IgProf

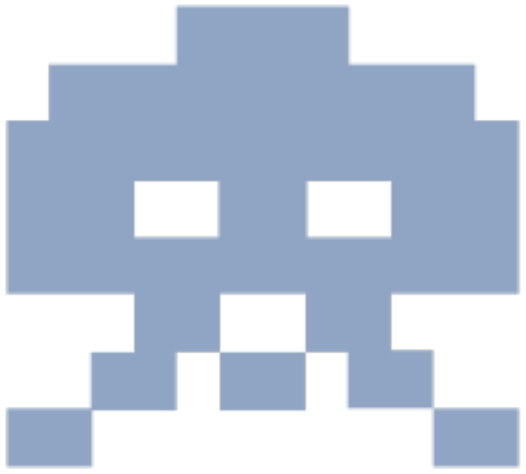
profiler support for power efficient computing

http://igprof.org

Giulio Eulisse
Fermi National Accelerator Laboratory
Filip Nybäck
Aalto University

Why profiling matters

- Before being “Power Efficient” you need to prove yourself you are not being very efficient at doing unneeded work.
- The most power efficient code is the one you do not write at all.
- Profiling is key to achieve that.



...any reference to real facts or persons is purely coincidental...

```
...
std::vector<int> foo;
for (int i = 0; i < 1000000; ++i)
  foo.push_back(0); // unneeded memory churn!
...
```

IgProf — The ignominious profiler

ignominy |ˈɪgnəmɪni|
noun [mass noun]
public shame or disgrace: *the ignominy of being imprisoned*.
ORIGIN mid 16th cent.: from French ***ignominie*** or Latin ***ignominia***, from ***in- 'not'*** + a variant of ***nomen 'name'***.

Features

Performance & memory profiling, with full call tree

Works in managed environment

- *No kernel support required*
- *No superuser privileges required*
- *Can't require recompilation or full debug symbols, must work for system/rpm-originated libraries*

Fast

- *Minimal overhead, must work for low-latency applications (GUI, web servers)*

Easy

- *Results can be shared via a simple web page*

Support for ARM32 / ARM64 added

- ***Including fast libunwind backtracing***

Memory profiling

Hooks into `malloc` & c

Three different kind of counters:

- `MEM_TOTAL`: sum of allocations in a call-path
- `MEM_LIVE`: sum of allocations from a given call-path, still present when profile dumps results
- `MEM_MAX`: largest **single** allocation in call-path

For each counter we store the number of calls and the allocated bytes. “Peak” mode also available.

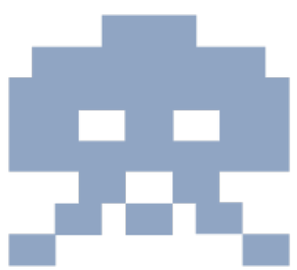
Performance profiling

How it works

Uses SIGPROF to have time uniform callbacks every ~ 1/100s. Callback stores the backtrace of where the signal happened. Supports both CPU and wall-clock time. Biggest advantage is the limited interference with the program itself.

It does converge

If you wait long enough, this actually converges to the right distribution of time spent in any given function. Works brilliantly for repetitive payloads. Unsurprisingly results correlate with ***`MEM_TOTAL`****.*



More goodies

File descriptor profiling

The previous concepts can actually be extended for any workflow which handles generic “resources” with a “cost” attached. For example you can count the number of writes / reads to a file by hooking into read / write.

Tracing exceptions

A very common pattern we had was to use C++ exceptions as a way to communicate between different parts of the program. This is both slow and leads to unmaintainable code.

Empty memory profiler

Work done by Jakob Blomer @ CERN / SFT. Useful to tune your I/O buffers. On allocation, fill memory with some magic pattern (usually zeros or `0xaa` *depending on what we are looking for). On free, scan for the same magic pattern counting untouched 4KB pages. At profile dump we report untouched pages.*

Power monitoring

ACPI

Since long time motherboards have had various ways of controlling and monitoring power usage via ACPI or similar mechanism. In particular they provide information about the CPU states which can be correlated to actual power consumption.

RAPL

Starting from SandyBridge, Intel Xeon provides the ability to limit and monitor in software the power consumptions of various parts of the SOC (so-called domains). In particular it allows to measure the power consumption of the CPU core itself, of the graphics card, and of the memory banks.

PAPI

Performance Application Programming Interface

PAPI provides an interface and methodology for use of the performance counter hardware found in most major microprocessors. Very nice interface to read performance counters (using either of perf_events, PerfCtr, Perfmon).

RAPL support

Thanks to the fact perf_events supports RAPL since Linux Kernel 3.14, it's now possible to obtain power consumption measurements via PAPI.

PAPI Support in IgProf

Support added since v5.9.12.

Requires kernel component so it's optional.

Introduces three new counters:

- *NRG_PKG (full socket)*
- *NRG_PP0 (CPU)*
- *NRG_PP1 (GPU)*

How does it work?

Similar to the performance profiler, igprof callback gets invoked at regular intervals. It checks PAPI counters at that point, if the counter overflows we count 1 in the igprof results.

Gotchas

Power information is global

RAPL information is per socket, not per process.

Overflow level empiric

While this converges, deciding the overflow level is currently completely empiric and requires better tuning.

IgProf web

Instant gratification

`igprof-navigator` *is a simple CGI script which we use to present reports as webpages. Especially in an environment of non software developers, it is important to have a clickable report at which experts can use to drive non-experts through performance optimization.*

Clang examples

MEM_LIVE, x86_64, dumped while compiling a monster machine generated file: http://cern.ch/go/k9Sw

Firefox benchmark (RoboHornet) examples

MEM_TOTAL, x86_64: http://cern.ch/go/nG7C , PERF_TICKS, x86_64: http://cern.ch/go/H9ct

CMSSW examples

…and for the High Energy Physics enthusiasts among you, a few CMS experiment software examples:

PERF_TICKS, x86_64: http://cern.ch/go/NWg9, PERF_TICKS, ARM64: http://cern.ch/go/7M9D

IgProf web

igprof_pp_25202.0_step3 - x86_64, igprof-navigator

Back to profiles index

Counter: PERF_TICKS, first 1000 entries

Sorted by cumulatative cost

(Sort by self cost)

Rank	Total %	Cumulative	Symbol name
1	100.00	2,130.54	<spontaneous>
3	97.40	2,075.14	__libc_start_main
2	97.40	2,075.14	_start
4	97.36	2,074.30	main
5	97.31	2,073.21	main::{lambda()#1}::operator()() const
6	94.11	2,005.03	edm::EventProcessor::runToCompletion()
7	94.11	2,005.02	boost::statechart::state_machine<statemachine::Machine, statemachine::Starting, std
12	92.60	1,972.82	edm::EventProcessor::readAndProcessEvent()
11	92.60	1,972.82	statemachine::HandleEvent::readAndProcessEvent()
10	92.60	1,972.82	statemachine::HandleEvent::HandleEvent(boost::statechart::state<statemachine::Handl
9	92.60	1,972.82	boost::statechart::state<statemachine::HandleEvent, statemachine::HandleLumis, boos
8	92.60	1,972.82	boost::statechart::simple_state<statemachine::FirstLumi, statemachine::HandleLumis,
15	92.60	1,972.81	edm::EventProcessor::processEventsForStreamAsync(unsigned int, std::atomic<bool>*)
14	92.60	1,972.81	edm::StreamProcessingTask::execute()
13	92.60	1,972.81	tbb::internal::custom_scheduler<tbb::internal::IntelSchedulerTraits>::local_wait_fo
16	92.59	1,972.62	edm::EventProcessor::processEvent(unsigned int)

IgProf web

Counter: PERF_TICKS

Rank	% total	Counts: to / from this	Counts: Total	Paths: Including child / parent	Paths: Total	Symbol name
	0.06	1.28	21.22	1	1	RecoMuonValidator::analyze(edm::Event const&, ed
	0.21	4.48	6.95	1	1	ObjectSelector<SingleElementCollectionSelector<s
	0.27	5.78	36.41	1	1	MuonTrackValidator::analyze(edm::Event const&, e
	1.35	28.77	30.36	1	1	MTVHistoProducerAlgoForTracker::fill_recoAssocia
	5.21	111.03	817.76	1	1	MultiTrackValidator::analyze(edm::Event const&,
[31]	**7.10**	**19.08**	**132.26**	**5**	**5**	**TrackingParticleSelector::operator()(TrackingPar**
	2.51	53.38	56.47	5	12	TrackingParticle::charge() const
	1.82	38.74	80.06	5	311	log
	0.88	18.76	26.10	4	14	ROOT::Math::Cartesian3D<double>::Eta() const
	0.61	13.07	14.19	5	9	TrackingParticle::vertex() const
	0.28	6.02	6.14	5	8	TrackingParticle::momentum() const
	0.05	1.08	1.08	4	4	TrackingParticle::numberOfTrackerLayers() const
	0.03	0.62	2.99	2	5	TrackingParticle::eventId() const
	0.03	0.58	1.86	2	15	TrackingParticle::pdgId() const

Back to summary