Profile and reorder code execution in Geant4 to increase performance

A Google Summer of Code Project

Stathis Kamperis

Department of Physics
Aristotle University of Thessaloniki
Greece

ekamperi@gmail.com

July, 2012

Overview

Geant4

- Large source code base
- Lots of classes
- Highly conditionalized code
- Complex numerical calculations

Full CMS

- Complex geometry and physics

Very low visibility to the runtime aspects of the simulations

Profiling targets

- Full CMS experiment
- Simplified Calorimeter
  - Faster initialization, faster profiling cycles
  - Simpler Geometry
    - Useful for examining how geometry affects performance
- Examples bundled with Geant4

Profiling tools

Ported Geant4 to Solaris 11/x64

- DTrace
  - A dynamic tracing framework
  - Available also in Mac OSX (an officially supported platform by Geant4)
  - Fine-grained profiling
- mdb (Modular debugger)
- cputrack
  - Access CPU performance counters
  - data cache misses, instruction cache misses, branch mispredictions, ...
- libumem
- pbind (to bind profiled process to a specific CPU)
- A pseudo device driver to invalidate CPU caches on demand
- Visualisation tools and Statistics
  - gnuplot, ggplot2, R

Profiling tools - Alternatives

Not propagandizing in favor of Solaris

Alternatives for Linux users:

- DTrace → SystemTap
- mdb → gdb
- cputrack → perf, cachegrind
- libumem → valgrind
- pbind → taskset

The rest are common for both platforms (visualisation and statistics)

DTrace

- pid provider
- Flamegraphs
- USDT (user-level statically defined tracing)
- Speculative tracing
- All of the above combined

Particle "bunching"

Definition Process *same* particle types before switching to another particle type. E.g.,

$$e^-, e^-, \ldots, e^-, \gamma, \gamma, \ldots, \gamma, \ldots$$

Why Better *cache utilisation*

Number of stacks we are using: 5

1. Primary particles + everything not belonging to:
2. Neutrons
3. Electrons
4. Gammas
5. Positrons

Particle "bunching" - Problems

Problems

- Stacks can grow very large
  - e.g., when processing electrons, the gamma stack explodes, and vice versa
- So we have to restrict them, which leads to another problem
  - What is the optimal size for each one?
  - How much aggressively should we process a track, once it reached its upper limit ?

Particle "bunching" - Problems cont.

If we allow *too large* sizes

- we diverge a lot in terms of geometry (it hurts)

If we allow *too small* sizes

- we switch too often between stacks, and we thrash (it hurts)

DTrace - Simple examples 1

How many times does the G4Allocator grow in size during 100 simulated events ?

```
# dtrace -n '
pid$target::*G4AllocatorPool*Grow*:entry
{
    @ = count();
}' -c '/home/stathis/geant4.9.5.p01/bin/full_cms ./bench1_100.g4'

            5921
```

How much time do the above resizes consume ?

```
dtrace -n '
pid$target::*G4AllocatorPool*Grow*:entry
{
    self->ts = vtimestamp;
}

pid$target::*G4AllocatorPool*Grow*:return
/self->ts/
{
    @ = sum((vtimestamp - self->ts)/1000);
    self->ts = 0;
}' -c '/home/stathis/geant4.9.5.p01/bin/full_cms ./bench1_100.g4'
            4859  # ~5 msec
```

DTrace - Simple examples 2

How do we skip the initialization part of Geant4/Full CMS ?

- Use a predicate that checks whether we are inside the DoEventLoop()

```
dtrace -n '
BEGIN
{
    tracing = 0;
}

pid$target::*DoEventLoop*:entry { tracing = 1; }
pid$target::*DoEventLoop*:return { exit(0); }

someprobe
/tracing != 0/
{
    ...
}
' -c '/home/stathis/geant4.9.5.p01/bin/full_cms ./bench_100.g4'
```

USDT (user-level statically defined tracing)

Allows to place custom probe points in application code

- Available both in development and production builds
  - No need to recompile with a debug flag set
- DTrace dynamically activates the probes when asked
  - By dynamically modifying the instructions of the profiled app
- Negligible overhead when not in use (a few NOPs)
- Take advantage of DTrace rich reporting capabilities (aggregations)

USDT - Example 1

Objective Everytime we *push* a track to the track manager or we *pop* one from it, dump the sizes of all stacks.

```
# dtrace -qn '
simple$target:::
{
    printf("%s track=%d size=%d\n", probefunc, arg0, arg1);
}'
-c '/home/stathis/geant4.9.5.p01/bin/mainStatAccepTest ./exercise.g4' | c++filt -np
...
G4SmartTrackStack::PushToStack track=0 size=1
G4SmartTrackStack::PopFromStack track=0 size=0
G4SmartTrackStack::PushToStack track=2 size=1
G4SmartTrackStack::PushToStack track=2 size=2
G4SmartTrackStack::PushToStack track=2 size=3
G4SmartTrackStack::PushToStack track=0 size=1
...
G4SmartTrackStack::PopFromStack track=2 size=446
G4SmartTrackStack::PopFromStack track=2 size=445
G4SmartTrackStack::PopFromStack track=2 size=444
G4SmartTrackStack::PopFromStack track=2 size=443
```

USDT - Example 2

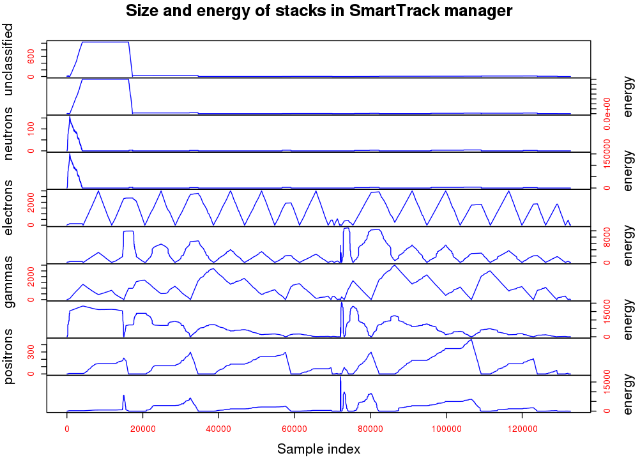
Objective Print the distribution of stack sizes for unclassified particles (primaries + any particle not beloning to the set n^0, e^-, γ, e^+

```
# dtrace -qn '
simple$target:::
/arg0==1/
{
    @["distribution of 1st stack's size"] = quantize(arg1);
}' -c '/home/stathis/geant4.9.5.p01/bin/mainStatAccepTest ./exercise.g4'
^C
  distribution of 1st stack's size
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |                                         111
               1 |                                         308
               2 |@                                        963
               4 |@                                        2241
               8 |@@                                       3193
              16 |@@@                                      4452
              32 |@@@@@                                    7700
              64 |@@@@@@@@@@                               15574
             128 |@@@@@@@@@@@@@@@                          23497
             256 |@@@                                      4459
             512 |                                         0
```

USDT - Example 3

Objective Visualize the size of stacks and the total energy of their particles

The following graph is from a simulation of 2 events in Full CMS:



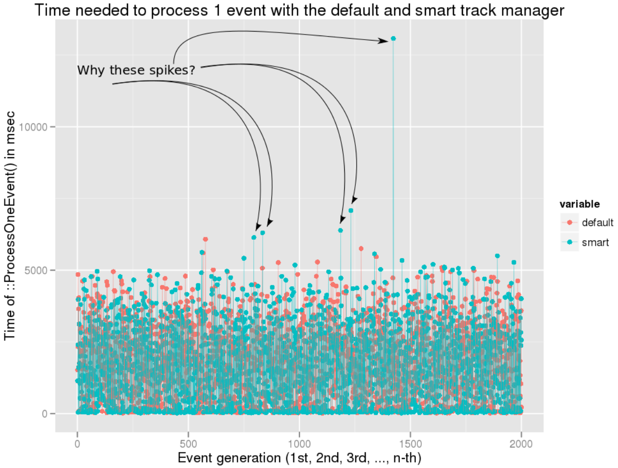
Speculative tracing - Introduction

Definition The ability to tentatively trace data and then later decide whether to commit the data to a tracing buffer or discard it.

From DTrace guide

Speculative tracing - A real use case

Problem Some `ProcessOneEvent()` need more than average time to complete

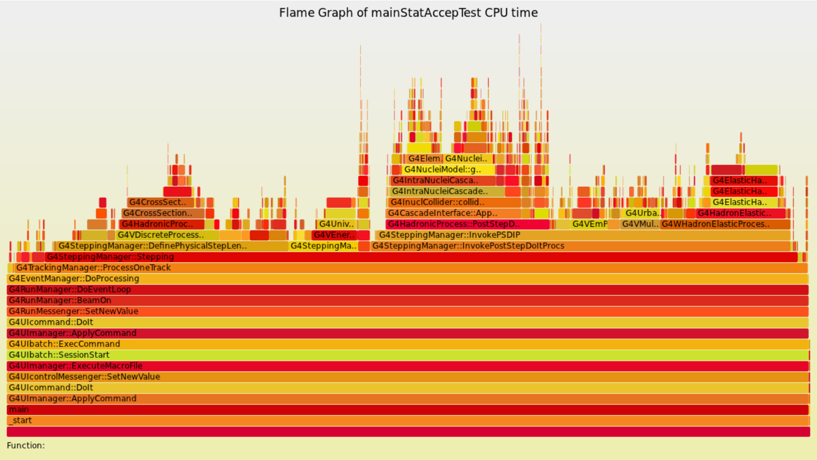


Speculative tracing - A real use case

Strategy We are going to trace all ProcessOneEvent() calls, but commit to our tracing buffer *only* those that behave bad.

Flame graphs

Definition Flame graphs are a visualization method for sampled stack traces



Flame graphs

Scope Anything that can be sampled by DTrace can be visualized as a flame graph

- Function execution time
- Data cache misses
- Instruction cache misses
- Branch mispredictions
- Memory allocation sizes
- ...

Developed by Brendan Gregg

Flame graphs

Hints

- Identification of hot code-paths
- The x-axis is the sample population
- The y-axis is the stack depth
- The width of a box is proportional to the measured quantity. E.g.,
  - A wide box means that a function either takes a lot of time to complete or that it is called too often (in either case the probability that its stack trace is sampled increases)
- The colors are *not* significant (they are picked at random to be "warm")

Flame graphs - Validation

Problem How do we know that flame graphs are valid ?

We picked a function that caused only few cache misses, and made it on purpose *invalidate all the cpu caches*.

We then *regenerated* the flame graph and the function's box in the was *vastly increased*.

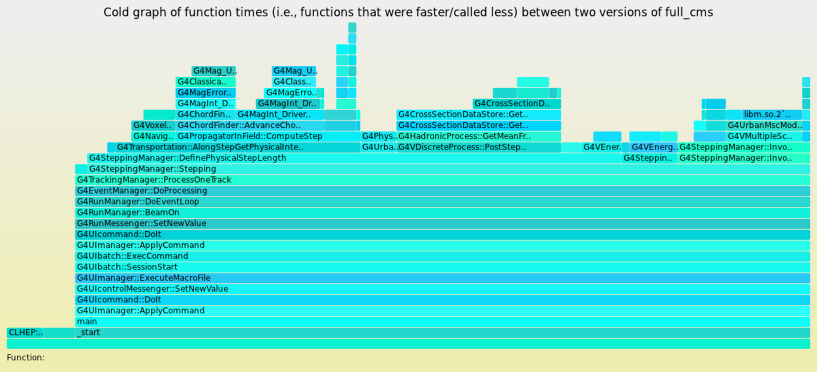
Flame graphs - Deltas

Definition A "delta" is a new graph derived by the subtraction of two flame graphs

- Examine how a property's value increases or decreases between two versions of the same application. E.g.,
  - Which functions became faster and which ones slower
  - Which functions cause more instruction cache misses and which ones less
  - ...
- A delta graph consists of two graphs, the flame graph and the cold graph

Flame graphs - Deltas - Cold graph

Example of a cold graph



The end

Thank you. Questions?