Profiling C/C++ Using Eclipse

Jeff Johnston / Roland Grunberg

Red Hat
March 25 / 2013

v0.1

Setup

- Boot from USB (enter boot selection mode)
  - Usually f12 on PC and Options key on Mac
- Launch Eclipse from Applications
  - Default workspace (/home/liveuser/workspace)
- Create 2 Local P2 Repositories
  - Help -> Install New Software -> Add (Local)
    - /usr/share/java/eclipsecon-setup/repository-linuxtools
    - /usr/share/java/eclipsecon-setup/repository-eclipsecon
- From the “repository-eclipsecon” install the single feature

Setup Continued ..

- Make sure to restart Eclipse
- File -> Import -> General -> Existing Projects
  - Add the following to root (File System)
  - /usr/share/doc/eclipsecon-setup/examples
  - Enable “copy projects into workspace”
  - Click Finish

Agenda

- Linux Tools Profiling Framework
- Gcov / Gprof
- Valgrind
- Oprofile
- Perf
- Pipelining stalls / Caching
- Systemtap / LTTng
- C/C++ Coding Tips

Introduction

- Profiling/optimization is an art
- Program already works
- How do we know we are done?
- Competitor or user requirement
- Optimizations are usually arch-specific
- How do we not get in compiler's way?
- What if we have multiple architectures?

Installed Tools

- Eclipse – eclipse-platform-4.2.2
- CDT – eclipse-cdt-8.1.2
- Linux Tools – eclipse-linuxtools-2.0.0 from nightly build
- SystemTap – systemtap-2.1.2
  - systemtap-runtime-2.1.2
- Gcc – gcc-4.7.2
- Valgrind – valgrind-3.8.1
- Oprofile – oprofile-0.9.8
- Perf – perf-3.8.2

Linux Tools

- Eclipse project
- provides Eclipse C/C++ plug-ins for
  - Gcov / Gprof
  - OProfile
  - Valgrind
  - Perf
  - SystemTap
  - LTTng

Profiling Framework

- Profiling tools tied to Profiling Framework
- Novice users can profile via categories
  - Timing, Coverage, Memory, etc..
  - Each category has default tool
  - Each tool has default settings
- User can access each tool directly
  - Via Profile as... context menu or Profiling Configurations menu

GCov

- Profile code coverage
- Requires special hooks compiled in by gcc
  - -fprofile-arcs and -ftest-coverage
  - Each compiled file will generate a .gcda file when the application is run
  - Run unoptimized to avoid inlining
- Bring up Gcov viewer
  - Double-click on .gcda files
  - Run application via Gcov profiling

GProf

- Generate flat timing profile or call graph
- Requires special hooks compiled/linked in
  - -pg compile/link option
- Creates gmon.out in CWD when app executed
- To bring up Gprof view
  - Double-click on gmon.out file generated after run
  - Run application via Gprof profiling

Gprof Cont...

- Uses set sampling interval – not very accurate
- Hooks are left in program
- To improve accuracy one can do multiple runs and combine gmon.out files using -s command-line option
- Eclipse Gprof just supports flat profile for single run
- -g option needed for useful reports

Valgrind

- Uses simulation (can be very slow)
- Multiple sub-tools
  - Memcheck (check for memory leaks)
  - Massif (check for memory usage)
  - Helgrind (check for proper memory use in threads)
  - Cachegrind (check for cache usage)

OProfile

- Requires root access
  - Set-up and usage
- Can profile system wide
- Similar events to Perf
- Operf uses perf events and doesn't require root access
- Uses sampling

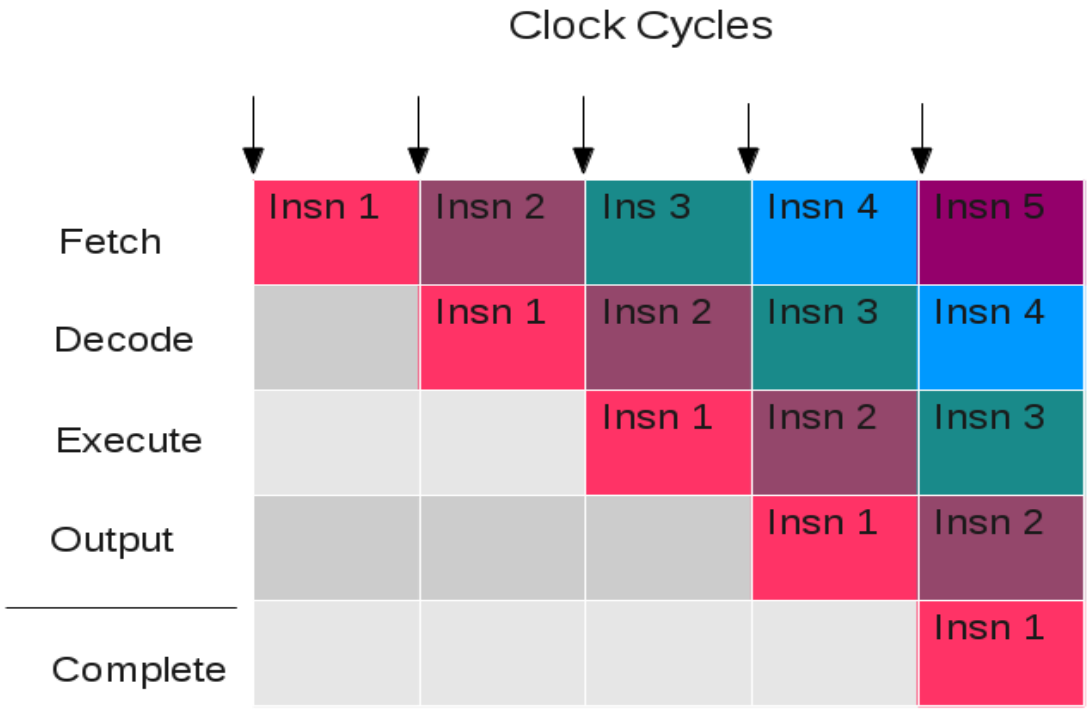
Perf

- Requires Linux kernel 2.6 or greater
- Hardware registers count events
- Multiple tools:
  - Top – not supported
  - Stat – added for Kepler (including averaging)
  - Record/report/annotate – normal usage
- Low overhead, easy to use
- No root access required

Pipelining

- Most modern architectures break up insn execution into multiple phases
  - Typically at minimum: instruction fetch, instruction decode, execution, and output
- Phases can execute in parallel and so while one insn is being fetched, another can be decoded, another executed, etc...
- Maximum through-put occurs when we always keep the pipe full and all phases are doing work

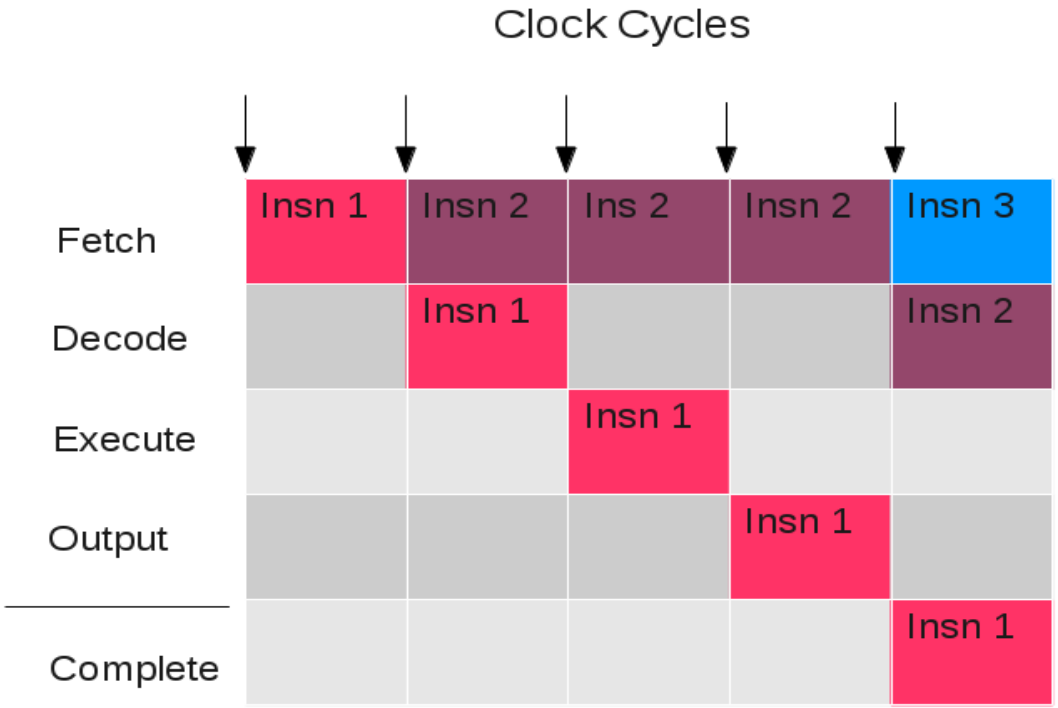
Pipelining Cont...



Pipelining Cont...

- Run into problems when any pipe-line phase stalls
- Branches cause issues because a decision has to be made about what the next insn to load after the branch
- Architectures employ branch prediction to try and predict more accurately the next insn after a branch
- Guess wrong and we have to flush the pipe and start refilling with the proper insn
- Instructions execution can cause stalls
- Cache access can cause stalls

Pipelining Stall



Caching

- Processor has fast cache memory which can be accessed far quicker than main memory
- Multiple levels of caches
- Each level up is bigger cache with higher latency
- Lowest level checked first, then next, etc... until finally main memory is used
- /sys/devices/system/cpu/cpu*/cache/index*/size
- Each cache is divided up into multiple lines

Caching Cont...

- Processor must efficiently keep cache lines filled and evict when necessary or refresh when dirty
- L1 Instruction cache – read-only usually per processor
- L1 Data cache – read/write usually per processor
- Multi-level caches L2 and up are shared for insns/data and often shared among processors/hyper-threads

Instruction Caching

- modern CPUs can execute hundreds of instructions in the latency time taken to fetch a cache line from main memory
- Insn cache read misses are the most costly because processor thread has to wait for insn to be fetched
- Out-of-order CPUs (Pentium Pro and later Intel designs, for example) attempt to execute independent instructions after the instruction that is waiting for the cache miss data
- hyper-threading (HT), allows an alternate thread to use the CPU core while a thread waits for data

Optimizing Insn Cache

- Reduce code footprint as much as possible
- Keep rarely used paths of code in separate functions (e.g. Initialization and corner cases)
- Use built-ins and inlining strategically
  - Functions used only once might as well be inlined *
  - Small functions used frequently should not be inlined
  - -finline-limit can be used to set maximum size of inlined function
  - -Os can be used to see what happens if optimizations affecting size are prohibited (loop unrolling, inlining) – sometimes surprising

Systemtap

- Compiles scripts into kernel modules
- Need kernel-debuginfo, kernel-debuginfo-common-arch and kernel-devel pkgs installed
- Different groups to avoid needing root access
  - (stapusr) minimum needed to use staprun
    - Must use compile server or use scripts in /lib/modules/kernel_version/systemtap/
  - (stapdev) can compile all scripts
  - (stapsys) same as dev but can't use guru mode

Systemtap Cont...

- Scripts are C-like (guru mode allows embedded C)
- Scripts are compiled to C and kernel module installed
- Scripts specify probe points and functions
- Probe points are synchronous and asynchronous events
- Functions are helper routines to probes
- Tapsets provide useful functions / aliases
- Can add manual probe events in code

LTTng

- Linux Trace Toolkit next generation
- Kernel and user-space tracing
- Kernel tracing requires installation of kernel module
- User-tracing requires statements added to application or using gdb tracepoints
- Uses CTF (common tracing format)
- Rich UI for examining traces

C/C++ Coding

- Use of switch statements rather than large if/else
  - Compiler can generate jump table
- Moving conditions out of loops
  - Move condition outside and have multiple loops
  - Condition evaluated once so less branching
- Combining loops
  - Combining loops of same size can help
  - Note that sometimes we up the insn cache misses

C/C++ Coding Cont..

- Unrolling loops
  - Perform inner loop behaviour on multiple elements
  - cut the indexing and branching by factor
- Use of count-down indexing
  - compiler only has to compare to 0
  - frees up a register
- Avoid bit-fields (DIY) – they are inefficient

C/C++ Coding Cont...

- Avoid C++ virtual inheritance if possible (slow)
- Avoid C++ virtual functions if possible (slow)
- Use static qualifier to promote function inlining
- Don't use static qualifier for function-local data
- Use of const qualifier where appropriate
- Use restrict keyword to help compiler (C99)
  - Tells compiler that pointers aren't accessing same data
- Declare vars in inner-most scope possible and postpone until needed (avoid unnecessary ctor/dtor)

C++ Coding

- Pass C++ parms by reference except for base types
- Initialize a variable rather than declare and assign
  - Declare and assign will result in copy performed
- Use ctor initialization lists for setting member objects
  - Avoids additional default constructor being run for object
- Use Operator= instead of just Operator
  - Again we avoid a copy being performed (e.g. += vs +)
- Avoid post-fix operators
  - Need to save old value and for class this is costly

Further Information

- http://wiki.eclipse.org/Linux_Tools_Project
- http://oprofile.sourceforge.net
- http://sourceware.org/binutils/docs/gprof/
- http://valgrind.org/
- https://perf.wiki.kernel.org/index.php/Main_Page
- http://sourceware.org/systemtap/wiki
- http://gcc.gnu.org/onlinedocs/gcc/Gcov.html