebug tools.

Memory Bus Bandwidth Profile

You must have a G5 processor to use the memory bus bandwidth profile. When you use the memory bus bandwidth profile, Shark records every memory read and write your program makes. Shark slices the reads and writes into individual beats. A *beat* is a transfer the size of the data bus, which is 16 bytes on G5 processors. If your program reads 128 bytes of memory, Shark records the eight beats that make up the 128 bytes. Recording beats measures the G5's memory bus bandwidth.

Static Analysis Profiles

The static analysis profiles differ from the rest of the Shark profiles in that they don't require your program to be running. Shark examines your code and reports its analysis to you. There are two static analysis profiles: function browser and problem search.

The function browser profile creates one listing for each function in the program or file you're profiling. By using the function browser profile, Shark can examine your source code and provide code tuning advice to make your code run faster.

The problem search profile examines your code for possible performance problems. You specify the CPU to use to look for problems and specify the severity of problems to look for. Use the problem search profile to look for performance problems on a variety of CPUs without having to run your program.

VM Faults Profile

The VM Faults profile tells Shark to take a sample when virtual memory page faults take place. A page fault occurs when the operating system can't find a page of memory in physical memory. You get to specify how many page faults occur between samples.

Setting the Sampling Rate

The time profile has timing settings that let you specify the following information:

- How long Shark should wait before starting to sample your program. You can specify the time to wait in microseconds, milliseconds, or seconds.
- How long Shark should profile your program before stopping. You can specify the time to profile in microseconds, milliseconds, or seconds.
- The sampling rate, which you can specify in microseconds, milliseconds, or seconds. Shark's default sampling rate is 1000 samples per second.

Recording Performance Events

What sets Shark apart from other performance tools is its ability to record performance events. There are three categories of performance events.

- CPU events, such as cache hits, CPU stalls, and wrongly predicted branches.
- Memory events, like reads and writes. The reads and writes can occur in RAM, in PCI memory, or in AGP memory.
- Operating system events, such as file reads, file writes, system calls, and exceptions.

To set the performance events to record, your profile must use CHUDDataSource. If you select CHUDDataSource from the list of available plug-ins, the lower right portion of the configuration window shows a list of available events to record and a list of events you're recording, which you can see in Figure 6.1. To add an event to the recording list, select it from the available events list and choose Counter from the pop-up menu in the Mode column.

If you choose Trigger from the pop-up menu in the Mode column, the event becomes a trigger. When you make an event a trigger, Shark records a sample when the trigger event occurs instead of when a period of time passes. The advantage of using an operating system event or CPU event as a trigger is you can profile multiple programs without having to profile every running program. When you use a timer trigger, Shark profiles every program or profiles just one program. If you're interested in how much time your program spends in each function, use a timer as a trigger.

All Macs can use operating system events as triggers. Macs with the Power PC 7450 and 970 processors as well as Macs with version 1.3 or later of the PowerPC 7410 processor can use CPU events as triggers. If you make a CPU or operating system event a trigger, go to the Trigger Settings section of the configuration window. Use the Trigger Settings section to tell Shark how many trigger events must occur before Shark takes a sample.

Sometimes when you try to add a performance event, Shark won't be able to add it. There are two possible reasons for Shark being unable to add an event to the list of events to record. First, you exceeded the event limit. The exact limits depend on your computer, but every Mac can record at least four CPU events, four memory events, and four operating system events. Second, the event you want to add doesn't apply to your Mac. The available event list shows the events for all Macs. Some of these events aren't available on your computer. The program PMC Index, which is part of the CHUD Tools, lets you see the performance events available for each CPU, memory controller, and version of Mac OS X.

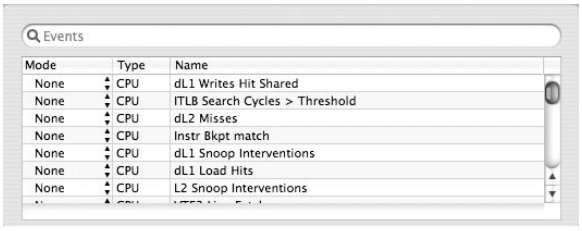

Figure 6.1

Performance event list.

Advanced Configuration Options

Choosing Advanced from the View pop-up at the bottom of the configuration window opens the advanced view, where you can choose the data sources Shark uses for the profile. Each of Shark's built-in profiles has one data source, but you can use multiple data sources by selecting the appropriate checkboxes. Some data sources don't work well together. Including CHUDDataSource and SamplerDataSource is pointless because CHUDDataSource does everything SamplerDataSource does.

Selecting a data source fills the lower right portion of the configuration editor with configuration options for that data source. Only the CHUDDataSource advanced configuration options are interesting; the other data sources' advanced configuration options are the same as the simple configuration options.

Choosing a Trigger

Because many of the trigger settings in the simple and advanced views are identical, I'm going to focus on the trigger settings that have no equivalent in the simple configuration. For the identical settings, it doesn't matter where you set them. They are usually easier to set from the simple view.

The simple view gives you three possible triggers: timer, CPU event, and operating system event. The advanced view gives you a fourth trigger: the function `chudRecordUserSample()`, which is part of the CHUD framework. This function tells Shark to take a sample, allowing you to control Shark from your source code. To use `chudRecordUserSample()` with your program, you must add the CHUD framework to your project and add calls to `chudRecordUserSample()` in your program.

When you use a timer trigger, the timer fires at the exact time you specify. The Fuzz checkbox lets you randomize the sampling rate. You can randomize the sampling rate up to 50%. With a fuzz percentage of ten percent and a sampling rate of one millisecond, Shark takes a sample after 0.9 to 1.1 milliseconds pass.

Why would you want to randomize the sampling rate? The sampling rate can affect how your program executes when Shark profiles it. Randomizing the sampling rate reduces its impact, allowing Shark to profile your program more accurately.

The Sample Limit checkbox and text field lets you specify how many samples Shark takes before stopping. You can set both a time limit and a sample limit. Shark stops when it reaches one of the limits.

Filtering Programs to Profile

When you choose an operating system event or CPU event trigger, two radio button groups become enabled. The first group tells Shark the privilege level of programs to profile. There are two privilege levels: user and supervisor. Applications have user privilege. Programs with supervisor privilege are low-level programs the operating system uses, such as kernel extensions. Profiling programs with user privilege is sufficient in most cases.

The second group tells Shark to profile marked or unmarked processes. In most cases you want to profile marked processes, with the marked processes being the programs you want to profile. To mark a process, open the process marker panel by choosing Shark > Process Marker. The process marker panel contains every program you have running. Some processes cannot be marked. These processes have gray text. To mark a process, select it. The background color in the panel changes from white to brown to signify that the program has been marked. To unmark a marked process, select it from the panel.

Should you change any settings in the radio button groups, they have meaning only if you tell Shark to sample all running programs. If you tell Shark to profile only one program, marking processes and telling Shark to sample only programs with user privilege have no effect. Shark profiles only the program you tell it to profile.

Choosing the CPU and Memory Controller

Initially Shark uses your Mac's CPU and memory controller, but you can tell Shark to pretend your Mac has a different CPU or memory controller. Pop-up menus at the bottom of the window let you choose the CPU and memory controller. By choosing a different CPU or memory controller you can simulate how your program runs on another Mac, alerting you to performance problems on other Macs.

If you're running Shark on a Mac with a PowerPC 750 or 7400 processor, you might be tempted to simulate your Shark session on a PPC 7450 or 970 (you can't simulate the PowerPC 7410) so you can use CPU events as a trigger. While you can simulate how your program runs on a PowerPC 7450 or 970, you can't use CPU events as a trigger because the PowerPC 750 and 7400 can't record CPU events. Shark will report an error when you try to profile your program.

Setting Events to Record

Shark uses the PowerPC's performance monitor counters (PMC) to record performance events. There are 4–8 PMCs for CPU events, 4–6 PMCs for memory events, and 4 PMCs for operating system events. When you use the simple view to record performance events, Shark assigns a PMC for each event for you. The advanced view lets you manually set the event each PMC records.

Click the disclosure triangle next to the event type to set the performance events for that type. Figure 6.2 shows the performance monitor counters for CPU events. The PMCs for the other types of events look similar. Choose an event to record from the combo box. Click the button next to the PMC to turn on the PMC. The button's title is Counter when you turn on a PMC. If you're using the PMC as a trigger, the button's title is Trigger.

G5 Specific Events

The G5 processor has its own set of CPU events. If you have a G5 processor on your Mac or you tell Shark to pretend your Mac has one, you can configure these events.

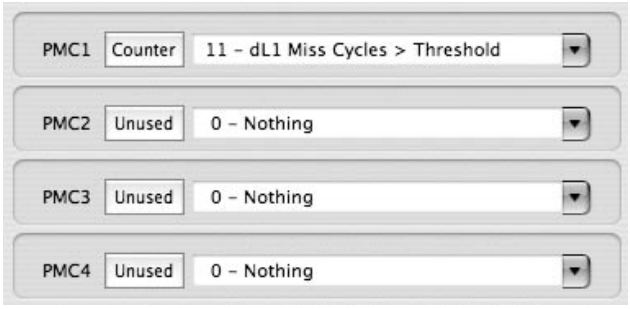

Figure 6.2

Settting performance monitor counters.

Event Multiplexing

G3 and G4 processors have only direct events, where the performance monitor counter (PMC) directly records one event. IBM introduced indirect events with the G5 processor. With indirect events the CPU multiplexes some of the performance events. These multiplexers give the G5 a greater number of possible events to record. Shark lets you configure the multiplexers.

The G5's multiplexers consist of four general lanes and one spec lane. For each general lane you can specify the multiplexer to use as well as the execution unit from which the multiplexer records events. Table 6.1 lists the multiplexers for the general lane. The spec lane can use any of four speculative event multiplexers. The spec lane is for speculative event counting, which occurs when an instruction group stalls or the global completion table (GCT) becomes empty.

By setting up the lanes, you determine the events each of the G5's eight PMCs can record. Lanes 0 and 2 affect PMCs 1, 2, 5, and 6. Lanes 1 and 3 affect PMCs 3, 4, 7, and 8. The spec lane affects PMCs 5 and 7. Suppose you tell lane 0 to use TTM0, which you set to record events from the FPU. By setting lane 0 to use the FPU, only PMCs 1, 2, 5, and 6 can record all the events that occur in the floating-point units.

There are pop-up menus to select the multiplexer to use for each lane as well as menus to set the execution unit for each general-purpose multiplexer, as you can see in Figure 6.3. Keep three things in mind. First, the general lanes can record events from the load/store units instead of using the multiplexers listed in Table 6.1. Second, choose GPS from the TTM1 pop-up menu if you want TTM1 to record events from the storage subsystem. Third, the TTM3 pop-up menu lets you specify whether bytes 2 and 3 come from the lower or upper 32 bits of LSU1. The pop-up menu has the following entries:

- LSU1 2|3, which means bytes 2 and 3 come from the lower 32 bits.
- LSU1 2|7, which means byte 2 comes from the lower 32 bits and byte 3 comes from the upper 32 bits.
- LSU1 6|3, which means byte 2 comes from the upper 32 bits and byte 3 comes from the lower 32 bits.
- LSU1 6|7, which means bytes 2 and 3 come from the upper 32 bits.

Table 6.1 General-Purpose Multiplexers

Multiplexer	Lanes	Execution Units It Can Record From
TTM0	0, 1, 2, 3	Floating Point Unit (FPU), Vector Permute Unit (VMX), Instruction Fetch Unit (IFU), Instruction Sequencer Unit (ISU)
TTM1	0, 1, 2, 3	Instruction Dispatch Unit (IDU), Instruction Sequencer Unit (ISU), Storage Subsystem (STS)
TTM3	2, 3	Load/Store Unit 1 (LSU1)

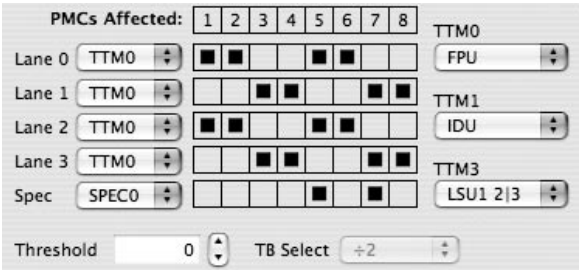

Figure 6.3

Configuring the G5 event multiplexers.

CPU Instruction Matching

The G5 processor has an instruction matching facility you can configure to limit the assembly language instructions Shark samples. There are two decisions to make. First, you must decide whether to use the instruction fetch unit (IFU) or instruction dispatch unit (IDU) for instruction matching. Use the IFU when you know the instructions you want Shark to sample. Use the IDU when you care about the internal instructions, iops, generated from an assembly language instruction.

Second, you must decide which iops Shark samples using the IOP Marking radio button group. You can tell Shark to sample.

- All iops.
- Only iops expanded from architected instructions.
- Only one iop for each assembly language instruction.
- The first iop to travel to the load/store unit for each load/store instruction.

IFU Instruction Matching

The IFU instruction matching facility, shown in Figure 6.4, lets you specify the instructions you want Shark to sample. If you were interested only in AltiVec instructions, you would tell the IFU instruction matching facility to match AltiVec instructions.

When configuring the IFU instruction matching facility, the first task is to choose an Instruction Match CAM register (IMC) row from the pop-up menu. There are six rows: rows 0 through 5. You can configure each row to match a different set of instructions. Select the Enabled checkbox to start matching instructions for a row.

To match instructions for an IMC row, you must set bit masks to determine the instructions that match for that row. There are six bits for the major opcode, 11 bits for the minor opcode, and four bits for the machine state register (MSR). The major opcode identifies the type of instruction. AltiVec instructions have a major opcode of 4, and floating-point instructions have major opcodes 59 and 63. The minor opcode uniquely identifies an instruction from others of the same type.

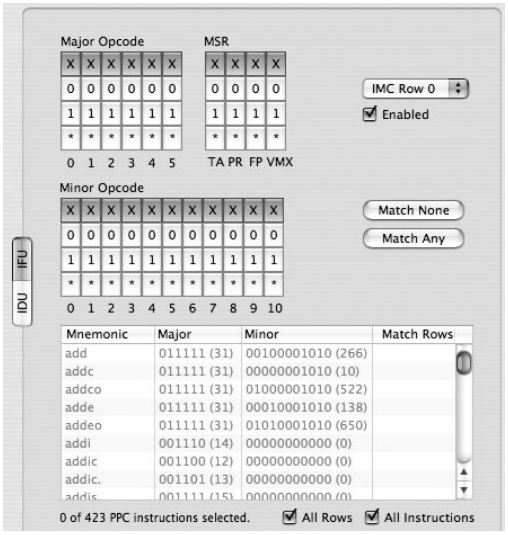

Figure 6.4

IFU instruction matching.

There are four bits you can set for the MSR.

- TA. There is no mention of the TA bit in the PowerPC documentation on the MSR so I have no idea what the TA bit does.
- PR, which is the privilege level bit. Setting PR to 1 means the CPU can handle only non-privileged instructions. Setting PR to 0 means the CPU can handle any instruction.
- FP, which is the floating-point available bit. Setting FP to 1 means the CPU can handle floating-point instructions.
- VMX, which is the vector unit available bit. Setting VMX to 1 means the CPU can handle vector (AltiVec) instructions.

You can set a bit to have any of the following values:

- Never Match. If any of the bits in the major opcode or minor opcode are set to Never Match, no instructions match.
- Match 0, which means an instruction matches if one of its opcode bits has a 0 where you set the bit to be 0.
- Match 1, which means an instruction matches if one of its opcode bits has a 1 in that particular position.
- Don't Care, which means the instruction matches automatically. Setting all the bits to Don't Care means every instruction matches, which defeats the purpose of instruction matching.

When you set more bits to Match 0 or Match 1, fewer instructions match. Setting more bits to Don't Care increases the number of instructions that match.

IDU Instruction Matching

The IDU instruction matching facility, shown in Figure 6.5, lets you specify the instructions for Shark to sample based on the iops the instructions generate. The IDU uses four predecode bits for instruction matching.

- Branch bit (B), which creates an iop from a branch instruction when the bit is set.
- Split bit (S), which splits an instruction into multiple iops when the bit is set.
- First bit (F), which makes the iop the first instruction in a dispatch group when the bit is set.
- Last bit (L), which makes the iop the last instruction in a dispatch group when the bit is set.

Initially the IMRMASK and IMRMATCH are 0000, which means all instructions pass. Setting a bit in the IMRMASK blocks the instructions where you set the bit; setting the B bit blocks all instructions where the B value is 1, BSFL values 1000 through 1111. What do you do if you want to set the instructions you want to pass instead of setting the ones you want to block? That's where the IMRMATCH comes in. Set the appropriate bits in the IMRMASK and set the same bits in the IMRMATCH. Setting a bit to 1 in the IMRMATCH that doesn't have the corresponding bit set in the IMRMASK blocks all instructions.

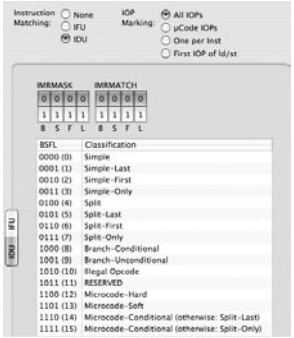

Figure 6.5

IDU instruction matching.

Profiling Your Program

When you launch Shark, the main window, shown in Figure 6.6, opens. The main window is where you tell Shark what to profile and how to profile. This window can have two or three pop-up menus. The leftmost pop-up contains the list of available profiles. Next to the profile list is a second pop-up menu that tells Shark what to do: analyze a file, sample a program (Process) or sample all running programs (Everything). Only profiles that use CHUDDataSource, including the time profile and the data cache miss profiles, can sample all running programs. Only the static analysis profiles can analyze a file. All the built-in profiles can sample a process, or analyze in the case of a static analysis profile. If you create a custom profile, what you can profile depends on the built-in profile you used as a base.

If you choose to profile a single program, a third pop-up menu appears containing all the currently running programs. Choose the program you want to profile from the pop-up menu. If the program you want to sample isn't running, choose Launch. Click the Start button to start profiling. If you launch your program from Shark, a dialog box opens asking for the program to sample. After clicking OK, Shark starts sampling.

If you choose to analyze a file, the third pop-up menu contains the list of files you can analyze. If there are no files to analyze, choose Open. Click the Start button. If you chose Open, a dialog box opens asking for the file to analyze. After clicking OK, Shark starts analyzing the file.

After clicking the Start button, Shark goes to work sampling programs. The sampling continues until you click the Stop button or Shark takes the configured number of samples. When you stop sampling, Shark takes some time to show the results. The reason for the delay is that Shark doesn't do its calculations and analysis until you stop sampling. Waiting to perform the calculations keeps your program running smoothly, but causes a delay when you're finished. To avoid the delay, use batch mode. Batch mode tells Shark not to show the results until you turn off batch mode. Choose Shark > Batch Mode to turn batch mode on and off.

Viewing Shark's Results

After Shark finishes processing the recorded samples, it opens a results window and places the results there. Figure 6.7 shows an example of the results window. Initially Shark shows the results of the dominant process, the program where the CPU spent the most time, and the dominant thread in that process. But you can use the pop-up menus at the bottom of the window to choose another program or thread. If you profile only one program, there's going to be only one program to choose in the pop-up menu.

Figure 6.6

Shark launch window.

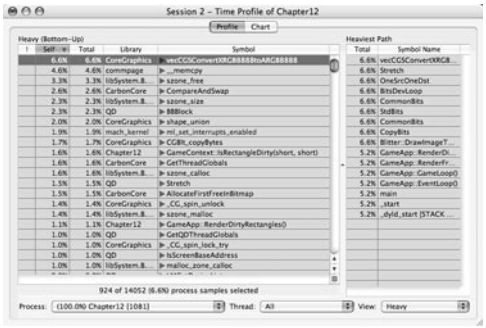

Figure 6.7

Shark results window.

Initially the Shark results window shows the profile view, which is a table of profiling statistics. For each function your program calls, the profile view has five columns of information.

- Tuning advice column, designated by an exclamation point. If there's an exclamation point in the column, clicking the exclamation point brings up advice to make your code run faster.
- Self, which initially is the percentage of samples where this function is at the top of the call stack. For a time profile, the Self column tells you the percentage of time your program spent in this function.
- Total, which initially is the percentage of samples where this function appears in the call stack. For a time profile, the Total column tells you the percentage of time your program spent in this function and its descendants.
- Library, which is the library where the function resides.
- Symbol, which is the name of the function. To see symbols in your program, you must compile your program to generate debugging information. Refer to the section "Before You Debug" in Chapter 2 for instructions on generating debugging information.

In addition to these five columns, the profile view has one column for each performance event you told Shark to record. Shark stores the total number of times a performance event occurred. The event's corresponding column tells you what percentage occurred in a particular function and its descendants.

Heavy and Tree Views

The initial view in the results window is the heavy view. The heavy view is the best view for finding the slow areas of your code. It shows you the calling paths where your program spends a lot of time. There's one listing in the heavy view for every function that appeared at the top of the call stack at least once when Shark took a sample. A function appears at the top of the call stack when your program is inside that function. Shark sorts the listings so the listings that consume the most time appear first. A heavy view listing for a function contains the path from the function back to the top of the call tree, which is the first function your program called. Click the disclosure triangles to move up the call tree.

When you look at a heavy view listing, remember that it represents a path from one function to the top of the call tree. A single function can appear in multiple heavy view listings. The statistics for a function in the heavy view listing reflect the samples taken in the path, not the total number of samples where the function appears. When a function shows a percentage of 20 in the Total column in the heavy view, it means your program spent 20 percent of the time in this path. The heavy view focuses on the statistics for a calling path, not the statistics for an individual function.

When you want to focus on statistics for individual functions, switch to the tree view using the View pop-up menu at the bottom of the results window. The tree view starts displaying functions at the top of the call tree. Clicking the disclosure triangles moves you down the call tree. The tree view gives you a global view of your program's performance. When a function shows a percentage of 20 in the Total column in the tree view, you know your program spent 20 percent of the time in that function and its descendants.

Showing the Call Stack Table

The Shark results window has a button below the vertical scroll bar. Clicking the button makes the call stack table appear in the right half the results window. Selecting a row from the results window fills the call stack table with the function's call stack, the series of functions that called this function. The call stack table lets you view the call stack without having to click a series of disclosure triangles.

Viewing Source Code

Double-clicking a function name in either the results window or the call stack table shows you the source code for that function in the Shark results window, turning the results window into a code browser. Shark highlights the line of code with the most samples.

The code browser lets you look at a function's source code in the high-level language you used to write the code or in assembly language. Looking at a function in a high-level language lets you find the lines of code where your program spends the most time. Looking at the assembly language code provides additional information, such as the number of clock cycles your program spends in each assembly language instruction.

There are three buttons at the top of the code browser that determine what appears in the code browser. Clicking the Source button tells Shark to look at the code in the high-level language it was written in. Clicking the Assembly button tells Shark to look at the assembly language code. Clicking the Both button shows both the high-level language and the assembly language code in the code browser. If a function does not have high-level source code available to view, such as functions in Apple's frameworks, Shark disables the Source and Both buttons.

Customizing the Code Browser

Choose View > Show Advanced Settings to open the advanced settings drawer. The advanced settings drawer lets you customize what appears in the results window and how the data appears. In the Code Browser section of the advanced setting drawer, which you can see in Figure 6.8, there are checkboxes that let you determine the columns that appear in the code browser. If you told Shark to record performance events, selecting the Show Perf Event Columns checkbox tells Shark to add a column in the code browser for each performance event Shark recorded.

The most interesting code browser setting is the CPU Model pop-up menu. With the CPU Model pop-up menu, you can look for code problems on PowerPC 7400, 7450, and 970 processors. The CPU model affects only the assembly language code statistics.

Viewing Assembly Language Code

The advantage of viewing assembly language code is you can see assembly language source code for every function in your program, including the functions in Apple's frameworks. If you want to see what's happening in your program when you make Cocoa function calls, look at the assembly language. Shark initially reports the following data for each assembly language instruction:

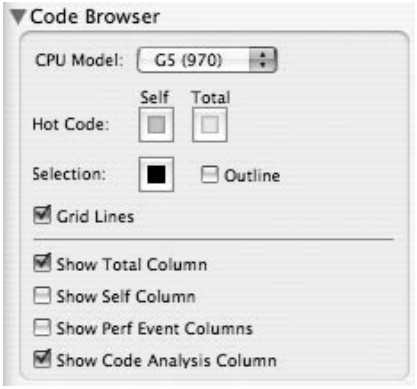

Figure 6.8

Shark's code browser settings.

- Total, which is the percentage of time your program spent in this instruction when the program was in this function.
- Address, which is the memory address for the instruction.
- Code, which is the instruction.
- Cycles, which is the number of processor (clock) cycles spent in this instruction. The number of processor cycles per second equals one billion multiplied by the gigahertz of the processor. A 2 GHz Power Mac G5 has two billion clock cycles per second.
- Tuning advice column, designated by an exclamation point, which provides advice to speed up your code.
- Comment, which alerts you to any performance problems, such as stalls. Deselecting the Show Code Analysis Column checkbox in the advanced settings drawer eliminates the Comment column.
- Source, which lists the file name and line number. Code you didn't write has a blank Source column.

The Cycles column displays two numbers separated by a colon. The first number represents latency, and the second represents throughput. *Latency* is the number of cycles that elapse between an instruction starting execution and the instruction producing a final result. *Throughput* measures the number of cycles between finishing instructions of this type when the corresponding pipeline is full. If an integer instruction finishes, throughput measures the number of cycles it takes for a second integer instruction to finish when the integer instruction pipeline is full. If the instruction has an asterisk in the Cycles column, the instruction causes the pipeline to be serialized, which means the pipeline must wait until the serialized instruction finishes.

Use the checkboxes in the advanced settings drawer to show more columns of information in the code browser.

G5 Assembly Language Options

If you're examining your code using the G5 CPU model, Shark provides additional options. Selecting the Show G5 Dispatch Groups checkbox in the Asm Browser section of the advanced settings drawer, shown in Figure 6.9, adds grid lines around each group of instructions. The PowerPC 970 takes up to four non-branching instructions and one branch instruction and places them into a group. The processor takes the group of instructions, splits the group into individual instructions for execution, and regroups the instructions. By using groups of instructions, the PowerPC 970 can have more instructions in the pipeline at once, over 200 instructions.

Because a group can have up to five instructions, showing the dispatch groups can alert you to performance problems on G5 processors. If you have lots of groups with only one or two instructions in them, you're not running at peak efficiency. Groups containing four or five instructions show that your code is running efficiently.

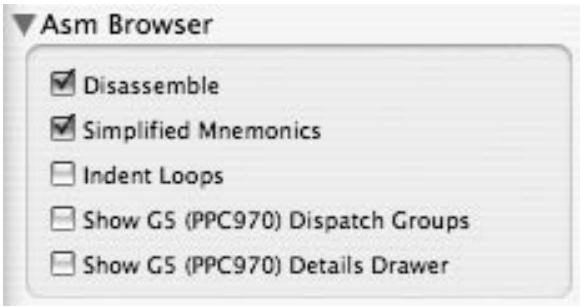

Figure 6.9

Shark's assembly language browser.

Selecting the Show G5 Details Drawer checkbox in the Asm Browser section opens the details drawer, which you can see in Figure 6.10, at the bottom of the code browser. The details drawer tells you the execution unit and group dispatch slot an instruction uses. Selecting instructions in the code browser shows the following information about the instructions in the details drawer:

- The execution units these instructions use.
- The number of groups these instructions belong to.
- The average size of each group.
- The dispatch slots the instructions use.

Assembly Language Reference

Assembly language instructions use mnemonics, making it difficult to figure out what the instruction does. Do you know what the instruction `stswi` does? There are hundreds of PowerPC assembly instructions, many with names that are as difficult to understand as `stswi`, but Shark provides help.

Clicking the PPC Help button opens a window containing the PowerPC assembly language instruction reference manual. Selecting an instruction from the code browser opens the material on that instruction in the reference window. The reference manual provides one to two pages of information on each instruction, enough to give you an idea of what the instruction does.

Viewing High-Level Language Code

You can view high-level language code only for functions you wrote. For code you wrote, Shark loads the function's file in the results window. Shark initially shows the following information for each line of code:

- Total, which is the percentage of time your program spent in this line of code when the program was in this function.
- Line, which is the line number.
- Code, which is the line of code.
- Tuning advice column, designated by an exclamation point, which provides advice to make your code run faster.
- Comment, which alerts you to any problems in your code.

Use the checkboxes in the advanced settings drawer to show more columns of information in the code browser.

Clicking the Edit button opens the source code file in Xcode and takes you to the function you were examining in Shark. Xcode must be running for Shark to open the source code file.

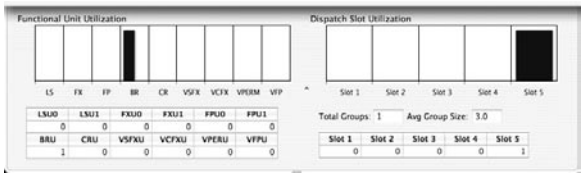

Figure 6.10

G5 details drawer.

Customizing What the Results Window Displays

Earlier I told you the Self and Total columns in the results window display the percentage of samples. I also told you the performance event columns display the percentage of the event. What appears in these columns depends on what you tell Shark to show in them.

The advanced settings drawer, which you open by choosing View > Show Advanced Settings, is where you customize what appears in the results window. Click the Profile tab in the results window to show the Profile Analysis section of the advanced settings drawer. Figure 6.11 shows the Profile Analysis section. The Stats Display pop-up menu tells Shark to display either a percentage or a value. If you'd prefer not to see percentages, choose Value from the pop-up menu.

If you choose to display a value instead of a percentage, the Weight By pop-up menu comes into play. From the Weight By pop-up menu, you can tell Shark to display the value as the number of samples or as the amount of time. Only the Self and Total columns can display the amount of time. If you choose Time from the pop-up menu, Shark displays the number of each performance event you told Shark to record.

The Granularity pop-up menu lets you decide how Shark groups samples. The granularity level determines what a row in the Shark results window represents. Initially the granularity level is Symbol, which means there's one row for each function. Other levels of granularity include the following:

- Address granularity, where Shark groups samples by instruction address. A function can appear in multiple rows when you choose address granularity.
- Library granularity, where Shark groups samples by code library. Library granularity generates one row in the results window for each library in your program. The Symbol column disappears when you choose library granularity.
- Source file granularity, where Shark groups samples by source code file. Source file granularity generates one row in the results window for each source code file in your program. Code you didn't write has the file name `NO_SRC_FILE`.
- Source line granularity, where Shark groups samples by source code file and line number. Source line granularity generates one row in the results window for each line of code in your program. Code you didn't write has the file name `NO_SRC_FILE`. Source line granularity lets you pinpoint lines of code that are slowing down your program.

Viewing Charts

Clicking the Chart tab in the results window changes the results window to show charts. The charts let you examine individual samples. If you wanted to, you could examine every sample chronologically.

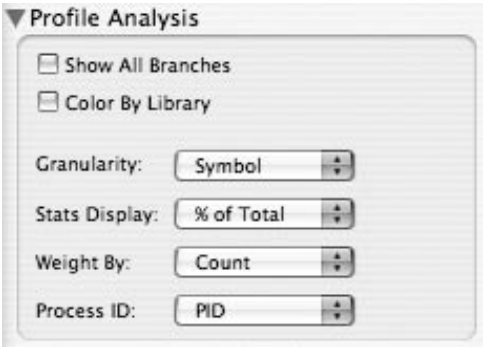

Figure 6.11

Profile analysis drawer.

Call Stack Chart

Every Shark profile has at least one chart, which plots the call stack depth for each sample. Sharp spikes in the graph indicate your program was busy when Shark took that sample. A squiggly line signifies a context switch, which occurs when the operating system switches threads.

Initially the call stack chart type is Saturn, which means Shark draws a continuous bar graph, with one bar for each sample. In the Saturn chart type, user call stacks are blue and kernel call stacks are red. If you open the advanced settings drawer, shown in Figure 6.12, you can change the appearance of the call stack chart. Use the Type pop-up menu. There are three additional options.

- Trace, where Shark draws a line graph and does not differentiate between user and kernel call stacks.
- Delta, where Shark plots the change in call stack depth from the previous sample instead of plotting the call stack depth.
- Hybrid, which is a combination of Saturn and Trace. It plots a line like Trace, but when you select a sample, it fills a block like Saturn.

Performance Event Charts

In addition to the call stack depth chart, there's one chart for each performance event you told Shark to record. Shark plots the number of each event that occurred for each sample. If you open the advanced settings drawer, there's a sum column with a checkbox for each performance event Shark recorded. Selecting the sum checkbox plots the event as a running total with each sample adding to the previous samples. Plotting as a running total gives the graph a gradual upward slope as you move from left to right.

Viewing Individual Samples

Initially Shark fits the whole graph in the window. Fitting all the samples in the window provides an overview, but makes looking at an individual sample difficult. The slider below the charts zooms the graphs in and out, allowing you to focus on certain samples.

Clicking a chart fills the right side of the window with the call stack Shark recorded for that sample. Double-clicking a function in the call stack displays that function's source code in the results window. Use the arrow keys to navigate the samples. Pressing the right arrow key moves you to the next sample, and pressing the left arrow key moves you to the previous sample.

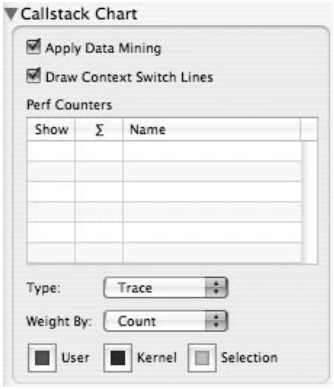

Figure 6.12

Call stack chart advanced settings.

For most charts clicking inside a chart draws a yellow line where you clicked. The yellow line denotes the current sample. The exceptions are the Saturn and Hybrid types of the call stack depth chart. In these cases clicking the chart colors the sample's call stack pyramid yellow. The top of the pyramid is the function at the top of the call stack and the base of the pyramid is the first function your program called.

Data Mining

Shark provides you with a staggering amount of information, making it difficult to find the information you need. Shark provides data mining capabilities to help you find what you need.

Choose View > Show Advanced Settings to open the data mining drawer, which you can see in Figure 6.13. The Apply to Heavy and Apply to Tree checkboxes let you specify the views where the data mining takes effect. Selecting only one of the checkboxes let you have data mining in one view and no data mining in the other view.

Excluding Portions of Code

The easiest way to find the information you want is to exclude the information you don't want. Excluding a function hides it in the Shark window and passes its costs to the functions that called it. Suppose function A calls function B and your program spent one second in A and three seconds in B. If you exclude B, Shark says your program spent four seconds in A, the one second the program spent in A plus the three seconds it spent in B.

Excluding Areas of Code

There are a series of checkboxes in the Data Mining section of the advanced settings drawer that let you exclude larger areas of your code. Selecting the Charge System Libraries to Callers checkbox tells Shark to pass the calls your code makes to system libraries to the functions in your code that made the calls. Passing the calls to your functions lets you see the functions in your code where your program is spending the most time. With this checkbox unselected, Shark reports lots of lower level function calls you didn't write. Selecting the checkbox moves these lower level calls to functions you wrote.

Selecting the Charge Code without Debug Info to Callers checkbox hides the functions that don't have source code available. In most cases, selecting the Charge Code without Debug Info to Callers checkbox excludes the code you didn't write, allowing you to focus on your code.

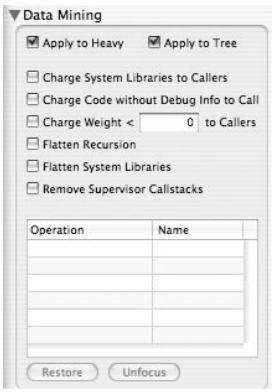

Figure 6.13

Data mining drawer.

If eliminating all the code you didn't write goes too far, the Remove Supervisor Callstacks checkbox may be for you. Selecting the Remove Supervisor Callstacks checkbox excludes functions in supervisor processes from Shark's display.

The Hide Weight < text field lets you exclude less important functions. How Shark excludes the functions depends on what you choose in the Weight By menu. If you weight by count, Shark excludes functions whose sample count is less than what you enter in the text field. If you weight by time, Shark excludes functions whose time is less than what you enter in the text field. The text field uses seconds as the unit of measurement. If you want to exclude functions where your program spent less than 25 milliseconds, put the value .025 in the text field.

Excluding Individual Functions and Libraries

There are two ways to exclude a function. Selecting a function and choosing Data Mining > Charge to Callers removes the function from the results window and passes the costs to the functions that call this function. Selecting a function and choosing Data Mining > Remove Callstacks with Symbol removes the call stacks where the function appears. Removing call stacks changes the percentages Shark displays in the results window because the total number of samples is lower due to the removal of the call stacks.

Choose Data Mining > Charge Library to Callers to charge calls made to a library's functions to the callers. If your program spends a lot of time in functions of a library you didn't write, charging the calls to that library lets you see the areas of your code that are responsible for consuming so much time.

When you exclude a library or function, Shark adds it to the list of excluded names. To restore the library or function, select it from the list and press the Delete key. The Restore All button restores all the excluded libraries and functions.

Flattening Libraries

Flattening a library excludes all the functions in a library except for its entry points. The costs of the excluded functions go to the entry points. A library's entry point is a function that is available to other programs. Apple's libraries use entry points for your program to enter the libraries. The entry points are functions your program calls to get into a particular library.

To flatten one library, select one of its functions from the Shark window and choose Data Mining > Flatten Library. The Flatten System Libraries checkbox lets you flatten all the system libraries your program links to. The Flatten Recursion checkbox tells Shark to flatten recursive function calls. When a function calls itself, the function call is recursive. Selecting the Flatten Recursion checkbox places all calls to the recursive function in one call tree listing, allowing you to focus on the recursive function itself, not each individual call to the function.

Focusing on Functions

Excluding functions and flattening libraries hide the information you don't need, but focusing lets you concentrate on the functions you're interested in. There are two ways to focus: focus on a function or focus on a function's callers. Focusing on a function sets the root of the call tree to the focused function, allowing you to concentrate on the routines the focused function calls. You can focus on a function by selecting it and choosing Data Mining > Focus Symbol.

Focusing on a function's callers sets the leaf of the call tree to the function, allowing you to see which routines call the function. Select a function and choose Data Mining > Focus Callers of Symbol.