Krzysztof Genser, Fermilab/SCD
Geant4 Collaboration Meeting
SLAC, September, 2011

Introduction to Profiling Geant4 Applications with FAST

Talk Outline

- What is FAST?
- People involved
- How to use the profiler?
  - The main emphasis of the talk
- Example profiling plots and call graphs
- Summary

FAST

- FAST (Flexible Analysis and Storage Toolkit)
  - set of tools designed to analyze the performance - primarily the speed - of singly-threaded programs written in C++, C
  - It has components for the collection, analysis, and display of the performance data
- Can be used (almost) standalone
  - Does not require code instrumentation
    - except for building the application with the debug symbols and frame pointers
  - Generates mostly text files which can be inspected “by hand” (or eye)
- Is designed with an exploratory type of analysis in mind

FAST cont'd

- FAST is available from
  - https://cdcvs.fnal.gov/redmine/projects/fast
- Current releases of FAST include a copy of libunwind which is built automatically when building FAST
  - http://www.nongnu.org/libunwind
    - based on the libunwind git repository head as of ~April 25th, 2011
      - includes many contributions by Lassi Tuura
- Data Collection (i.e. running the profiler) is currently supported on Linux
  - tested on Scientific Linux (SL5)
- Data Analysis is supported where underlying tools are supported (Linux, Windows, Mac OS X)

People Involved

- People working (usually at a fraction of their time) on some aspects of the project over the last year or so:
  - Marc Paterno
  - Anthony Baldocchi – NIU Intern
  - Jim Kowalkowski
  - Krzysztof Genser

Prerequisite Tools

- C++ compiler
  - so far tested with the GNU g++, versions 4.1 through 4.5
- GNU binutils, specifically libbfd library
  - to build a dynamic library
- CMake
  - version >= 2_6)
- Ruby
  - version >= 1.8.7 http://www.ruby-lang.org
  - for call graph generation
- Graphviz
  - (version >= 2.24) http://www.graphviz.org
  - for call graph visualization
- Optionally, ps2pdf
  - http://www.ghostscript.com
  - for PDF output of call graphs
- Optionally, R
  - version >= 2.11 http://www.r-project.org
  - for analysis of data

FAST Components

- SimpleProfiler
  - Sampling profiler
    - with a default sampling frequency of 100Hz
    - with an overhead of up to 1%
- ProfGraph
  - call graph analysis tool
    - uses Graphviz to produce a visualization of the call graph represented in the data collected by SimpleProfiler

Geant4 Applications we have been profiling

- CMSSW cmsRun
  - recently with patches by Sunanda Banerjee for more recent versions of Geant4
- SimplifiedCalo
  - from Andrea Dotti; minimally modified to add timing printout and to read a PYTHIA event file
    - example results in this talk (Geant4 9.4.p01)
- Mu2e Offline program
  - simulating conversion electrons (~105MeV)
- All with QGSP-BERT (or a default) physics list

Basic Profiling/Code Analysis Steps

- Get/build/setup FAST and its prerequisites
- Build the application to be profiled
  - with the debug symbols and frame pointers (esp. in highly optimized builds)
- Run the application with the SimpleProfiler:
  - profrun [options] application [application options]
- Inspect the profdata_<n>…_<m>_names files
  - e.g. for most compute intensive functions
  - this step can be done using text tools like cat and grep
- Look at the call graphs
  - profgraph -n profdata_<n>…_<m>
  - profgraph -n profdata_<n>…_<m> nfunc
- Look at the code based on the above results…

Building and Setting up FAST

- Fetch the latest README file from:
  - https://cdcvs.fnal.gov/redmine/projects/fast
  - follow instructions in it (sketched below)
- Fetch the latest tar file and uncompress it to a location of choice
  - > tar zxf fast…
- Make a bin directory and cd to it
  - > cd <path-to-bin>/bin
- Run CMake
  - > cmake <path-to-fast>/fast
- Run make
  - > make
- Establish the work environment
  - > source <path-to-bin>/etc/setup

Building and Profiling an Application

- Build the application to be profiled
  - with the debug symbols and frame pointers
    - e.g. -g -O2 -fno-omit-frame-pointer -DNDEBUG
- Run the application with the SimpleProfiler:
  - profrun [options] application [application options]
    - e.g. profrun SimplifiedCalo inputfile
      - profrun -h provides help info
  - profrun also extracts call path, function call and library call information from the collected raw data

SimpleProfiler Output files

- All file names have the format profdata_<n>_<m>_<ts>*
  - n and m are identifying process ids (or possibly child process ids), ts is a time stamp
- The most important files are the
  - …_names
  - …_paths
  - …_libraries
- Most output files are in human-readable tab-separated text format
- A full description of the output files is provided in the FAST Users' Manual

Look at the Data profdata_…_names file content

- For each function seen profdata_…_names file contains:
  - A unique function id
  - The address of the function
  - The leaf, total, and path count for the function, and the leaf and path fractions
  - The library in which the function is found
  - The mangled and unmangled names of the function
  - e.g.: (exp69,10,1)

```
22
0x310e4106d0
118686        118686        118686
0.0763286     0.0763286
"libm.so.6"
"__ieee754_log"        "__ieee754_log"
```

Look at the Data profdata_…_libraries file content

- For each library seen profdata_…_libraries file contains:
  - The full path to the library (a unique identifier)
  - The “short name” for the library
  - The sum of the leaf counts of all functions belonging to this library
  - e.g.:

```
/lib64/libm.so.6        libm.so.6        272587
```

Important Definitions

- Function Path Count
  - number of samples in which that function was observed anywhere in the call stack
    - not the number of times a function was called
- Function Total Count
  - Total number of times that function was observed in the call stacks (it may be more than once per call stack e.g. for recursive calls)
- Function Leaf Count
  - number of samples in which that function was observed at the top of the call stack

Example Application

SimplifiedCalo

SimplifiedCalo Top Functions

- 5 top functions and their leaf count fractions
  - Leaf count is good quantity to look at to asses a relative impact of a function
  - based on one run with 50 events on Quad Core AMD Opteron 2389 at 2915MHz (data from profdata…_names file) (exp69,10,1)

__ieee754_log	0.076
G4HadronCrossSections::CalcScatteringCrossSections	0.071
G4PhysicsVector::Value	0.071
CLHEP::MTwistEngine::flat	0.032
__ieee754_exp	0.024

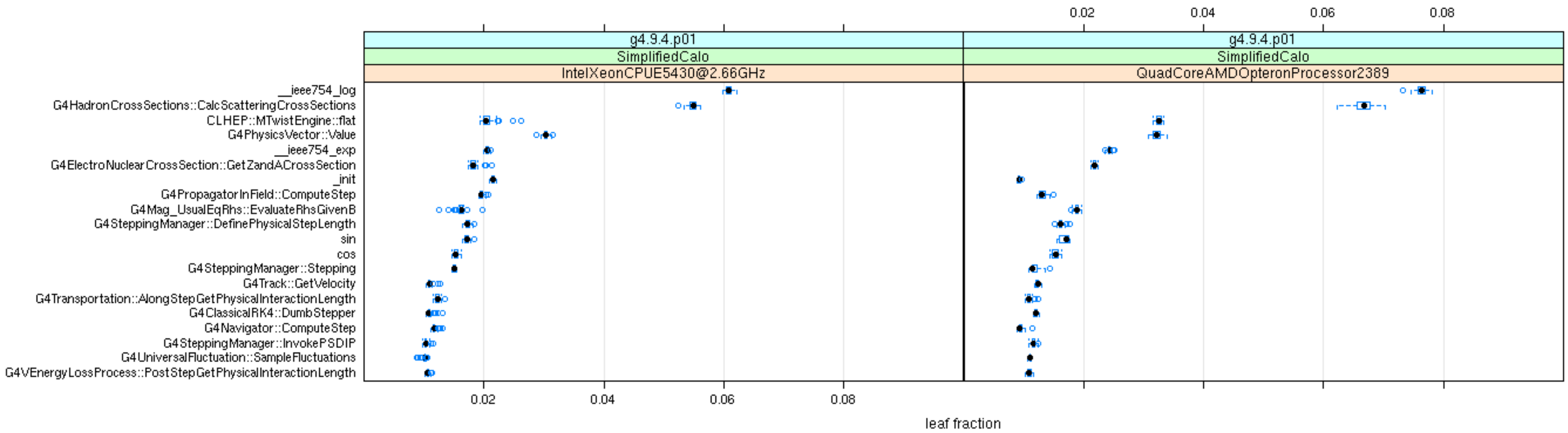
SimplifiedCalo Top Functions

- 5 top functions and their leaf count fractions
  - based on 112 runs with 50 events each on both Intel and AMD nodes

__ieee754_log	0.076
G4HadronCrossSections::CalcScatteringCrossSections	0.067
CLHEP::MTwistEngine::flat	0.033
G4PhysicsVector::Value	0.032
__ieee754_exp	0.024

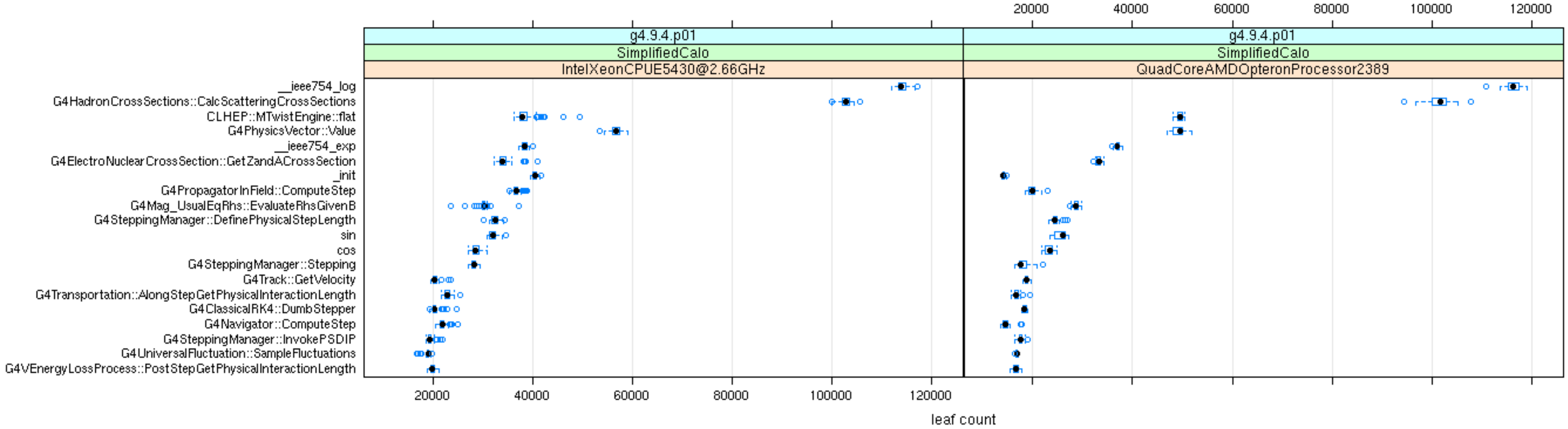
running on different processor models is the main reason for the difference compared to a single AMD run from the previous page

SimplifiedCalo Top Functions



- Leaf count fractions for top functions
  - A good quantity to look at to asses a relative impact of a function

SimplifiedCalo Top Functions



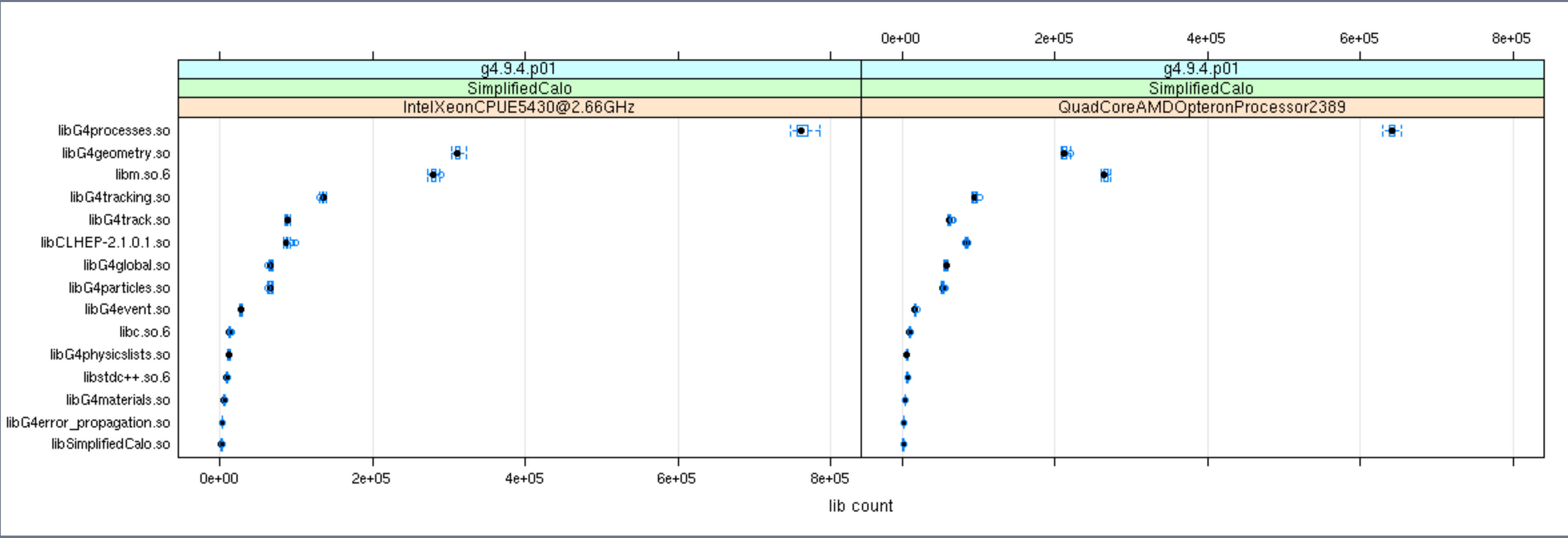
- Leaf counts for top functions
  - A good quantity to look at to see how a code modification affects a a specific function or rather a group of functions (when a relative change would not be seen in the leaf count fraction distribution)

SimplifiedCalo Top Libraries

- Top libraries for a specific run
  - on Quad Core AMD Opteron 2389 at 2915MHz (exp69,10,1)

../geant4.9.4.p01/lib/Linux-g++/libG4processes.so	libG4processes.so	652824
/lib64/libm.so.6	libm.so.6	272587
.../geant4.9.4.p01/lib/Linux-g++/libG4geometry.so	libG4geometry.so	220006
.../geant4.9.4.p01/lib/Linux-g++/libG4tracking.so	libG4tracking.so	98013
.../libCLHEP-2.1.0.1.so	libCLHEP-2.1.0.1.so	86019

SimplifiedCalo Top Libraries

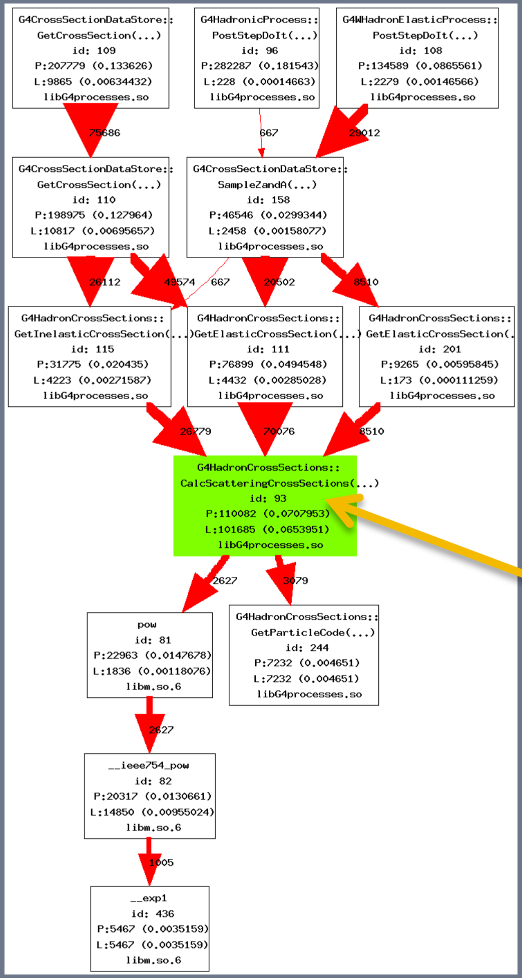


- top libraries plot
  - based on 112 runs with 50 events each

Looking at an Application in more detail - Call Graphs

- profgraph – command to produce a call graph from SimpleProfiler data
  - profgraph [opts] datafile [func-id [max-nodes-up [max-nodes-down [path-trim-count]]]
    - -n, --names
      - print function names rather than function IDs
  - Make sure to cd to the directory containing the data files obtained with profrun

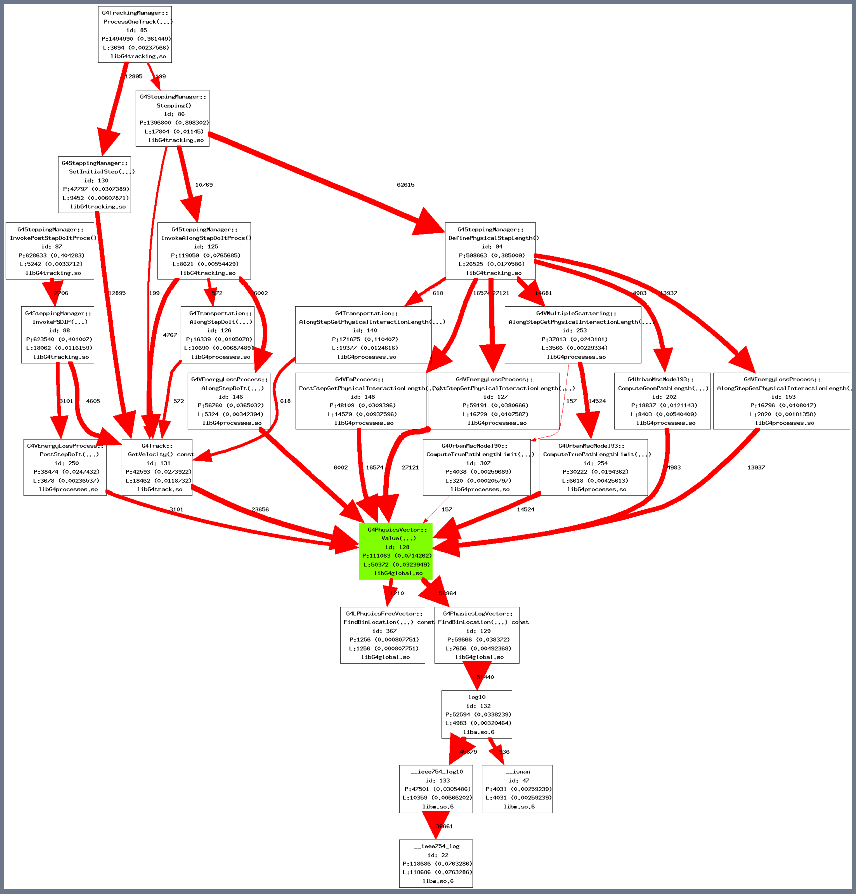
Call Graph centered on G4HadronCrossSections::CalcScatteringCrossSections in SimplifiedCalo



93

- A Geant4 function with a very significant time spent in it:
  G4HadronCrossSections::
  CalcScatteringCrossSections
  (G4DynamicParticle const*, int, int)
  - Path Count 110082 (7.1%)
  - Leaf Count 101685 (6.5%)
- profgraph –n profdata... 93 3 5 400
  - All paths with a count smaller than 400 were removed
    - this affects the edges(arrows) which are removed
    - the numbers in the boxes are unaffected

Call Graph centered on G4PhysicsVector::Value in SimplifiedCalo



- A function called by many callers, calling other functions itself, with a significant time spent in it:

  G4PhysicsVector::Value (double)

  - Path Count 111063 (7.1%)
  - Leaf Count   50372 (3.2%)
- profgraph –n profdata… 128 3 5 100
- Quad Core AMD Opteron 2389 at 2915MHz (exp69,10,1)

SimpleProfiler characteristics

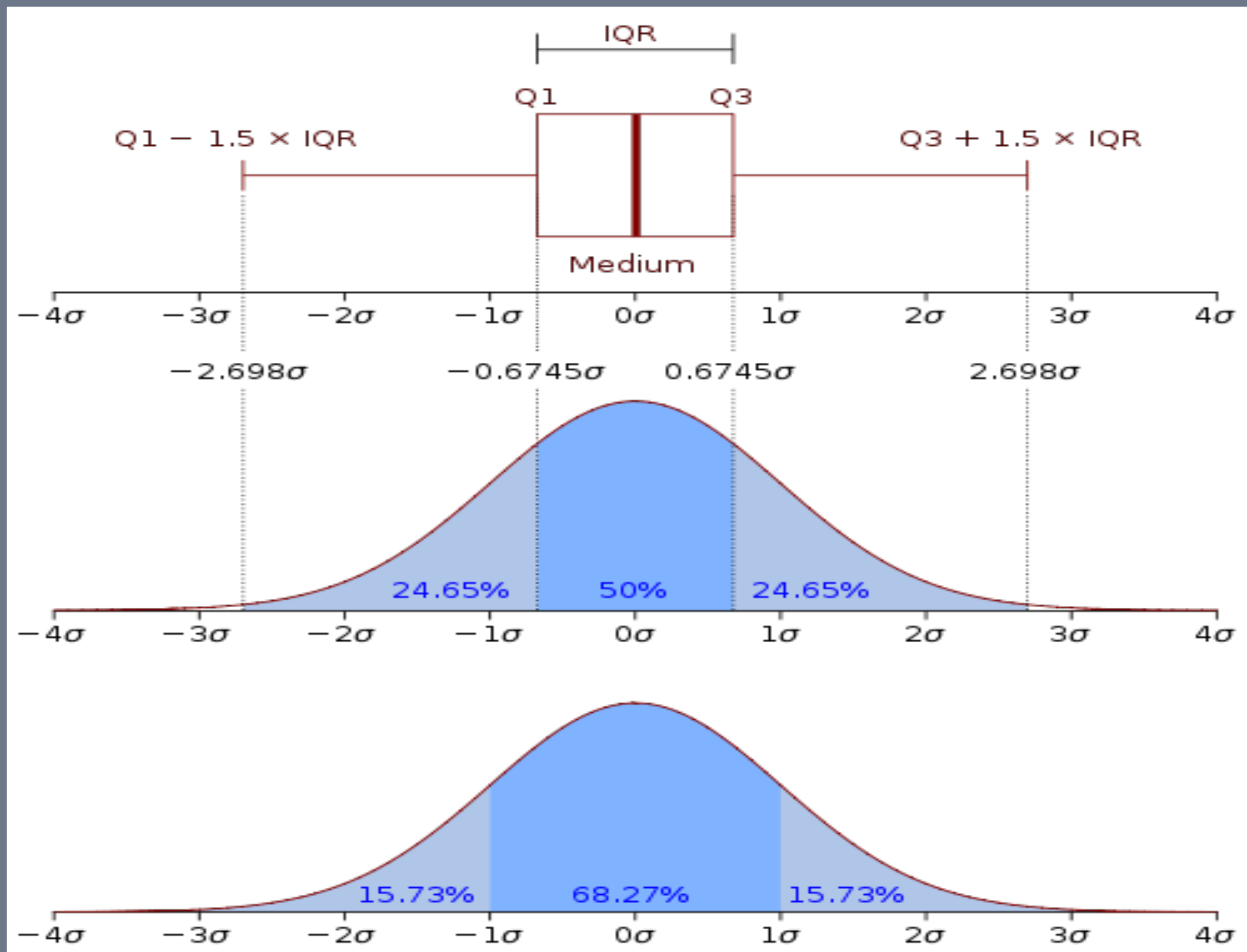
- SimpleProfiler is a sampling profiler
  - Which means that it is not 100% accurate
    - Running a test many times both increases and allows one to determine the accuracy for a given set of tests
      - Due to "other" process running on the machine
      - Due to random nature of running a process on an otherwise "idle" machine
      - The accuracy is reflected in the distribution of the measurements (see the previous plots)
    - Running an application longer (e.g. with more events) increases the number of call stacks collected and increases the accuracy as well
  - But it is very good at getting the "big picture" of the profiled application
- There are unwind errors <2%
  - Due to a sophisticated nature of optimization or kernel behavior which libunwind does not handle correctly
    - the number of unwind errors does go down when using -fno-omit-frame-pointer g++ compiler option
- Does not handle truly multithreaded programs
  - due to the very nature of the POSIX threads/signals and inability to send (here: timing) signals to a thread (and not a process)

Summary

- FAST has been available for some time and can be used to profile Geant4 (and other) applications
- It is a very good tool to obtain a "big picture" of the profiled application with a minimal overhead
  - It can be used as a starting point for further studies
    - Also with other tools like valgrind/callgrind
- Basic profiling information can be obtained with a very simple set of tools/steps:
  - SimpleProfiler - profrun - plus e.g. cat and grep
- profgraph gives an additional convenient way to look at the call graphs of the application
- Collected data in a well described format can be used for further analysis with statistical and display tools
  - See talk in parallel session 7B for more profiling results

Backup Slides

Box & Whisker Plot



from Wikipedia

Software/Hardware Versions

- SimplifiedCalo
  - As obtained from Andrea Dotti in May this year
    - minimally modified to add timing printout and to read a PYTHIA event file
    - PYTHIA 14TeV pp, 500 GeV Higgs to ZZ (all decays) input file
    - magnetic field turned on (see next page for exact parameters)
- Geant4/CLHEP
  - 9.4.p01/2.1.0.1
- Compiler
  - gcc 4.1.2 with -g -O2
- OS/Hardware
  - Scientific Linux SL release 5.4 (Boron)
  - kernel 2.6.18-238.12.1.el5
  - processors/memory
    - Intel Xeon E5430 @ 2.66GHz/16GB
    - Quad-Core AMD Opteron Processor 2389 (2.9GHz) /24GB

SimplifiedCalo Parameters

- ...
- /mygen/generator PYTHIA
- /mydet/setField 4.0 tesla
- /mydet/absorberMaterial AHCALWalloy
- /mydet/activeMaterial Scintillator
- /mydet/isCalHomogeneous 0
- /mydet/isUnitInLambda 0
- /mydet/absorberTotalLength 7000
- /mydet/calorimeterRadius 3000
- /mydet/activeLayerNumber 100
- /mydet/readoutLayerNumber 20
- /mydet/activeLayerSize 4.0
- /mydet/radiusBinSize 0.1
- /mydet/radiusBinNumber 10
- /mydet/update
- /run/beamOn 50