System Performance Tracing and Analysis

Michel Dagenais
Mathieu Desnoyers
Pierre-Marc Fournier
Gabriel Matni

Department of Computer and Software Engineering
Ecole Polytechnique, Montreal
Summary

- Introduction
- Data providers
- Data collectors
- Data analysis
- Frameworks
- Challenges
Introduction

- Embedded real-time systems with detailed timing data.
- General purpose computers with application level tools and some operating system statistics.
- High performance parallel computing with library level (MPI) tracing.
But!? 

- Real-time multi-core high performance systems, with virtualization?
- Rapidly evolving heterogeneous systems?
Data providers

- Source-level, auto-generated
  - gcc -finstrument-functions
  - gcc -fprofile-arcs
  - gcc -ftest-coverage
  - gcc -pg

- Source-level, transformations
  - javacc
  - TXL
Data providers

- Source-level, manual insertion
  - Logging API, Java, log4cpp, .NET...
  - printk
  - evlog
  - Driver tracing infrastructure (DTI)
  - Kernel markers
Data providers

- Static binary instrumentation
  - ATOM, EEL

- Dynamic binary instrumentation
  - DTrace
  - SystemTap
  - DynInst
  - GDB
Data providers

- Sampling
  - GProf
  - OProfile
- Simulators
  - Valgrind
  - QEMU
Data collectors

- Simple counter
- Code hook (interpreted or precompiled)
- Unbuffered write
- Buffered write
- Buffered write with lock or atomic ops
- Flight recorder mode (in memory)
- Trace written to disk or network
Data collectors

- Per CPU buffers
- Atomic operations
- Count of lost events
- Zero copy of buffers in memory
- Init time tracing
Data analysis

- Total number of events
  - code coverage
  - number of read/write, pf, irq, packets...
  - number of execution samples and calls
- Elapsed time
  - Events with timestamps...
  - “time”
- Kernel Function Tracer
- LTTng
Data analysis

- Check for conditions (filters, assertions).
- Check for visual patterns.
- Compute delays (average, maximum).
- Critical path (dependency analysis) between command and answer.
- Compare traces (regression test, health monitoring...).
Example: LTT

User Mode:
- CPU 5.894726
- Elapsed 15.677299
- WaitCPU 2.358053
- WaitFork 0.000002

Syscall Mode:
- Elapsed/Calls 0.0003861
- CPU 0.308954
- Elapsed 5.119347
- WaitCPU 0.599823
- WaitFile-001.png 0.000453
- WaitFile-002.png 0.000346
- WaitFile-003.png 0.000213
...
Overhead

- test unmodified gzip: 28.16s
- sampling: 28.30
- sampling and function entry: 29.88
- basic block entry: 31.36s
Overhead

- Java logging API: 7.7us/event in memory, 208us/event to file
- DTrace: 1.18us/event, 3.1us/read event
- SystemTap: 1.3us/event
- Kernel marker inactive: 0.0005us/event
- Kernel marker active: 0.198us/event
- LTTng flight recorder: 0.746us/event
Frameworks

- Programming environment
- Views, plugins, resources, scripting
- Detailed event list
- Statistics
- Graphs
- Control flow view (Gantt chart)
- Filters, assertions
- Trace bookmarks
- Profiling, coverage, memory analysis, disk analysis, critical path, client/server requests...
Frameworks samples

- QNX, WindRiver, ZealCore... Eclipse!
- Intel VTune, Eclipse.
- DTrace, Chime.
- SystemTap, Frysk (Java-GNOME, Eclipse).
- LTTng, LTTV.
QNX Momentics
WindRiver Workbench

```c
#define HARD

BALL * p;

p = ballNew (HARD, hardBounce, hardCollide, ballMove, hardShow);
ballPlaceMovable (p);

return p;

/* hardBounce - bounce incoming ball by 190% (back where it came from)
*/
```
Intel VTune

VTune(TM) Performance Tools - Call Graph Results [localhost] - Thu Jan 24 19:05:26 2008 - Intel(R) Software Development Platform

Tuning Activities:
- **First Use** - Run this first. Find where your program is spending its time.
- **Call Graph** - Determine program flow and critical call sequences. High overhead ~8X slower. Learn more...
- **Sampling** - Identify the elements of code that use the most processor time and configure the monitored events.

Call Graph Results

- Process: /opt/intel/vtune/samples/gsexample/gsexample2a; PID:19457; Size:3

<table>
<thead>
<tr>
<th>Function</th>
<th>Calls</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Self Wait</th>
<th>Total Wait</th>
<th>Class</th>
<th>Module Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenDenormals</td>
<td>32,031</td>
<td>131,792</td>
<td>163,159</td>
<td>0</td>
<td>0</td>
<td></td>
<td>/opt/intel/vtune/samples/gs</td>
</tr>
<tr>
<td>__intel_new_proc_init.H</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td></td>
<td>/opt/intel/vtune/samples/gs</td>
</tr>
<tr>
<td>__intel_new_proc_init</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>/opt/intel/vtune/samples/gs</td>
</tr>
<tr>
<td>__get_cpu_indicator</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>/opt/intel/vtune/samples/gs</td>
</tr>
<tr>
<td>__libc_csu_init</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>/opt/intel/vtune/samples/gs</td>
</tr>
</tbody>
</table>

Run 1

- Processes
Challenges

- Variety of data providers (kernel markers, SystemTap, DTI, printk, logging API, gcc coverage and profiling...).
- Heterogeneous systems interacting (Linux kernel, QNX, VxWorks, Windows, Xen virtualization, JVM...).
Challenges

- Multi-core systems
- Distributed systems
- Real-time systems
- High-performance systems
- Low overhead tracing for all of the above
- Multi gigabytes traces
Challenges

- Integrate with different views or paradigms (UML diagrams, VM, RPC).
- Build more advanced analysis plugins (cache, RAM, virtual memory, garbage collector, real-time response, critical path, load leveling across CPUs, disks, computers).
- Mix and match data providers, data collectors and data analysis within the same framework.
Conclusion

- In depth study of the existing systems.
- Clear picture of the unmet needs of demanding users.
- Find common and complementary functionality among different tools.
- Common framework for tool interoperability.
- Needs for new tools.