System Performance Tracing and Analysis

Michel Dagenais
Mathieu Desnoyers
Pierre-Marc Fournier
Gabriel Matni

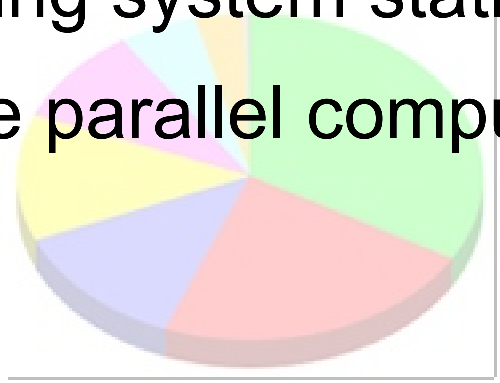
Department of Computer and Software Engineering
Ecole Polytechnique, Montreal

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.

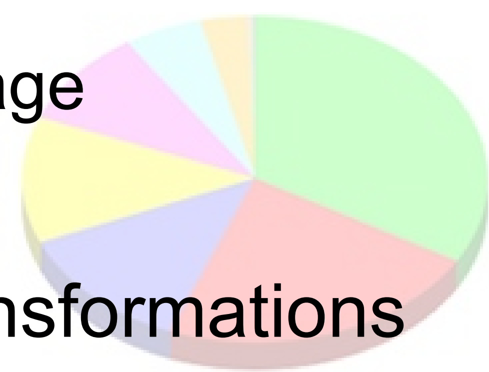


But!?

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

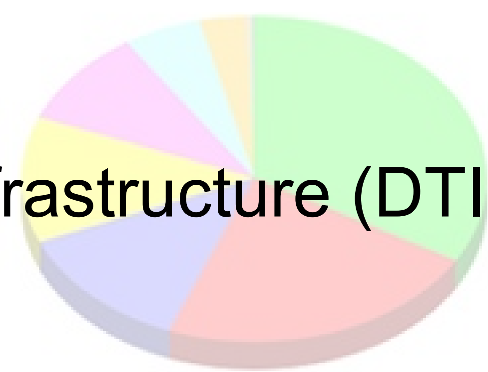
Data providers

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



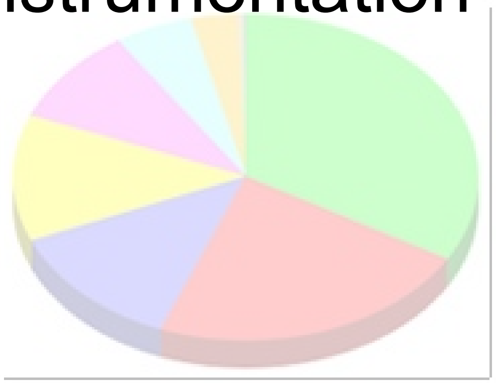
Data providers

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



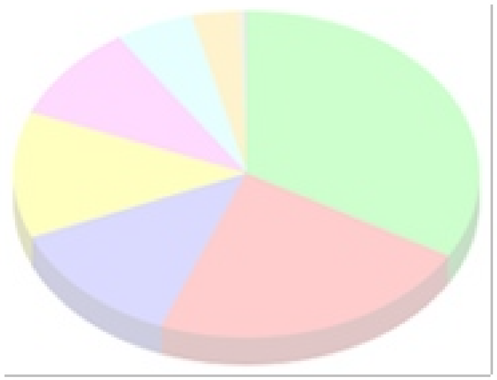
Data providers

- Static binary instrumentation
  - ATOM, EEL
- Dynamic binary instrumentation
  - DTrace
  - SystemTap
  - DynInst
  - GDB



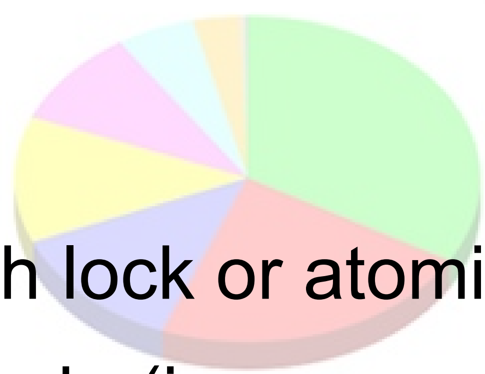
Data providers

- 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

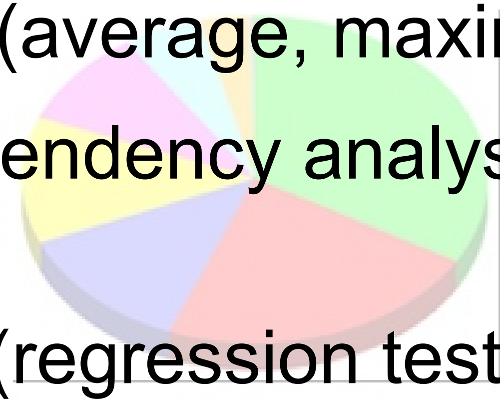
- Per CPU buffers
- Atomic operations
- Count of lost events
- Zero copy of buffers in memory
- Init time tracing

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

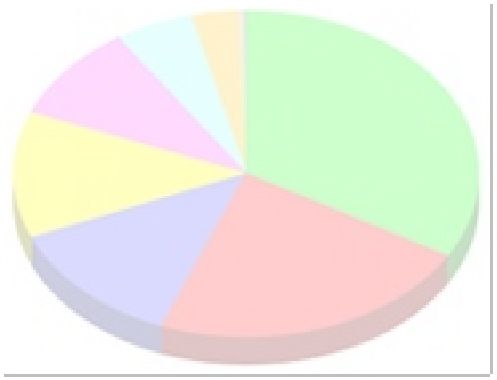
User Mode:

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

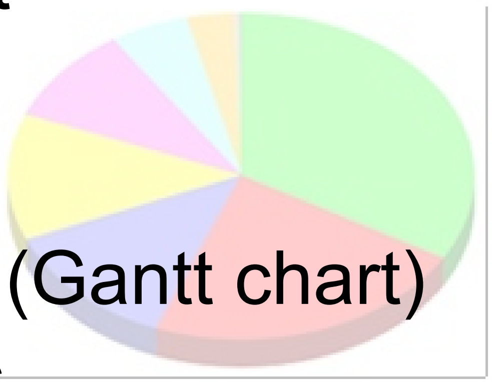
- test unmodified gzip: 28.16s
- sampling: 28.30
- sampling and function entry: 29.88
- basic block entry: 31.36s

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

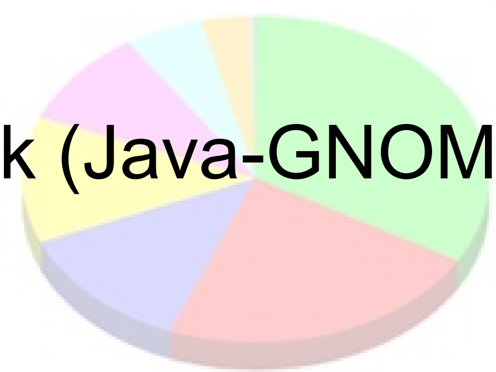
Frameworks

- 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...

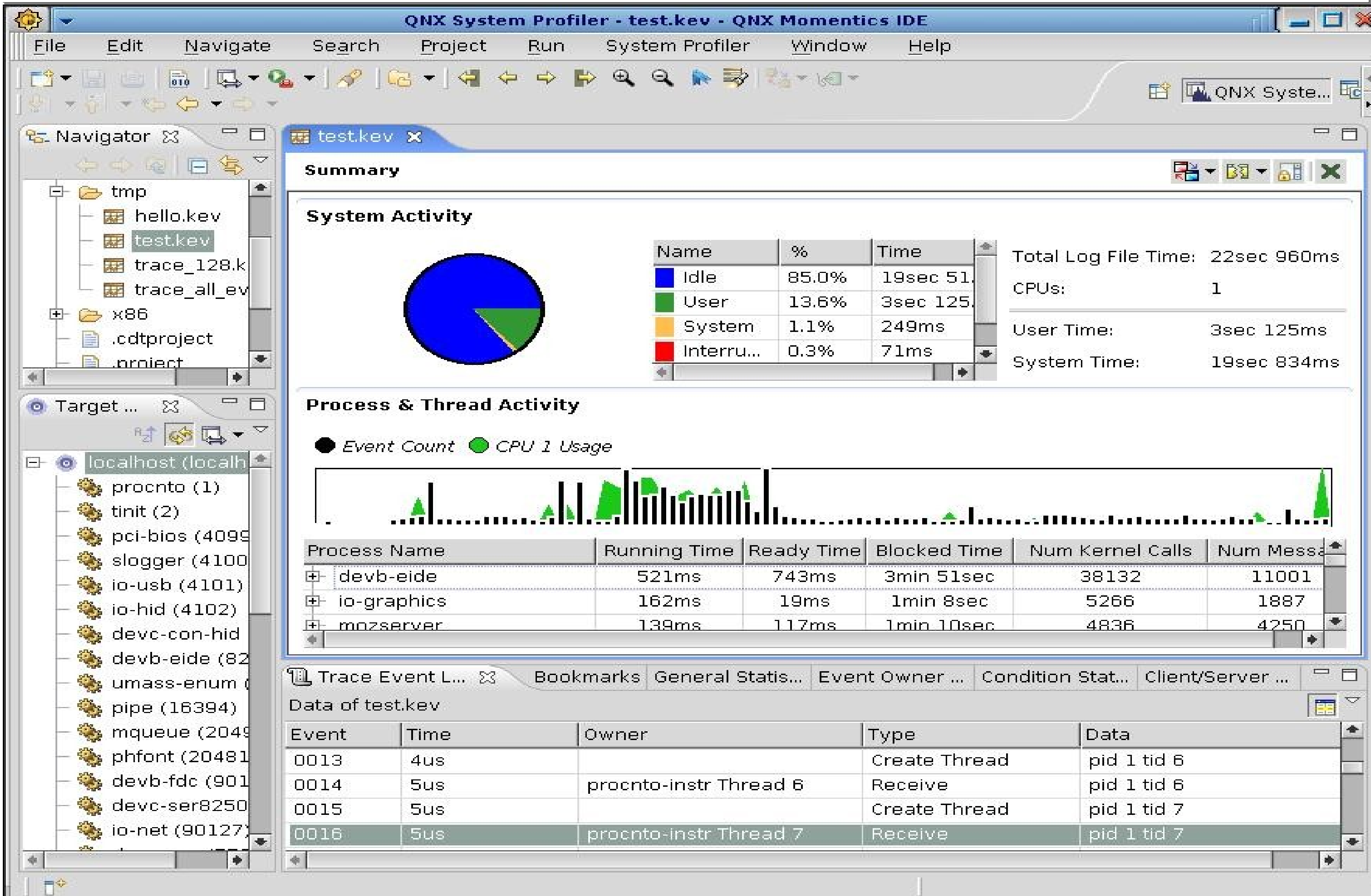


Frameworks samples

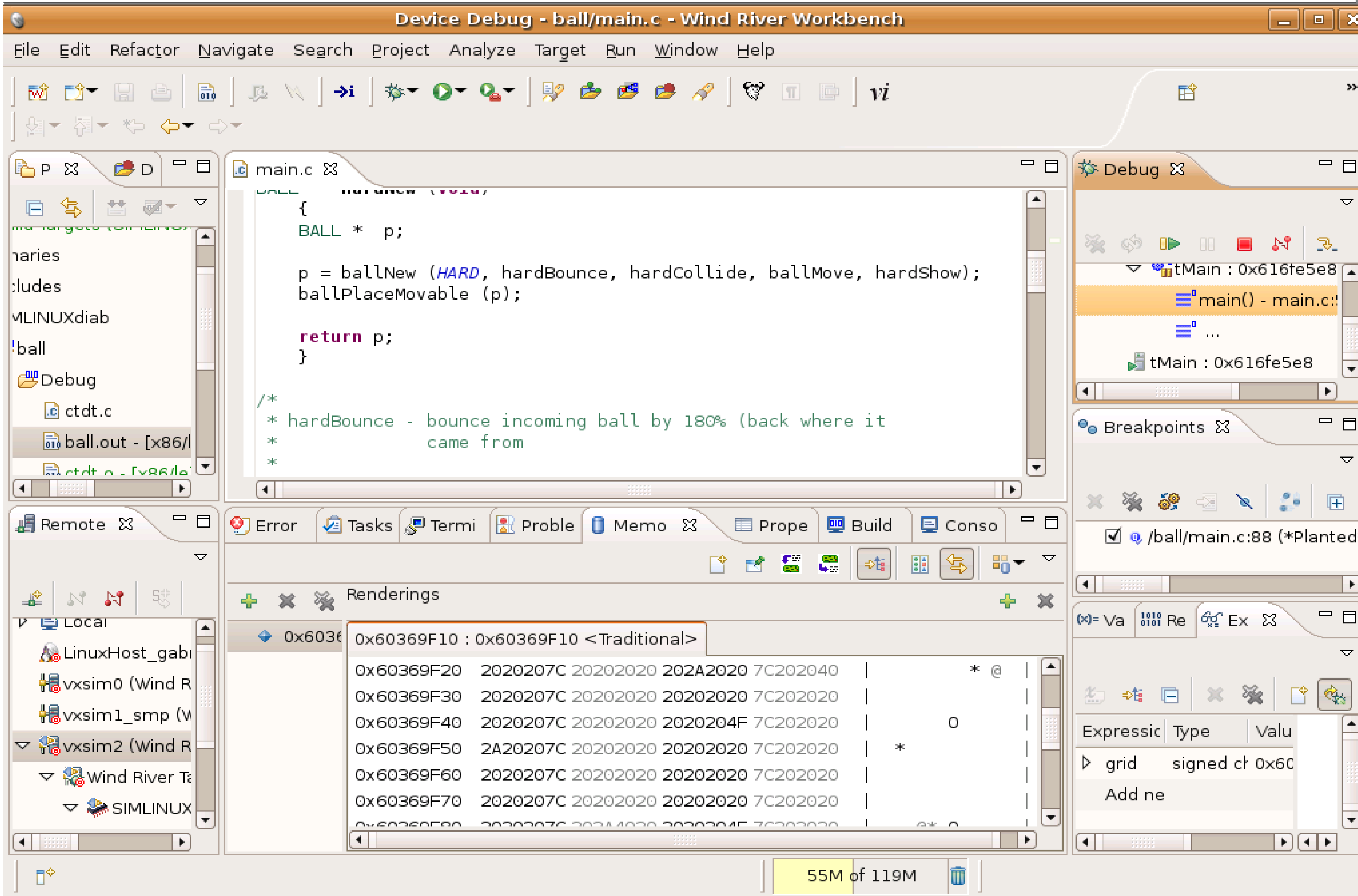
- QNX, WindRiver, ZealCore... Eclipse!
- Intel VTune, Eclipse.
- DTrace, Chime.
- SystemTap, Frysk (Java-GNOME, Eclipse).
- LTTng, LTTV.



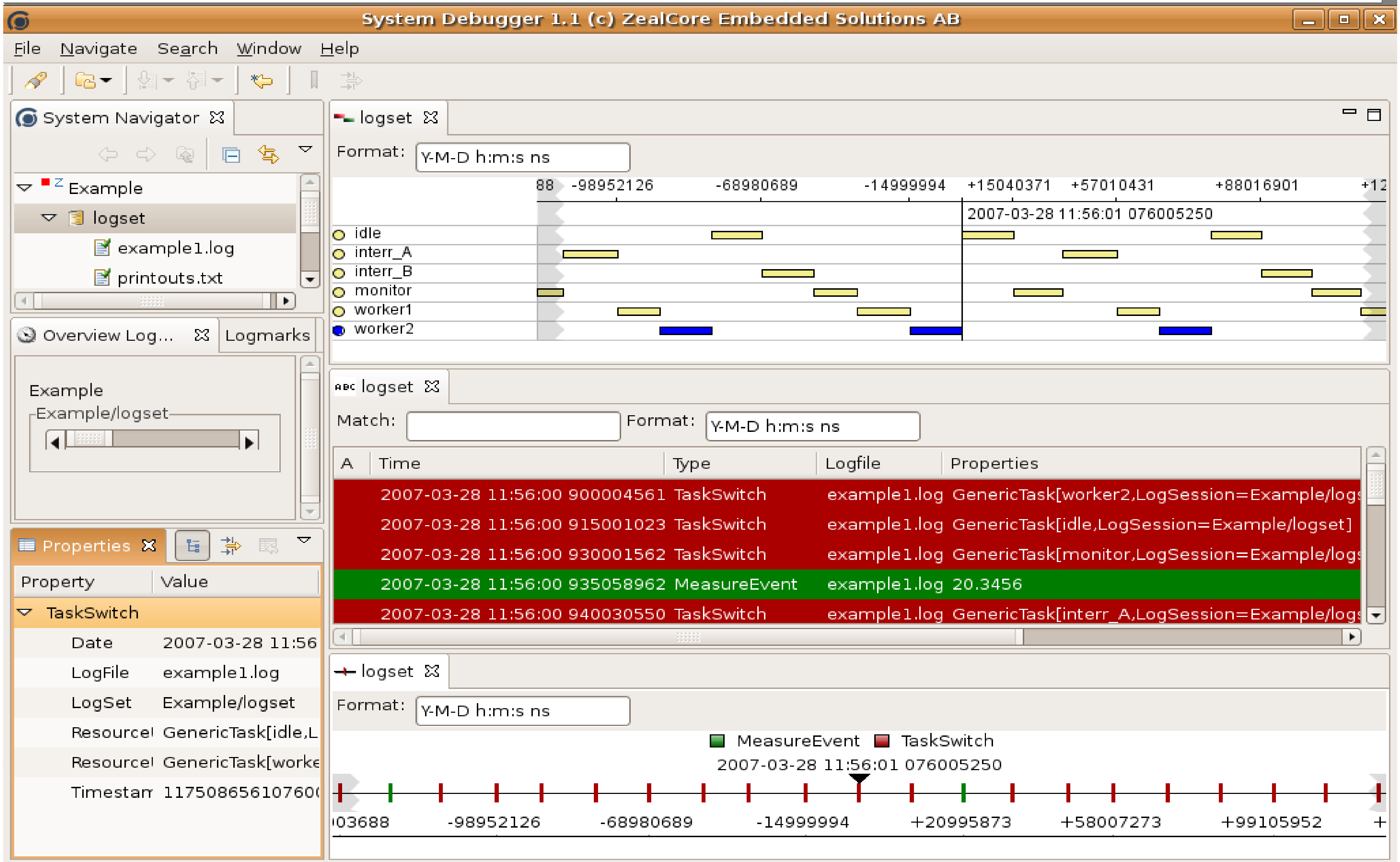
QNX Momentics



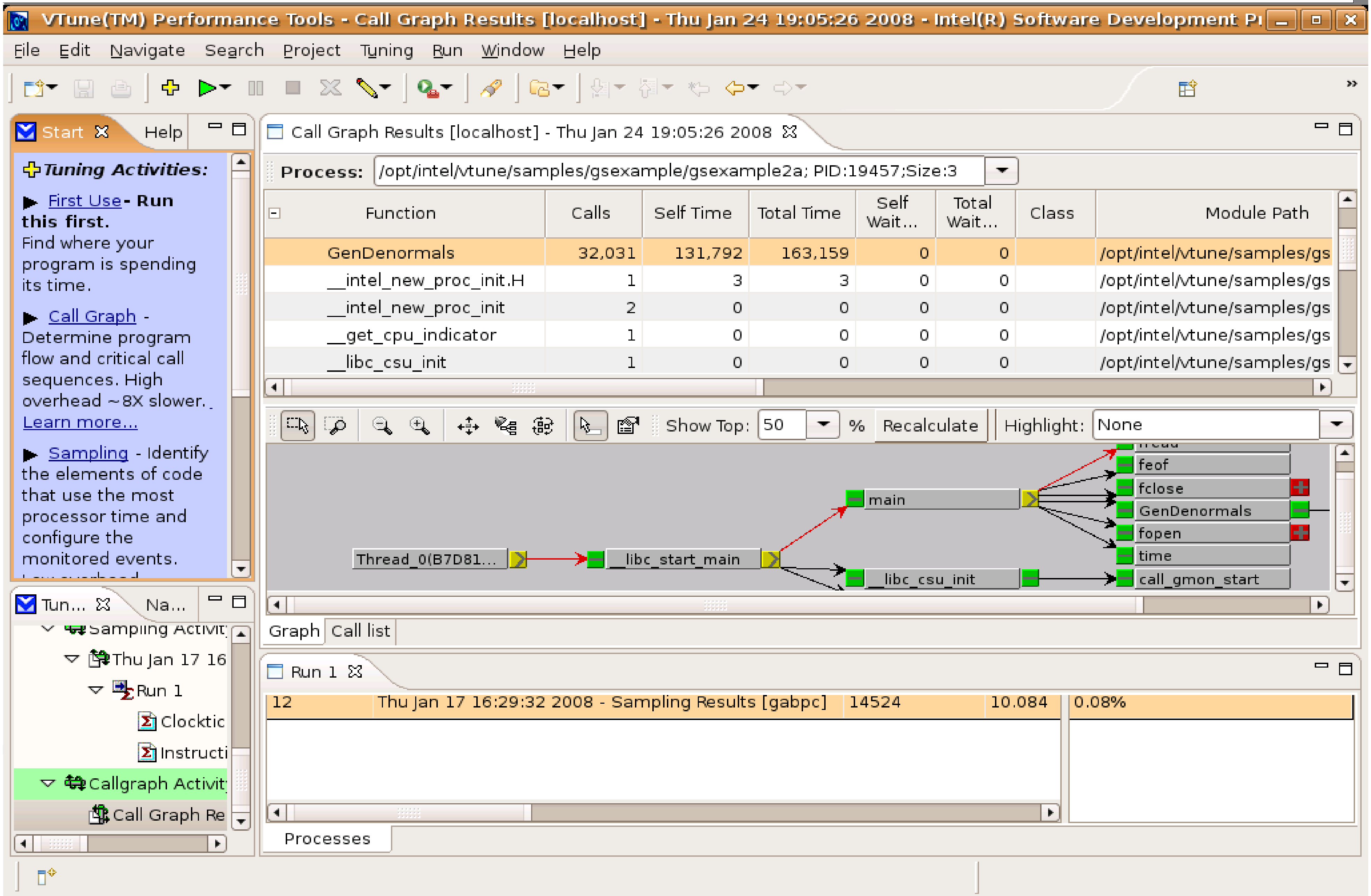
WindRiver Workbench



Zealcore SystemDebugger



Intel VTune



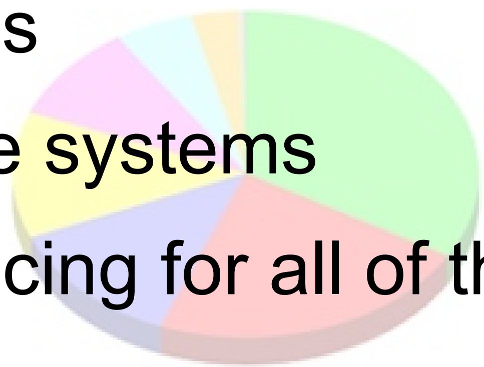
VTune(TM) Performance Tools - Call Graph Results [localhost] - Thu Jan 24 19:05:26 2008 - Intel(R) Software Development Pr
File Edit Navigate Search Project Tuning Run Window Help
Start
Help
Tuning Activities:
First Use - Run this first.
Find where your program is spending its time.
Call Graph - Determine 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 [localhost] - Thu Jan 24 19:05:26 2008
Process: /opt/intel/vtune/samples/gsexample/gsexample2a; PID:19457;Size:3
Function
Calls
Self Time
Total Time
Self Wait...
Total Wait...
Class
Module Path
GenDenormals 32,031 131,792 163,159 0 0 /opt/intel/vtune/samples/gs
__intel_new_proc_init.H 1 3 3 0 0 /opt/intel/vtune/samples/gs
__intel_new_proc_init 2 0 0 0 0 /opt/intel/vtune/samples/gs
__get_cpu_indicator 1 0 0 0 0 /opt/intel/vtune/samples/gs
__libc_csu_init 1 0 0 0 0 /opt/intel/vtune/samples/gs
Show Top: 50 %
Recalculate
Highlight: None
Thread_0(B7D81...
__libc_start_main
main
__libc_csu_init
feof
fclose
GenDenormals
fopen
time
call_gmon_start
Graph
Call list
Tun...
Na...
Sampling Activit
Thu Jan 17 16
Run 1
Clocktic
Instructi
Callgraph Activit
Call Graph Re
Run 1
12 Thu Jan 17 16:29:32 2008 - Sampling Results [gabpc] 14524 10.084 0.08%
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.