# 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

September, 2012

## Goals

- **Profile** Geant4 to identify potential targets of optimization (first half of GSoC period)
- **Reorder code execution** to improve serial performance (2nd half)

In reality Goals were interchangeable

## Methods

- **Port of Geant4 to Solaris** 11/amd64 to access **DTrace** profiling tool
- **Tool to compare 2 versions** of an application and generate an HTML report
  - Tested on FullCMS and Simplified Calorimeter
  - Example (clickable): `http://island.quantumachine.net/~stathis/geant4/run-5550/smartstack.html`

## DTrace - Introduction 1/2

- "D" stands for Dynamic- it dynamically instruments a running program, by modifying its instructions while it is executing
- **Deep inspection**
  - Arbitrary instructions
  - CPU registers
  - CPU hardware counters, etc
- **Sophisticated profiling** (e.g., speculative tracing)
- **Built-in aggregation** functions
  - count, sum, avg, min, max, stddev, {l,}quantize
- **Negligible runtime overhead**

## DTrace - Introduction 2/2

- **Safe** to use in production environments
  - Safety was one of the central architectural decisions upon DTrace was built
  - Explains why some common language constructs aren't supported (e.g., for-loops)
- **No source code modification** of the profiled application needed
- Can operate on **multithreaded** programs (has support for thread-local variables)
- Runs on **Mac OSX** out of the box; Linux port is on the way
- Profiling done via a simple language called D (resembling C and awk)
  - Scripts can be shared, reviewed, reused, made be run unattended

## Overview of ideas

Some of the ideas explored

- **Particle bunching (G4SmartTrackStack)**
- Caching of cross-sections calculations in hadronic processes (G4CrossSectionDataStore)
- Reducing branch mispredictions in Value() (G4PhysicsVector)
- Hard-coded stepping manager (G4SteppingManager)
- Caching values of ln(Energy) (G4Track)

## Particle "bunching" 1/2

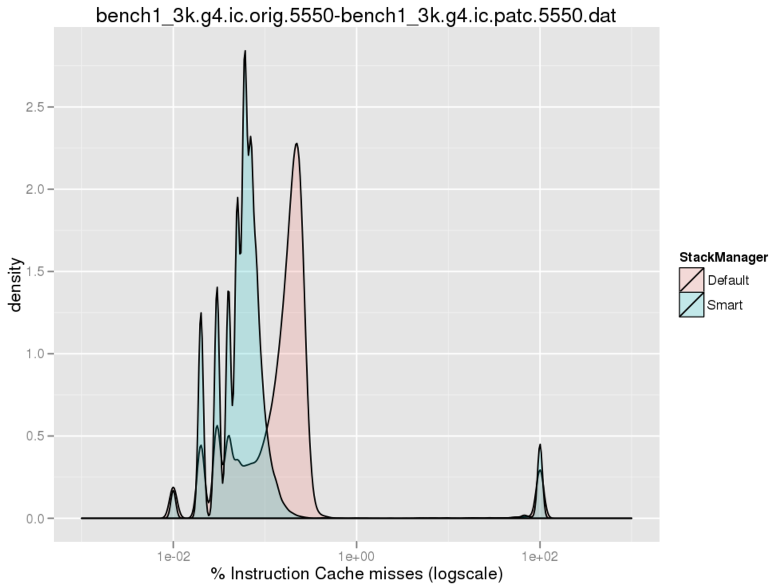
Definition Process *same* particle types before switching to another particle type. E.g.,

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

Why Better *cache utilisation* due to access to the same physics list

4-5% **persistent** reduction in total execution time in FullCMS experiment (less in SimplifiedCalorimeter)

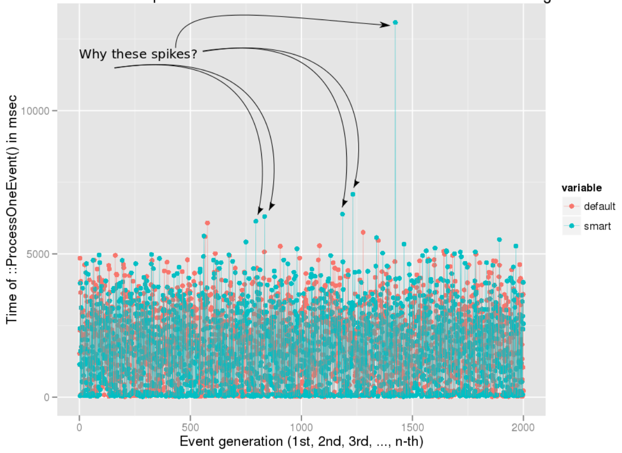
## Particle "bunching" 2/2



## Speculative tracing - A real use case

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

Time needed to process 1 event with the default and smart track manager



## Speculative tracing - A real use case

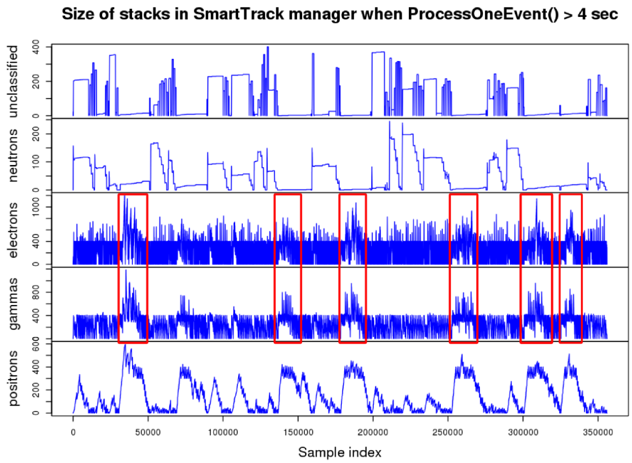
Strategy We are going to trace all `ProcessOneEvent()` calls, but commit to our tracing buffer *only* those that behave bad.

In this context, "trace" refers to looking at stacks' sizes when `ProcessOneEvent()` stalls while processing the event.

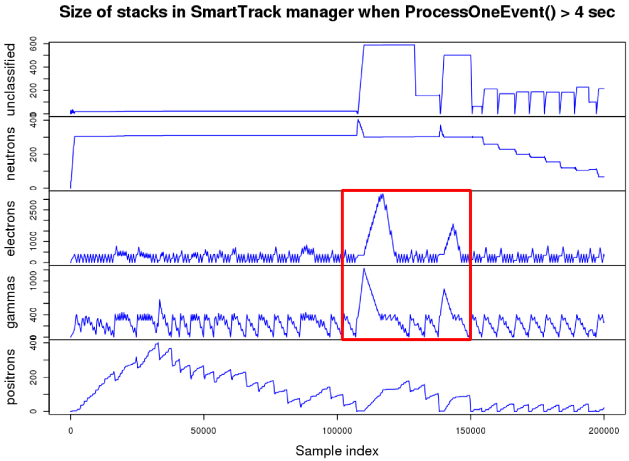
## Speculative tracing - A real use case cont.

Hint The maximum desired size for all stacks was requested to be 400.

$e^-$ and $\gamma$ too often will not honour that limit.



# Speculative tracing - A real use case cont. - Zoom 1/2



## Caching cross-sections in hadronic processes

Problem A flamegraph showing CPU utilization identified cross-section calculations in hadronic processes as a significant contributor

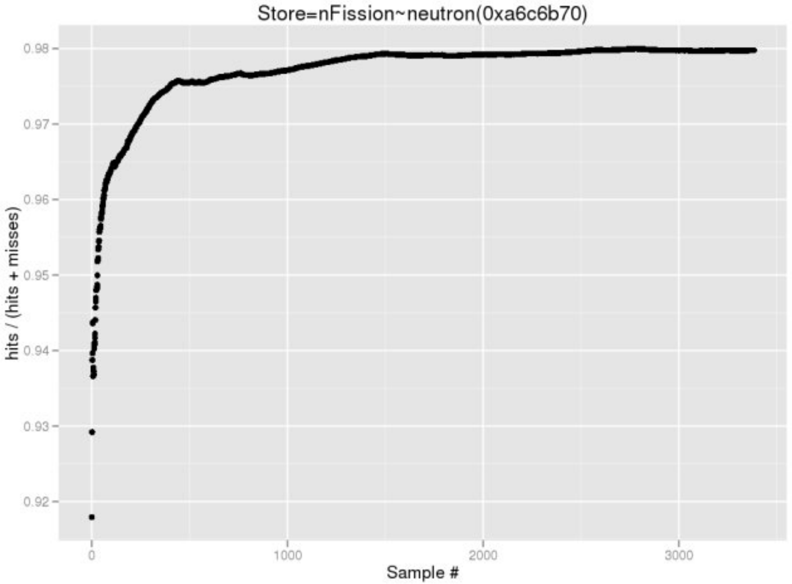
Idea Cache the values on some bin energy level

Result After many iterations, we have a version where the **hits ratio are very high** and there's **probably a benefit of a few percent** (not yet quantified)

TODO Run enough simulations to extract the benefit. Study the **ramifications of bin'ing the energy** from the physics POV.

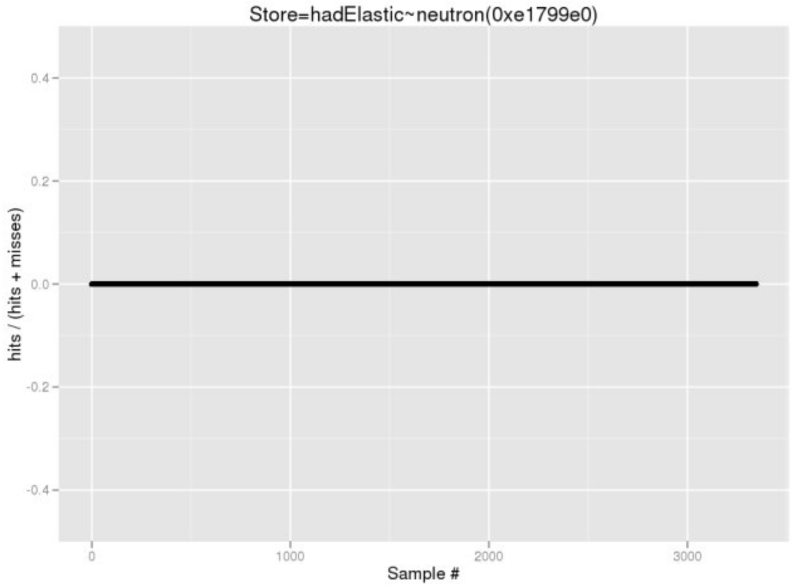
## Not all hadronic processes are cache-friendly 1/2

`http://island.quantumachine.net/~stathis/geant4/hits`



## Not all hadronic processes are cache-friendly 2/2

`http://island.quantumachine.net/~stathis/geant4/hits`



## Reducing branch mispredictions in G4PhysicsVector::Value()

"Problem" A flamegraph showing branch mispredictions identified G4PhysicsVector::Value() as a significant offender

Idea Try to collapse some of the if-blocks, gaining branch predictability, but executing more cpu instructions

Result The branch mispredictions reduced (expected), but the average time spent in that function was actually larger

## Reducing branch mispredictions in G4PhysicsVector::Value()

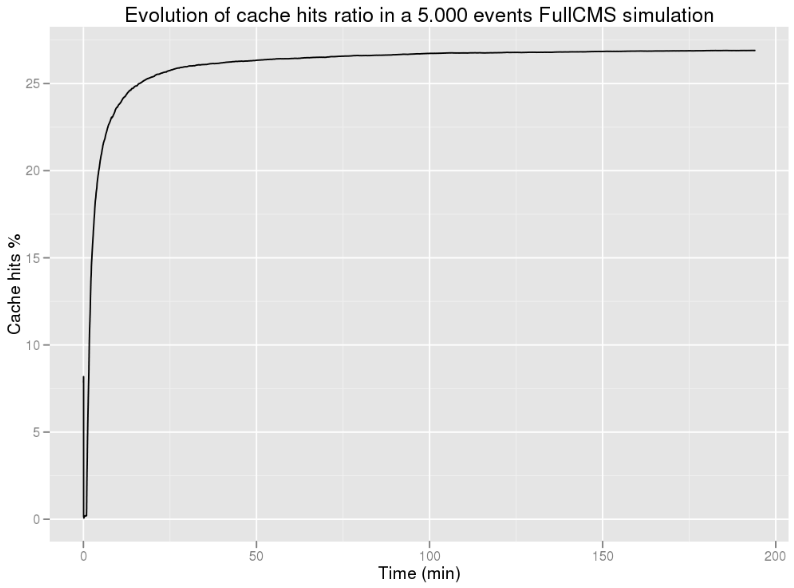
Objective Calculate the cache hits ratio in G4PhysicsVector::Value()

```
# dtrace -qn '
/* 0xc0 is the offset inside Value() where a fast cache hit takes place */
pid$target::_ZN15G4PhysicsVector5ValueEd:c0
{
    @branch = count()
}

pid$target::_ZN15G4PhysicsVector5ValueEd:entry
{
    @total = count()
}

tick-100ms
{
    printa(@branch)
    printa(@total)
}' -c '/home/stathis/geant4.9.5.p01/bin/full_cms ./bench1_5k.g4' -o val
```

## Reducing branch mispredictions in G4PhysicsVector::Value()



## Reducing branch mispredictions in G4PhysicsVector::Value()

- The benefit of caching outweighs (as reality dictates) the penalty of branch mispredictions
- The eventual ratio is higher than that I had initially in mind

- **Lesson learnt**: let the system reach its equilibrium before drawing any conclusions
- **Lesson learnt**: if you optimize 1 micro-benchmark, you may hurt another (or more)

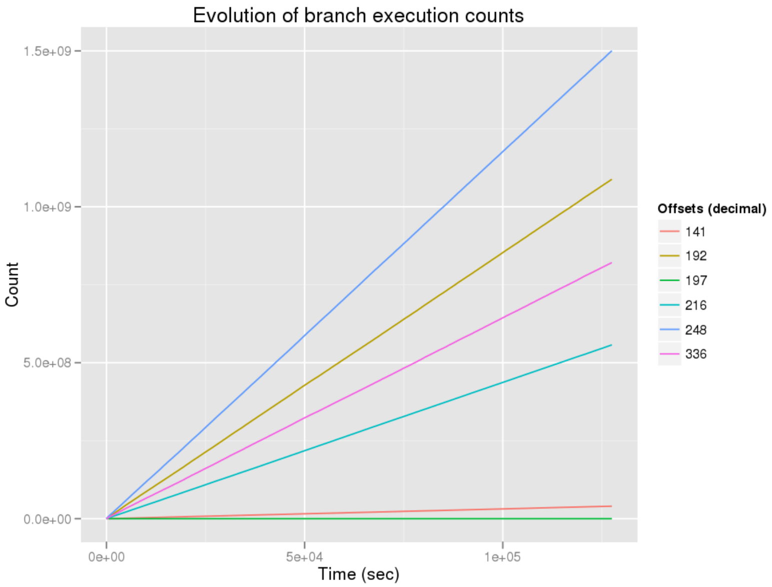
## Reducing branch mispredictions in G4PhysicsVector::Value()

Enter the "rabbit" hole

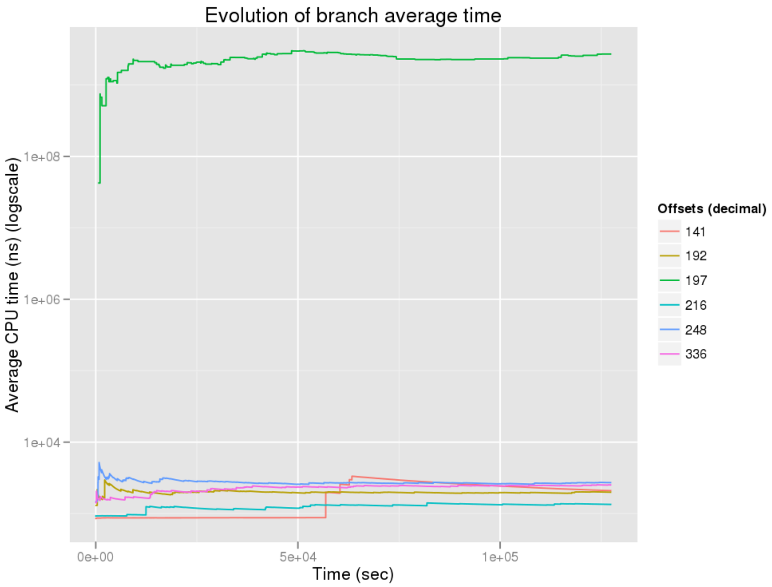
- Question ::Value() has many distinct branchs. How **fast** are compared to each other ?
- Question ::Value() has many distinct branchs. How **many times** is each one executed ?

I will skip the DTrace script which is a bit long for a slide, but here are the graphs:

## How many times is each branch in ::Value() executed ?



## How fast are the branches in ::Value() compared to each



## Hard-coded stepping manager (G4SteppingManager)

Data For some processes, {AtRest, AlongStep, PostStep}GPIL calls are placeholders.

Idea Replace them with direct hard-coded calls, instead of relying on compiled to do the dynamic dispatching

Result No detectable benefit. **But**, it was done quickly, so it deserves further exploration

## Caching values of ln(Energy) (G4Track)

Idea Cache the logarithm of energy inside G4Track

Exploration A preliminary analysis with DTrace showed that the anticipated benefit would be less than 1%

Result It hasn't been actively pursued until now

## The end

Thank you. Questions?