XAPP545 (v1.0) September 15, 2004

Statistical Profiler for Embedded IBM PowerPC

Author: Njuguna Njoroge

Summary

This application note describes how to generate statistical profiling information from the IBM PowerPC 405D, which is embedded in some Virtex-II Pro™ FPGAs. Specifically, the application note details how to convert trace output files generated from the Agilent Technologies Trace Port Analyzer into a gprof (GNU profiler) readable format. The gprof tool is capable of generating a histogram of a program's functions and a call-graph table of those functions.

Introduction

Profiling allows a programmer to assess where a program is spending most of its time and how frequently functions are being called. With this information, a programmer can gain insight as to what optimizations to make. In some cases, a programmer can identify bugs that would be otherwise undetectable. Using the RISCWatch application is a first step in profiling a program. Nonetheless, manually mulling through approximately 350 MB (or 8 million cycles) of trace data to discern the run-time profile of the program is a fruitless endeavor. In fact, the designers of RISCWatch realized this, so they standardized the trace output format to enable post processing of the trace. The regular format of the trace files allows scripts to parse through the trace and convert it to the gprof format. By converting the trace files into the gprof format, the widespread use of gprof in the profiling community is leveraged.

Prerequisite Knowledge, Tools, and Equipment Required

Hardware

- ML300 board with Virtex-II Pro FPGA
- Agilent Technologies E5904B Option 60 Trace Port Analyzer

Software

- EDK - tested with version 6.22 and 6.3
- IBM RISCWatch Debugger - tested with versions 5.0 and 5.1
- Latest/recent version of gprof (from GNU binutils) - tested with version 2.14 and 2.15 or powerpc-eabi-gprof that is shipped with EDK version 6.3

Prerequisite Knowledge

- EDK tool flow
- IBM RISCWatch Debugger
- GNU gprof tool

Contents of the xapp545.zip File

- `convert2gmonout.pl` - Converts trace file to gmon.out format
- `simple_test.c` - Simple Test Program
- `simple_test.elf` - PPC405D binary of above program

- `simple_test.trc` - Trace capture running on ML300. The trace capture tool was set to collect ~150 cycles of instructions after the trigger point and up to 4 million before the trigger point. Since the program is short, it captured approximately 164,000 cycles before the trigger point.
- `simple_test_le.out` - gmon.out file (for Little Endian machines)
- `simple_test_be.out` - gmon.out file (for Big Endian machines)
- `simple_test.pro` - gprof generated profile using above files
- `sample_linkerscript` - linker script modified to work with RISCWatch

Figure 1 provides an overview of the steps described in this application note.

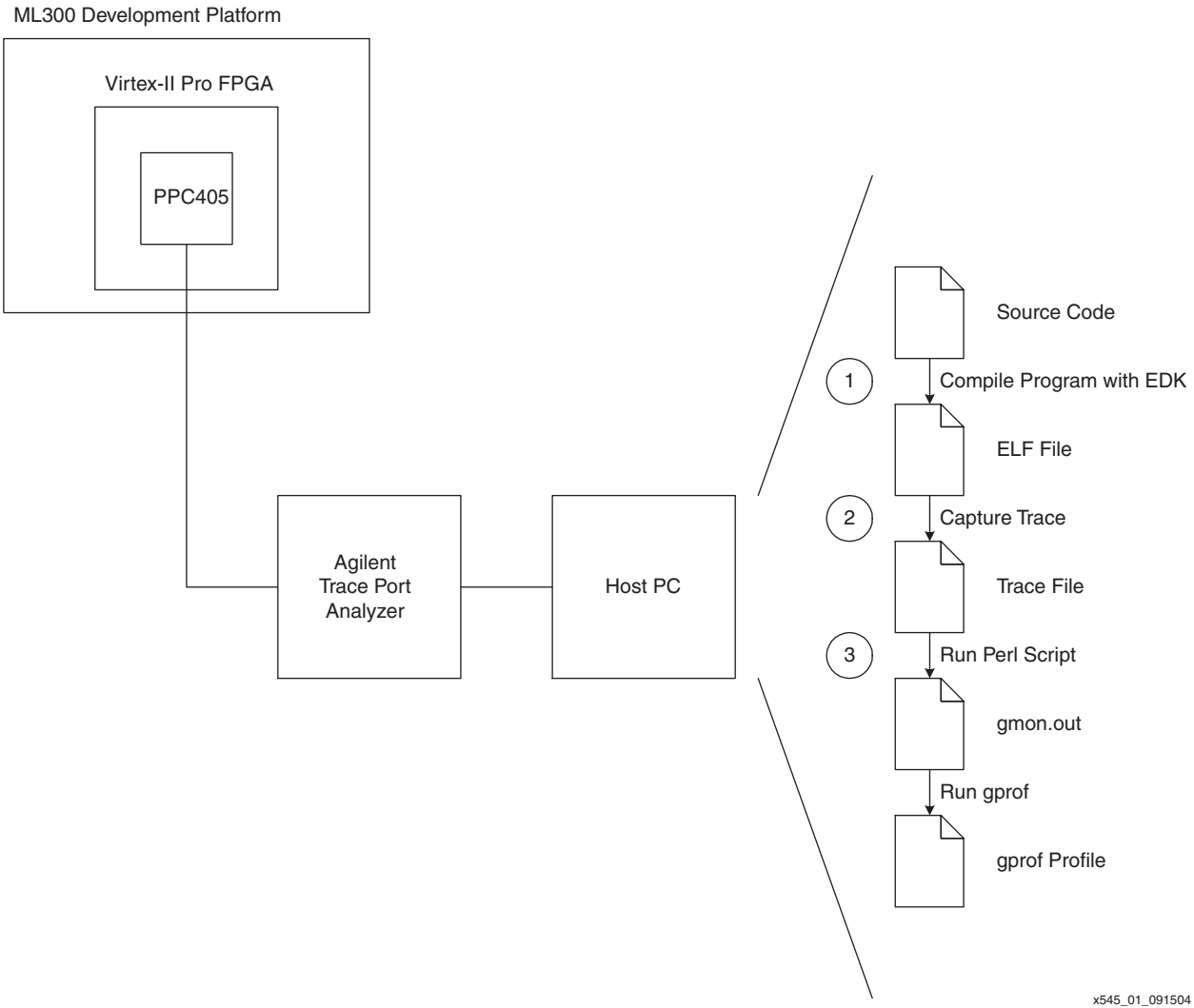

Figure 1: **Summary of Steps**

Generating a Profile

1. Compile your program using the EDK tool flow. Before compilation, the linker script must be specified to use the SDRAM for memory allocation because RISCWatch is not able to read/write the On-Chip Memory (OCM) or block RAM's. See the `sample_linkerscript` file for an example of how to make these changes. Additionally, the **`-g`** option in gcc must be used to provide the debug information for RISCWatch.

   Example Program (from `simple_test.c` in the xapp545.zip file):

```
void sub_routine1(void)
{
  int i;
  for(i = 0; i < 5; i++)
    {
     sub_routine2();
   }
  return;
}

void sub_routine2(void)
{
  int i;
  for(i = 0; i < 25; i++) {}
  return;
}


int main()
{
   int i, sum;

   for(i = 0; i < 100; i++)
     {
       sub_routine1();
     }
   for(i = 0; i < 100; i++)
    {
      sub_routine2();
    }
   exit(0);
      }
```

2. Generate a trace (`.trc`) output file. Refer to the *RISCWatch User Manual* for instructions on how to connect/configure the hardware and download the binary through the RISCWatch application.

   Snippet of Trace output for `simple_test.elf`:

```
# Instr      Total     Cycle/          (function
# Count      Cycle     Instr   Address  offsets) Opcode   Disassembly
# --------  --------  ------- ----------------- -------- -----------
  00000001  00000000           00010500           48000004 b          $+0x00000004
  00000002  00000003  0000003  00010504           3C000000 addis      R0,0,0x0000
  00000003  00000004  0000001  00010508           60000080 ori         R0,R0,0x0080
  00000004  00000005  0000001  0001050C           7C0803A6 mtlr       R0
  00000005  00000051  0000046  00010510           4E800020 blr
                                          .
                                          .
                                          .

  00000020  00000489  0000019  000000F0           7C1C43A6 mtspr      TBL_W,R0
  00000021  00000490  0000001 000000F4           7C1D43A6 mtspr      TBU_W,R0
  00000022  00000492  0000002 000000F8           4BFFFF4D bl         $+0xFFFFFF4C


$ FUNCTION: main  START_ADDR: 0x00000044  FILE: ?  PROGRAM: C:\Program
Files\RISCWatch\simple_test.elf
  00000023  00000543  0000051 00000044(+000000) 9421FFF0 stwu        R1,0xFFFFFFF0(R1)
  00000024  00000546  0000003 00000048(+000004) 7C0802A6 mflr        R0
  00000025  00000547  0000001 0000004C(+000008) 93E1000C stw         R31,0x0000000C(R1)
  00000026  00000615  0000068 00000050(+00000C) 90010014 stw         R0,0x00000014(R1)
  00000027  00000616  0000001 00000054(+000010) 480000B1 bl          $+0x000000B0

$ FUNCTION: ?  START_ADDR: ?  FILE: ?  PROGRAM: C:\Program Files\RISCWatch\simple_test.elf
  00000028  00000687 0000071 00000104          3D600000 addis       R11,0,0x0000
  00000029  00000690 0000003 00000108          616B04D4 ori         R11,R11,0x04D4
  00000030  00000691 0000001 0000010C          81AB0000 lwz         R13,0x00000000(R11)
(The above function is unknown, "?", by RISCWatch. Fortunately, gprof is smarter at linking the
function names with the addresses, so it is able to resolve the above function name)
```

3. Use the Perl script to convert the trace file. Run it using the following commands:

```
perl convert2gmonout.pl tracefile.trc gmon.out
gprof program.elf gmon.out
```

Results in an Output Similar to the Snippet of gprof Output for `simple_test.elf`:

```
Flat profile:

Each sample counts as 1 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 45.33  74540.46 74540.46      600   124.23   124.23  sub_routine2
 39.61 139664.86 65124.40      100   651.24  1272.41  sub_routine1
 12.56 160317.09 20652.24        1 20652.24 161296.15  main
  0.42 161013.64   696.54        1   696.54   846.05  exit
  0.32 161543.17   529.53                             _start
  0.09 161692.68   149.51        1   149.51   149.51  _exit
  0.08 161825.68   133.01        1   133.01   133.01  __eabi
                              .
                              .
                              .
Call graph (explanation follows)
granularity: each sample hit covers 2 byte(s) for 0.00% of 161825.68
seconds


index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0  529.53 161296.15                 _start [1]
             20652.24 140643.92       1/1           main [2]
-----------------------------------------------
             20652.24 140643.92       1/1           _start [1]
[2]     99.7 20652.24 140643.92       1         main [2]
             65124.40 62117.05     100/100         sub_routine1 [3]
             12423.41    0.00     100/600         sub_routine2 [4]
              696.54  149.51       1/1           exit [5]
              133.01    0.00       1/1           __eabi [7]
-----------------------------------------------
```

Using the histogram data from gprof for `simple_test.c`, a pie chart can be created for presentations like the one shown in Figure 2:

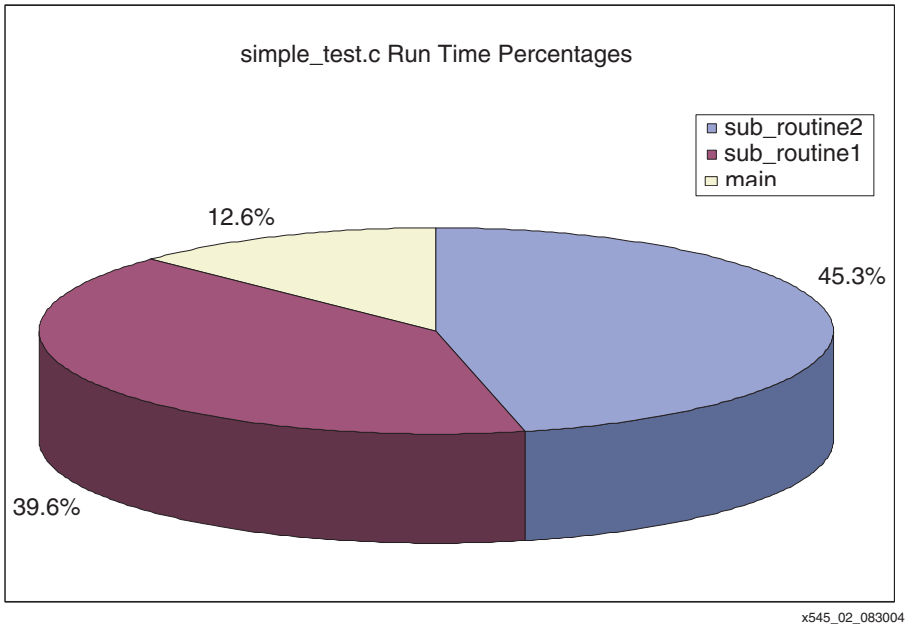

Figure 2: **simple_test.c Run-Time Percentages**

Notes Regarding Steps

- The `gmon.out` file does not have to be named "gmon.out". Choose whatever name is desired. Also, gprof can be run without `gmon.out` specified (that is, **`gprof program.elf`** is a valid command). However, by default, gprof searches in the current directory for a file named `gmon.out`. Thus, if another name is used, it must be included explicitly, as demonstrated above.
- Endianness matters: If you generate a `gmon.out` file on a Little Endian machine, then run gprof from a Little Endian machine, and vice versa for Big Endian.
- The xapp545.zip file contains the `simple_test.c` program, which can be used to replicate the above steps. The `simple_test.c` is a basic program that calls two subroutines from the main() function. It was used to verify that the `convert2gmonout.pl` script was working properly.

Known Issues

- gprof - gprof has problems reading PowerPC binaries. It adds functions to the profile that were never called during the execution of the program. Fortunately, gprof reports that these functions take typically less than 1% of the total runtime, so they do not influence the data collection. Running gprof on a PowerPC removes some of these "phantom" functions. Apparently, native powerpc gprof is better at reading PowerPC binaries. A bug note was sent to GNU regarding this issue.
- Profiling Linux - RISCWatch does not provide any mechanism to deal with the virtual addresses used by Linux. While the translation scheme for the Linux Kernel is linear, the Interrupt Handler uses physical addresses, which confuse RISCWatch (it passes these physical addresses to the TLB). Consequently, the trace capture is corrupted and not usable. For now, restrict profiling to stand-alone applications.
- ML310 Virtex-II Pro FPGA: The Agilent Trace Port Analyzer cannot properly communicate with the PowerPC 405 processor, because the PowerPC processors in Virtex-II Pro XC2VP20 or XC2VP30 devices have a slightly different version number, which is not recognized by the Trace Port Analyzer. Xilinx is working with Agilent and IBM to create a patch for this issue.

Conclusion

Profiling has become an invaluable tool when debugging and optimizing applications. This application note described the use of a Perl script that leverages the capabilities of two well-developed tools, RISCWatch and gprof, to easily profile an application.

References

- Agilent Technologies E5904B Option 60 Trace Port Analyzer User Manual
- IBM RISCWatch Debugger User Manual, version 5.1
- GNU Profiler, grof

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/15/04	1.0	Initial Xilinx release.