Profiling C/C++ Using Eclipse

Jeff Johnston (jjohnstn@redhat.com)
Principal Engineer, Red Hat
June 12, 2013
Packages

- devtoolset-2-ide (Eclipse components)
  - scl-utils, devtoolset-2-eclipse-* (cdt, platform, linuxtools)
- devtoolset-2-perftools (Performance Tools)
  - scl-utils, valgrind, oprofile, systemtap
- devtoolset-2-toolchain (Tool chain)
  - scl-utils, gcc, binutils, gdb, elfutils
- devtoolset-2
  - Includes all 3 packages above
CDT

- devtoolset-2-eclipse-cdt package
- C/C++ Developer Tools
  - Project editing, building, debugging, indexing
  - Similar to JDT (Java Developer Tools) only for C/C++
- Based on upstream Kepler contribution
- CDT 8.2.0
  - Previous version is 7.0.0
Linux Tools

Eclipse plug-ins for supporting:

- Perf - devtoolset-2-eclipse-perf
- Oprofile - devtoolset-2-eclipse-oprofile
- Valgrind – devtoolset-2-eclipse-valgrind
- Gcov - devtoolset-2-eclipse-gcov
- Gprof - devtoolset-2-eclipse-gprof
- SystemTap - devtoolset-2-eclipse-systemtap
Profiling Framework

- New in Eclipse Linux Tools for DTS 2.0
- Novice users can profile via categories
  - Timing, Code Coverage, Memory
  - Each category has default tool
  - Default tools can be set in C/C++ Preferences or overridden in Project Properties->C/C++ General
  - Categories accessed via new “Profiling Tools” context menu
    - Profile Memory, Profile Timing, Profile Code Coverage
Profiling Framework continued...

- New Profile As...-> Local C/C++ Application
  - Shares settings with Debug / Run launch configurations (e.g. Debug as...-> Local C/C++ Application)
    - Program arguments, environment variables
  - Profiling launch configuration dialog adds “Profiler” tab
    - Allows user to select which profiling tool to use and settings for that tool (any of the supported profiling tools can be specified)
    - Multiple launch configurations are possible
    - Default is to use Perf tool to record cpu-cycles
Profiling Framework continued....2

- New “Profiling Tools” context menu lists profiling categories as well as individual tools
  - Previously, these tools were shown under “Profile as..”
- “Profiling Tools Configurations” dialog allows user to create/edit launch configurations for specific tools or categories (e.g. Profile with Valgrind)
  - Individual profiling tool settings are presented in dialog as appropriate
Profiling Tools Menu

Profile As
Debug As
Run As

Profiling Tools
Clean Selected File(s)
Build Selected File(s)
Compare With
Replace With
Run C/C++ Code Analysis
Team

Properties

1 Profile Code Coverage
2 Profile Memory
3 Profile Timing
4 Function callgraph
5 Profile With OProfile
6 Profile With OProfile (Manual)
7 Profile with Perf
8 Profile With Valgrind

Alt+Enter Profiling Tools Configurations...
Gcov

- Profile code coverage (new for DTS 2.0)
- Only tool in “Profile Code Coverage” category
- Requires special hooks compiled in by gcc
  - -fprofile-arcs and -ftest-coverage
    - For Autotools projects, there is a checkbox under Project->Properties->Autotools->Configure Settings->Advanced
    - For Managed make projects, one can use Project->Properties->C/C++ Build->Settings
- Recommended that optimization be turned off as inlining fools coverage tool
Gcov continued...

- Each compiled file generates .gcda file during run
- Gcov viewer displays results in list with coverage percentages
  - double-clicking list item opens colored/annotated editor
    - Green means coverage, red means no coverage
- Two ways to bring up Gcov viewer
  - Double-click any .gcda/.gcno file after running
  - Profile the application using Gcov
    - e.g. Use “Profile Code Coverage” on binary
Gcov View

program runs = 0
program file: /home/jjohnstn/workspace/hello/src/hello.c

<table>
<thead>
<tr>
<th>Name</th>
<th>Total Lines</th>
<th>Instrumented</th>
<th>Executed Line</th>
<th>Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>25</td>
<td>8</td>
<td>7</td>
<td>87.5%</td>
</tr>
<tr>
<td>hello.c</td>
<td>25</td>
<td>8</td>
<td>7</td>
<td>87.5%</td>
</tr>
<tr>
<td>main</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>100.0%</td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>3</td>
<td></td>
<td>75.0%</td>
</tr>
</tbody>
</table>
Gcov Editor Colorization
Gprof

- Generate flat timing profile
  - call graph profile not supported in Eclipse
  - New for DTS 2.0
- Requires special hooks compiled/linked in
  - -pg compile/link option
  - For Autotools projects, there is a check-box under Project->Properties->Autotools->Configure Settings->Advanced
  - For Managed make projects, there is a check-box under Project->Properties->C/C++ Build->Settings->GCC Compiler->Debugging (-pg)
Gprof continued...

- Creates gmon.out in CWD when program is run
- Gprof viewer displays results in tree list with timing percentages
  - Clicking on leaf list items brings up C/C++ editor
- Two ways to bring up Gcov viewer
  - Double-click on gmon.out file after running application
  - Profile the application using Gprof
    - e.g. Use “Profile Code Timing” and reset tool to be gprof instead of perf
Gprof View

C/C++ - factorial/perf.stat - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Console gprof

Gmon file: /home/jjohnstn/workspace/factorial/gmon.out
Program file: /home/jjohnstn/workspace/factorial/src/a.out
4 bytes per bucket, each sample counts as 10.000ms

<table>
<thead>
<tr>
<th>Name (location)</th>
<th>Samples</th>
<th>Calls</th>
<th>Time/Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>factorial.c</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>factorial1</td>
<td>4</td>
<td>20000000</td>
<td>2ns</td>
</tr>
<tr>
<td>factorial1 (factorial.c:28)</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>factorial1 (factorial.c:32)</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>factorial2</td>
<td>10</td>
<td>1000000</td>
<td>99ns</td>
</tr>
<tr>
<td>factorial2 (factorial.c:1)</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>factorial2 (factorial.c:35)</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>main</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Valgrind

- Uses simulation (can be slow)
- Multiple sub-tools
  - memcheck (check for memory leaks)
    - Default for “Profile Memory” category
  - massif (display memory usage over time in chart)
  - helgrind (check for memory usage in threads)
    - New for DTS 2.0
  - cacheegrind (check for cache usage)
    - Better to use Perf instead
Valgrind continued..

- Errors found by valgrind result in markers added to the C/C++ editor
  - Markers are cleared by selecting to clear all markers in the Valgrind view
  - Markers are not cleared by new build
Valgrind View (memcheck)
Valgrind Massif Chart
Valgrind View (massif)
Valgrind Error Markers

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    void *x = malloc(5000);

    // 5,000 bytes in 1 blocks are definitely lost in loss record 1 of 1 [PID: 32100]
}
```
SystemTap

- Eclipse provides SystemTap script editor
  - Hover/completion support with probe documentation extracted from man pages (new for DTS 2.0)
  - Right-click on script to compile/run or to set up run configuration with SystemTap options (options new in DTS 2.0)
  - Can run script tied to running an Eclipse binary or user can manually start/stop script and run anything in-between (new in DTS 2.0)
SystemTap continued...

- “Run launch” configuration for SystemTap allows setting up graphing specifications (new in DTS 2.0)
  - User can specify regex filters for output data
  - Graph type can also be selected
    - Pie chart, standard graph, bar graph
Perf

- New for DTS 2.0 – default “Profile Timing” tool
- Similar events to Oprofile but no root access required
- Sub-commands supported:
  - `stat` – event counts for application
    - Can select which events to stat or use default set
    - Can diff previous stat run or saved runs
    - Can select multiple runs to get averaging and variance
  - `record/report/annotate` – default usage
    - Default event is cycles but is configurable
Perf continued...

- Perf view displayed
  - For “stat” the events and numbers are displayed
  - For “record”, tree lists are shown for each event
- Disassembly view supported for record (not stat)
  - Disassembly generation is option in profiling configuration settings
  - Can search in disassembly view using Ctrl+f
- Can invoke directly using “Profile with Perf” in Profiling Tools menu
Perf View

C/C++ - factorial/src/factorial.c - Eclipse Platform

Perf Profile View

/home/jjohnstn/workspace/factorial/src/a.out (5/22/13, 5:08 PM)

- 100.0% (148 samples) in a.out
- 96.23% (132 samples) in a.out (at /home/jjohnstn/workspace/factorial/src/a.out)
- 96.23% in /home/jjohnstn/workspace/factorial/src/factorial.c
- 96.23% (132 samples) in [.] main
  - 39.39% (51 samples) on line 38
  - 25.0% (33 samples) on line 39
  - 23.49% (31 samples) on line 32
  - 12.12% (15 samples) on line 29
- 2.18% (14 samples) in [k] [kernel.kallsyms]
- 1.39% (1 samples) in ld-2.12.so (at /lib64/ld-2.12.so)
Perf Stat

<table>
<thead>
<tr>
<th>Performance Counter Stats for /home/jjohnstn/workspace-summit/autohello/src/a.out (3 runs):</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.490979 task-clock # 0.633 CPUs utilized ( + 5.20% )</td>
</tr>
<tr>
<td>0 context-switches # 0.679 K/sec ( + 100.00% )</td>
</tr>
<tr>
<td>0 CPU-migrations # 0.679 K/sec ( + 100.00% )</td>
</tr>
<tr>
<td>119 page-faults # 0.242 M/sec ( + 0.28% )</td>
</tr>
<tr>
<td>1,296,980 cycles # 2.642 GHz ( + 5.22% )</td>
</tr>
<tr>
<td>936,389 stalled-cycles-frontend # 72.20% frontend cycles idle ( + 7.08% )</td>
</tr>
<tr>
<td>344,628 stalled-cycles-backend # 26.57% backend cycles idle ( + 32.62% )</td>
</tr>
<tr>
<td>716,719 instructions # 0.55 insns per cycle</td>
</tr>
<tr>
<td>126,935 branches # 1.31 stalled cycles per insn ( + 0.24% )</td>
</tr>
<tr>
<td>4,651 branch-misses # 258.534 M/sec ( + 0.20% )</td>
</tr>
<tr>
<td>0.000775905 seconds time elapsed ( + 7.96% )</td>
</tr>
</tbody>
</table>
Perf Disassembly View

```
/home/jjohnstn/workspace/factorial/perf.data (5/22/13, 5:44 PM)

0.00 : 400432: jmp 40043b <main+0x1b>
0.00 : 400434: nopl 0x0(%rax)
23.85 : 400438: mov %rdx,%rax
0.00 : 40043b: lea -0x1(%rax),%rdx
0.77 : 40043f: imul %rax,%rsi
     :    printf("x is %lld\n", x);
     :    return 0;
     : }
     :
     : unsigned long long factorial1(unsigned long long n) {
     : if (n <= 1)
21.54 : 400443: cmp $0x1,%rdx
0.00 : 400447: jne 400438 <main+0x18>
```

#redhat #rhsummit
Oprofile

- Requires root access to use
  - Also need to echo 0> /proc/sys/kernel/nmi_watchdog
- Similar events to perf (default: CPU_CLK_UNHALTED)
- Oprofile view shows current session
  - Session has tree list of events which open to show percentages
  - double-clicking on leaf nodes can bring up C/C++ editor
  - Sessions can be saved, otherwise current session is overwritten
Oprofile View

CPU_CLK_UNHALTED

- 100.00% in /home/jjohnstn/workspace/factorial/src/a.out
- f() 100.00% in main [factorial.c]
  - 31.34% on line 38
  - 25.90% on line 29
  - 25.68% on line 32
  - 17.08% on line 39
Further Information

- All profiling tools and CDT have user guides found in Eclipse Help menu->Help Contents
- Linux Tools project wiki pages can be found at:
  - [http://wiki.eclipse.org/Linux_Tools_Project](http://wiki.eclipse.org/Linux_Tools_Project)
- CDT project wiki pages can be found at:
Demo
Please fill in the survey form for this presentation found in the back of the room