Application Note

ubicom32-prof

Profiling an application

Contents

1 Synopsis 1
2 Description 2
3 Options 3
4 Notes 3
5 Page Tabs 4
5.1 Summary Page 4
5.2 Threads Page 4
5.3 Functions Page 6
5.4 Hazards Page 6
5.5 Hazard Details Page 7
5.6 Locks Page 7
5.7 Loops Page 7
5.8 Applications Page 7
5.9 Graphs Page 7
5.10 Statistics Page 7
5.11 Working Set Page 7
5.12 Heap Page 8
5.13 Latency Page 8
5.14 Instructions Page 8
5.15 HRT Threads Page 8
6 Menus 9
6.1 File Menu 9
6.1.1 Gprof 9
6.1.2 Hazards 9
6.2 View menu 9
6.3 Actions menu 10
6.3.1 Log File Format 10
7 Possible ubicom32-prof Problems 10

1 Synopsis

```
ubicom32-prof -ip:board_ip_address [-D:distro_path] [-a] \
   [-t:seconds] [-S[:seconds]] [-log[:seconds]] \
   [-lib:library_elf_path] \
   bootexec.elf vmlinux.elf [app_or_library_elf_file]
```

2 Description

Profiling has three components. The ipProfile package is included in your project. It runs in the GDB thread, and periodically samples the other threads. The profiler daemon (profilerd) runs on the board to send the samples to a PC, over the LAN, using UDP. Use "make menuconfig" to enable profiling in your project, and make sure profilerd is running on the board.

ubicom32-prof is a tool that runs under Windows on a PC, or under Linux, using Wine, to display and interpret the data sent by the ipProfile package. It has a real-time display of hundreds of performance parameters that can be saved into a text file. When ubicom32-prof is run, it connects to profilerd running at the specified IP address, and causes ipProfile to start taking samples. When ubicom32-prof is not running there is no overhead on the baord for profiling since ipPRofile and profilerd are both idle.

The profiler can display virtual counters, which represent any information the user might want to count and display. To count events, declare a global counter:

```
virtual unsigned int my_counter;.
```

Register the counter:

```
 profile_register_performance_counter(&my_counter, "My Counter Label");
```

The contents of my_counter will be graphed in the profiler tool with the specified label. The same API is available in both Ultra and the Linux kernel. In Linux, include "asm/profile.h".

The -ip parameter specifies the IP address of the board under test.

The -D parameter specifies the path to the root of the distribution, and defaults to ".".

The -a parameter causes key files to be autoloaded. There is no need to specify path names for autoloaded files. Autoloaded files are: ultra\projects\mainexec\ultra.elf uClinux\Linux-2.6.x\vmlinux uClinux\user\busybox\busybox_unstripped uClibc\lib\libuClibc-0.9.30.1.so uClibc\lib\ld-uClibc-0.9.30.1.so

ubicom32-prof searches for files specified on the command line. The search order is: path_name distro_path\path_name library_elf_path\path_name for each library path specified distro_path\library_elf_path\path_name for each library path specified distro_path\uClibc\lib\path_name distro_path\uClinux\user\path_name

If -a was not specified, the first file parameter must be the unstripped binary of the `mainexec` Ultra project. The second file parameter must be the unstripped binary of your linux OS. Additional optional file parameters specify application or library unstripped binary files (elf format). These additional files are dynamically loaded in the system, so the Profiler tool needs to know their load addresses. profilerd sends the addresses of all loaded applications and libraries to ubicom32-prof when ubicom32-prof connects or restarts. If new processes are started after ubicom32-prof connects, the "restart" command must be selected in ubicom32-prof to force the addresses to be sent again.

When "Save" is selected, the Profiler tool saves the displayed data to **ultra.txt**, and also saves several other files. The displayed data is also saved in **ultra.pgr** in a binary format. This file can be read by the Profiler tool using **ubicom32-prof ultra.pgr**, and will show all of the saved data using the Profiler tool interface. It creates **hazard.txt**, which gives the locations of the instructions causing performance loss due to hazards.

3 Options

-ip:ip_address

The **-ip** option tells the Profiler the IP address of the board under test. The Profiler will connect to profilerd on the board and read the **/proc/maps** file to find the starting address for each dynamically loaded module.

-D:distro_path

The **-D** option specifies the path to the distribution base. The default is ".".

-a

The **-a** options tells the profiler to autoload key files.

-t

The **-t** option tells the Profiler to run for the specified period of time, then save all files and exit.

-S[:seconds]

The **-S** option tells the Profiler to exit when it does not get a packet for the specified number of seconds (default 10, minimum 1), and save all files.

-log[:seconds]

The **-log** option causes the Profiler to log data at regular intervals to the file **profile_log.txt**, and all statistics data to the file **profile_stats_log.txt**. The log files contain one line of comma-separated values giving the performance for each log interval. The optional "seconds" parameter gives the logging interval. The default is every six seconds.

4 Notes

When you have a project running with ipProfile included, run profilerd on the project, and run **ubicom32-prof** on the target PC, giving it the ip address of teh board and names of the elf files being sampled (ultra, vmlinux, and any applications and libraries). Specify paths to unstripped executables so the Profiler can get the symbols. Typically apps and libraries are built unstripped, but they are stripped when they are copied to **romfs**. You may need to **make menuconfig** or modify a Makefile to build an unstripped binary.

5 Page Tabs

The Profiler tool analyzes and displays data received from the ipProfile package, organized into a set of pages. Click on a tab to select a page.

5.1 Summary Page

This page displays a summary of the performance of the system, along with any errors or warnings. The ELF files and any load addresses are shown on the first line, followed by the date and time and duration of the profiler run, and the specs of the chip under test. It then shows how many packets and samples were received. An accurate profile requires perhaps 100,000 samples or more.

CPU utilization is the fraction of time any thread has work to do. The number of I cache and D cache misses and utilization indicate how busy the memory subsystem is. An IP7K system is heavily loaded if the total misses is over 4.5 million per second. The multithreading rate gives an indication of how well multithreading is hiding cache misses and instruction hazards. Ideally it should be over 2.0. Higher multithreading rates reduce hazards significantly, but will increase the number of cache misses. If the memory system is close to capacity, increasing the multithreading rate will not help performance, and may hurt it.

"Time spent" show how time breaks down at the highest level. Each clock can be either executing one instruction, wasted due to an instruciton that was started but later cancelled (for example a jump that was mispredicted), an idle CPU because it is waiting for the I or D cache, or an idle CPU because there is no work to do.

Finally, the top few functions are shown, with the percentage of total time used by each function (including cancelled instructions and time spent waiting for caches).

5.2 Threads Page

This page displays performance information per thread. Most data is averaged over all samples received since the run started. To clear the results and start new averages, select Actions, Restart, from the menu.

- Thread type. "ULTRA" and "LINUX" are operating system threads. One Linux thread is the software interrupt manager, and the other Linux threads are virtual processors that can execute user applications. HRT is Hard Real Time, typically an Ultra software I/O. GDB is the thread used by the Debugger and Profiler.
- Peak execution rate. HRT threads have a peak rate specified by the HRT table.
- Active. Percentage of time each thread is not suspended and is trying to execute code. Several threads can be active simultaneously, so the sum of the active times can be greater than 100%. The total active time displayed is the percent of time that any thread is active.
- Schedulable. Percentage of time the thread is not suspended and not blocked waiting for a cache.
- Executed. Percentage of time instructions are completed.

- Percentage of time each thread is executing code in DDR (as opposed to on-chip memory).
- I blocked. Percentage of time this thread is waiting for the instruction cache.
- D blocked. Percentage of time this thread is waiting for the data cache.
- Multithread. For non-realtime (NRT) threads, the average number of schedulable threads when this thread is schedulable. Higher numbers reduce the performance lost due to hazards, but also reduce the peak MIPS available to this thread.
- Useful MIPS. Measured number of useful instructions executed per second (MIPS).
- Recent MIPS. The MIPS measured during the last one second interval.
- The MIPS used by the Profiler. That includes instructions in the profiler thread, and instructions used by ipOS to send the UDP packets.
- Instruction cache blocked MIPS. The time wasted by this thread when every active thread is blocked by the instruction cache.
- Data cache blocked MIPS. The time wasted by this thread when every active thread is blocked by the data cache.
- Hazard MIPS. The measured time wasted due to hazards. A hazard is a pipeline event (such as a mispredicted jump), that causes instructions to be cancelled and rescheduled. This time is accurately measured, per thread.
- Hazard estimated MIPS. The estimated time wasted in hazards. The Profiler estimates wasted clocks whenever it takes a sample. These estimates are used to report hazard time lost per function, and per instruction. This estimate usually undercounts the actual hazards, partly because the sampling technique cannot count instructions cancelled due to certain kinds of cache interactions.
- Total time. The total time used by this thread (M clocks per second). This is the sum of MIPS, blocked, and hazard rates.
- CPU utilization. The percentage of total time used by the application.
- The number of simultaneous active, unblocked threads. It shows the fraction of time for each count of some number of simultaneous threads. For best performance, the CPU needs 4 or more threads simultaneously active, to hide most instruction hazards. If there is only one thread active, the total available MIPS will be only 1/3 to 1/2 of the peak.
- NRT multithreading rate. For NRT threads, the average number of active, unblocked threads. Higher values hide more hazards, and give higher total throughput.
- The external DRAM clock rate. Higher frequency reduces the cache miss penalty as well as the I and D blocked MIPS.
- Instruction and data cache statistics. Off-chip code has one I cache access per instruction. Off-chip data has between 0.5 and 1.0 D cache access per instruction, so these statistics indicate how effective the project is using on-chip memory. The cache miss rates indicate how well the project localizes its code and data. Typical miss rates are under 1% for the I cache, and 2% to 4% for the IP71xx D cache. The number of misses gives the load on the caches. If the cache load is too high, cache misses will take longer, and the amount of time blocked waiting for the cache will go up. In the IP51XX, the peak cache capacity

is about 4.5 M miss/sec (170 MByte/sec). Block copy to memory, using the MOVEA instruction, can use 9.7 M miss/sec (310 MByte/sec). The Profiler reports the measured cache miss latency, the number of clocks from the cache request until data is available. It reports the average number of clocks per cache miss that all threads are blocked. The difference between the cache latency and the time all treads are blocked is the benefit of multithreading in hiding cache miss peanlties.

- Mqueue function timing. The Profiler reports the worst case and average latency for any Ultra thread to execute an mqueue message. Every message is measured, so this is an actual worst case, not based on samples.

5.3 Functions Page

This page displays information about each function executed, sorted by the amount of time spent in each function. It is similar to `gprof` profiler output. Each line gives information on a single function, including its location (OCM or DDR), fraction of total time spent in this function, time used by this function (total and by thread number), executed instructions (MIPS), hazard, and blocked time, and memory used by this function (DDR and OCM). Linux functions are prefixed with the application name. Columns give a running total of the percentage of NRT time, and total memory usage in DDR and OCM. The latency column gives the highest sampled time for this function (in microseconds) since the message was dispatched. It gives the number of times the Profiler was able to determine the caller of this function, and, after the function name, it gives the three most frequent callers' count and location. HRT thread functions are included, but the Profiler cannot determine callers for HRT threads, since they are sampled without being stopped first.

There is also a list of functions that are linked to on-chip memory but were never executed.

5.4 Hazards Page

Hazards are displayed, showing the estimated MIPS consumed in NRT threads by each type of hazard. For An/Dn hazards, it displays the distance between the instruction modifying the register, and the use of that register. Hazards in HRT threads are not included. Hazards are:

- JMPT: Unconditional taken JMP.
- JMPcc: Conditional JMP.
- CALL, CALLI, RET: Call and return from subroutine.
- Address: Address arithmetic or a CALLI uses an An or Dn register that was modified too recently.
- MAC: A MAC, MUL, MULF result is used too soon.
- I miss or D miss: A cache miss caused instructions fetched after it to be cancelled.

5.5 Hazard Details Page

This page displays a sorted list of hazards encountered, with the type, location, and MIPS used by each. The same data is available in the **hazards.txt** file.

5.6 Locks Page

This page displays time spent in the Linux kernel spin-waiting for locks, by type of lock, and for the most expensive locks.

5.7 Loops Page

This page displays time spent in the small loops, with the average trips (number of times the loop was executed when it was entered). The "trips" column will show "???" until the Profiler has collected enough data for that loop.

5.8 Applications Page

This page shows all of the loaded and known code memory segments for the operating system and applications. If the **-ip** option was used to specify the board's IP address, the tool telnets to the board and reads the **/proc/maps** file to get the code segment load addresses. These are displayed at the top of the page, along with total memory used by applications for code, stacks, and data. The bottom of the page has all code segments known to the Profiler. This includes segments loaded from ELF files specified on the command line, and all other segements discovered from the **/proc/maps** file. Segments from ELF files have symbol tables, so statistics are displayed for each function. Other segments have all statistics displayed for the segment, and are labelled "Unknown functions".

5.9 Graphs Page

This page displays a real-time graph of items sent from the project using a profile callback function in the project. See the ipProfile documentation for instructions on how to add your own callbacks. A menu of the items is at the bottom. Click on an item to enable bold display of its graph. Click it again to dim the graph. Click a third time to turn it off. Only items with at least one nonzero value are displayed. Ranges are adjusted automatically, and are displayed next to each menu item. Use the appropiate axis to read the graph, based on the range shown. The graph is not saved to a file, but you can use **alt-PrtScr** to save it to the clipboard, and a paint application to save it as a JPG.

5.10 Statistics Page

This page displays the current and maximum value of all the nonzero statistics sent using the profiler callback function. These values are saved in the output **.txt** file.

5.11 Working Set Page

This page displays performance information about the size of code being executed.

Details of the working set are displayed for code running in OCM. It shows the amount of code called only from initialization (before `start()` returns), and the amount of code that is never executed. Since this is sample based, you will need to run a thorough test to see the minimum values.

5.12 Heap Page

This page displays information about on-chip memory heap usage. To enable this display, use the Config Tool to enable heap profiling in ipProfile, enable global debug, enable ipHeap runtime debugging, and "record block allocation callers" in ipHeap. This page displays each heap block, its size, and who allocated it. You can set the depth of the heap caller function traceback in ipHeap. The heap output is arranged by block size and caller. Netbufs are reused rather than being returned to the heap, so the allocator of a netbuf says nothing about the current user.

5.13 Latency Page

This page displays call chains for long latency Ultra messages in the selected thread. It displays stack backtraces for samples that have the longest latency. The default display gives a filtered backtrace, showing only valid addresses in the stack. Select menu "View -> All Latency Data" to see the entire stack beacktrace, which will include other called functions that might not be in the same call chain.

5.14 Instructions Page

This page displays performance information about instructions executed.

- The number of memory operands per instruction, type of memory operands, and fraction that access the stack.
- Branch prediction rates and branch direction. Number of unique jump instructions executed, and histogram of fraction of jumps with different ranges of times taken. Jumps always taken or never taken are easily predicted using the static prediction bit, which can be set properly using the Profiler's "Action" menu. There is a histogram showing the fraction of `jmp` instructions in each 10% range of taken frequency.
- Instruction frequencies for each kind of instruction executed. Number of instructions and unique instructions sampled. Note that unconditional JMP (JMPT) is shown separately from conditional JMP. CALLI is often used to return from a function.

5.15 HRT Threads Page

This page gives the number of samples at each instruction in each HRT thread. HRT threads are not stopped before sampling, so speculative instructions (for example after a taken `jmp`) are included.

6 Menus

6.1 File Menu

Use menu "File -> Print" to print the contents of the display, and "File -> Save As" to save the results. "File -> Save" puts the display pages in a text file with the same base name as the ELF file. It also saves `gmon.out`, `gmonflash.out`, `coverage.txt`, and `hazards.txt`.

6.1.1 Gprof

When the Profiler exits, it saves files called `gmon.out` and `gmonflash.elf`. Run `gprof elf-file.elf`, giving it the ELF file name, to see profile output for your application, giving time used by each on-chip function, along with call graph information. The flat profile is based on the sampled data. For call graphs, traditional `gprof` includes code to capture every call. `ubicom32-prof` reconstructs this data from the sample values, so the call counts in `gprof` are statistical, not absolute. Run `gprof` specifying `gmonflash.out` to get a profile of code that is running in flash memory.

6.1.2 Hazards

`Hazards.txt` shows the instructions that cause the greatest loss of performance due to hazards. Each line gives the MIPS lost, the average number of lost clocks each time that instruction is executed, the type of hazard, and the instruction location. You can use `ip3k-elf-objdump -d -S` to get a mixed disassembly listing to find the source code for that instruction, or use addr2line. When multiple threads are active the penalty for a hazard is reduced or eliminated. The maximum penalties shown below happen when there is only one active thread.

Hazards include:

- JMPT: A jmpt instruction was taken, up to 3 clock penalty.
- JMPcc: A conditional JMP. Up to 7 clock penalty.
- Call: Each call has up to 3 clock penalty.
- CallI: Indirect call, used to return from subroutine. Up to 4 clock penalty.
- RET: return from subroutine. Up to 7 clock penalty.
- An or Dn Reg: An Address register or Data register is loaded, then used in an address calculation within 4 clocks. Up to 5 clock penalty.
- MAC: The MAC_HI or MAC_LO register are read within 3 clocks of being set. Up to 6 clock penalty.
- An CLI: An modified and CALLI uses it within 4 clocks. UP to 5 clock penalty.

6.2 View menu

Select "Small Font" to get more information on the screen.

"All Latency Data" shows all received data in the latency page, without filtering for valid function addresses.

"Graph Duration" selects the time scale on the X axis for the graphs page, from 1 minute to 60 minutes.

"Sort Functions by" lets you sort the functions page by time used, or by latency.

6.3 Actions menu

"Restart" reads the ELF file, clears the accumulated data, and gets the `/proc/maps` file from the board.

"Fix Mispredicts" changes the branch prediction bits on mispredicted JMP instructions to improve performance. For best results, let it take data for a million instructions or more. After fixing the ELF file, make sure that you download that file and that your Makefile does not recreate it.

"Collect Data for" lets you specify a single thread or all threads to collect data for.

"Log Data" causes the tool to create a file `profile_log.txt`, and every six seconds add a line of data, giving the performance during the previous six seconds.

"Collect Data" stops ot starts the Profiler from accumulating new data. Use it to freeze the display.

6.3.1 Log File Format

When the `-log` command line option is used, the Profiler logs data at regular intervals to a comma-separated file called `profile_log.txt`, that can be imported into a spreadsheet. Each row contains one sample. Sample data includes the time, MIPS rate for each thread, flash code execution percentage, idle percentage, heap minimum and average during the last interval, and netpage minimum and average during the last interval.

7 Possible ubicom32-prof Problems

`ubicom32-prof` updates its display once per second. If no data appears, the cause is most likely a bad IP address. If the IP address is correct, the problem is usually that some firewall software is not allowing the UDP packets to enter your PC.

When you recompile and download a new ELF file, you must restart the Profiler or select "New" or "Open" on the "File" menu, or select "Restart" in the "Actions" menu, so that the Profiler will read the new ELF file. Otherwise the reports will have incorrect function names.

Application Note

Ubicom®, Inc.

Ubicom develops networking and media processor solutions that address the unique demands of real-time interactive applications and multimedia content delivery in the digital home. The company provides optimized system-level solutions for a wide range of products including wireless routers, access points, bridges, VoIP gateways, networked digital photo frames, and streaming media players.

195 Baypointe Parkway
San Jose, CA 94134

Tel 408.433.3300
Fax 408.433.3339

Email sales@ubicom.com
Web www.ubicom.com