Summary of Geant4 Computing Performance Session

V. Daniel Elvira (Fermilab)

The G4 Computing Performance Task

The G4CPT started in Feb. 2010, not a "Task Force" but an open ended activity organized as a group with regular meetings every ~ 6-8 weeks

Charge

(a) Profiling to identify bottlenecks in Geant4 based on main stream applications. We need to discuss profiling tools, what we want to measure, metrics. EM, Geometry and hadronics are the areas more involved in CPU usage.
(b) Code reviews geared towards improving computing performance and coding practices.
(c) Establish computing performance activities with the medical and space G4 communities.
(d) Identify issues in multi-core-multithread G4. ← Not discussed beyond the first meeting – See Tuesday plenary

Parallel session: presented a few examples of G4 applications profiled with different tools, described features of profilers, showed some results, discussed plans for 2011 at the end

CMS Simulation profiled with FAST

(developed by the FNAL team: A. Baldacchi, J. Kowalkowski, M. Paterno. J. Torola)

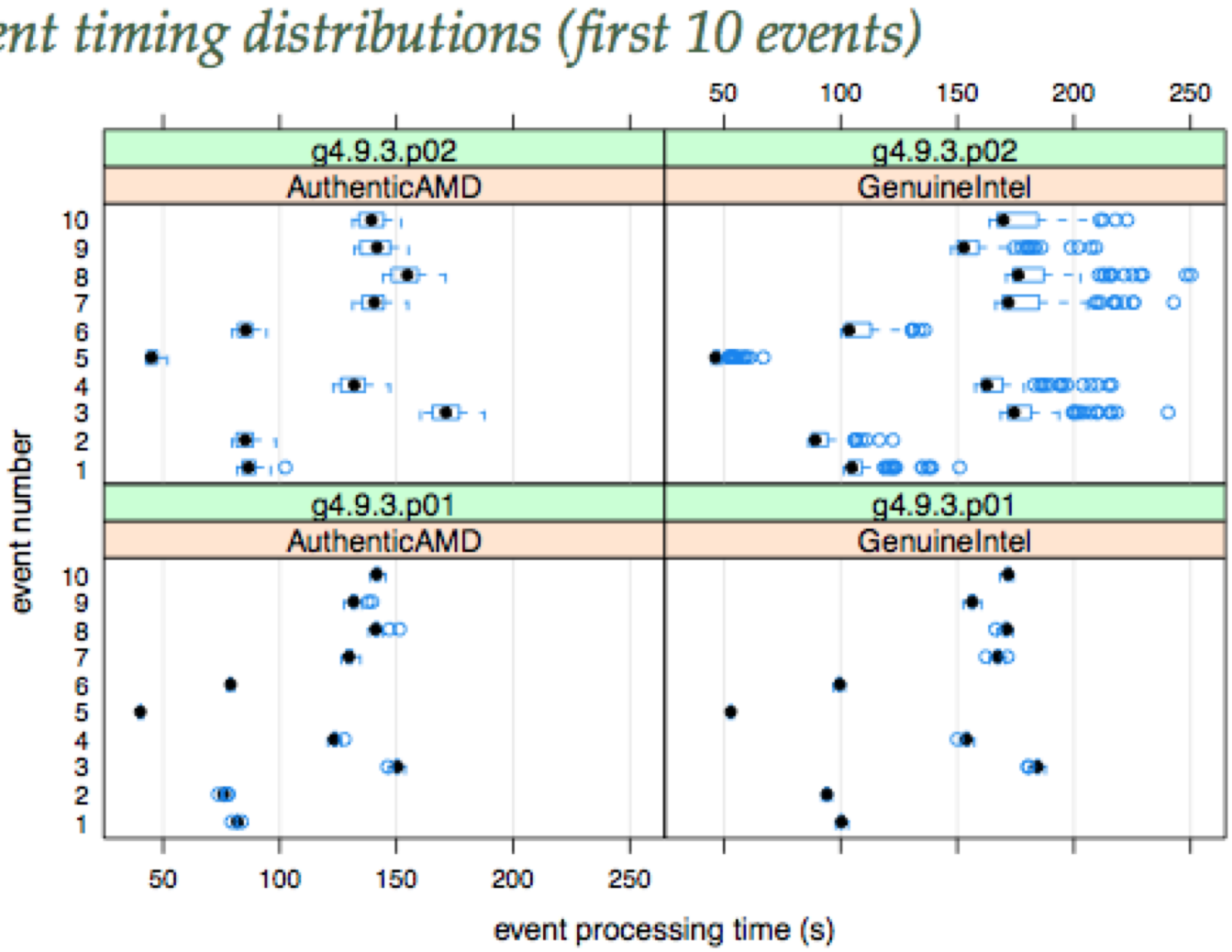
FAST includes several tools and features:

- a sampling-based profiler for collecting measurements,
- tools for extracting call path, function call and library call information from the raw data,
- tools for the collection of information about the environment in which the profiled program is run,
- tools for graphical display of call paths, and
- a documented file format in which profiling data are written, to facilitate statistical analysis of profiling data.

Analysis tools are still under development.

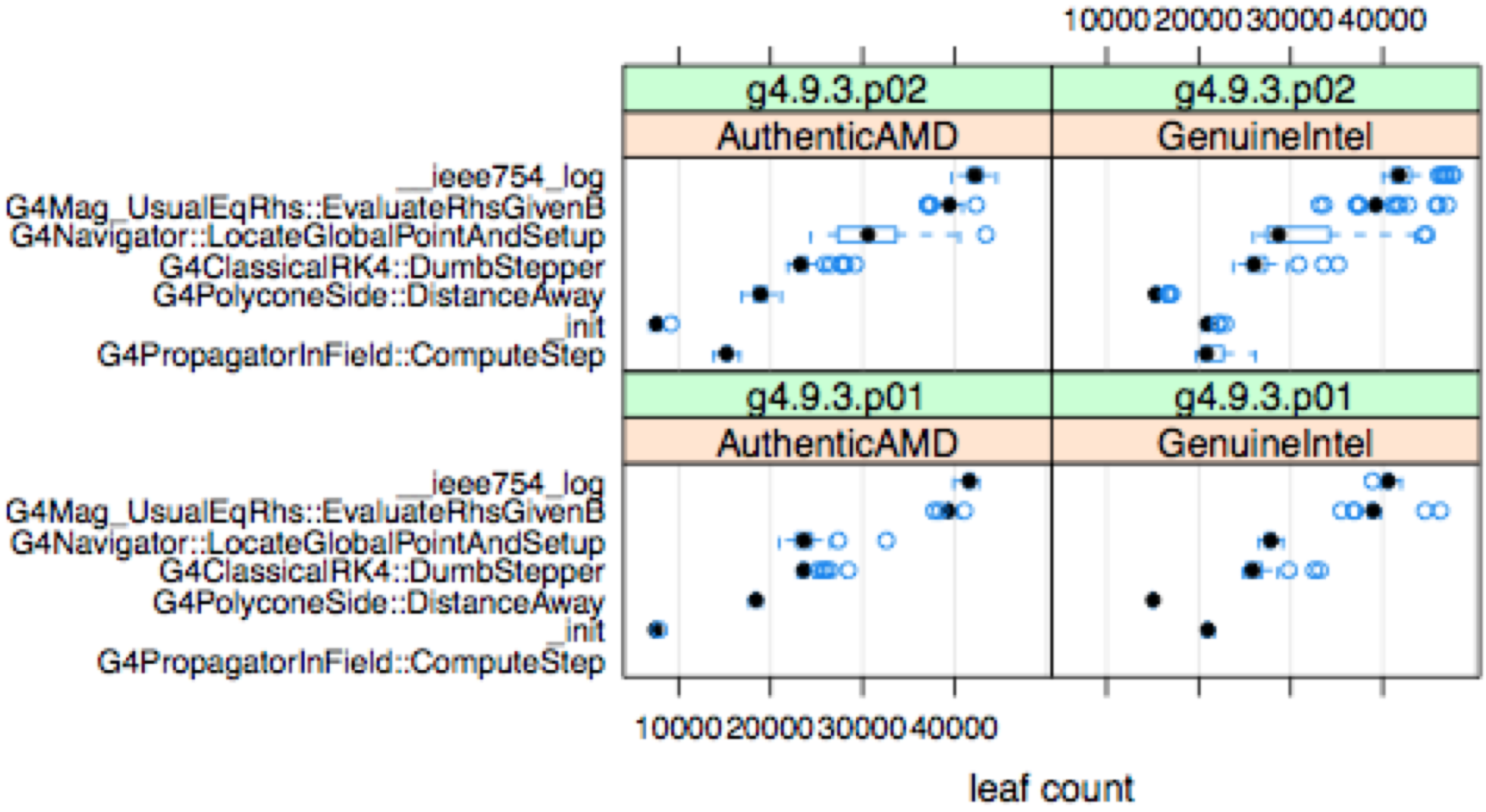
FAST can profile multi-process programs, but not multi-threaded programs.

Example 1
(CMS simulation)



Example 2 (CMS simulation)

Most time-consuming functions



Conclusion

- The FAST profiler and associated tools are now available.
- Statistically significant data samples and appropriate analysis tools are necessary for serious comparisons.
- Freely available tools aid in exploring the data.
- Tools for "standard" analysis of new releases are ready.
- ~ 85% of long-running jobs complete:
  - some failures in `libunwind`,
  - some failures in our postprocessing of data,
  - some failures in profiled application,
  - some failures due to cluster environment issues.
- Almost "turnkey" operation now possible.
- Web-publishing of results still must be done manually.
- Validation of results is not automated; we do not want to publish any mischaracterization of Geant4 or CMS code.

`https://cdcvs.fnal.gov/redmine/projects/fast`

CMS (ATLAS) profiled with Perfmon (PTU)

How PTU and perfmon work

(David Liventhal-Intel, D. Kruse-CERN, J. Apostolakis-CERN)

- They use the hardware performance counters available in modern CPUs to count:
  - CPU cycles
  - cycles when the CPU is not doing useful work
  - cache misses (for each level) and other 'stalls'
  - And many other harder-to-understand 'events'.
- Each CPU microarchitecture has different
  - Number of counters (e.g. 4 fixed, 12 'programmable' on Nehalem)
  - New types of counters (many in common, some old ones are deleted)

Key quantities measured on *Nehalem*

- **BASIC STATS:** Total Cycles, Instructions Retired, CPI;
- **INSTRUCTION STATS:** Branches, Loads, Stores, Other, Packed UOPS;
- **BASIC STALL STATS:** Show often CPU cannot get/do work
  - Stalled Cycles, % of Total Cycles
- **INSTRUCTION "intensity":**
  - # of Instructions per Call, Instruction Starvation
- **FLOATING POINT EXCEPTIONS:**
  - % of Total Cycles spent handling FP exceptions;
- **LOAD OPS STALLS:** Why Loads stalled
  - L2 Hit, L3 Miss -> Local DRAM Hit
- **Misses of TLBs (Translation Lookaside Buffers)**
- **DIVISION & SQUAREROOT STALLS:** Cycles spent during DIV & SQRT Ops;
- **L2 IFETCH MISSES:** Where were the instructions found
- **BRANCHES, CALLS & RETS:** Total Branch Instructions Executed, % of Mispredicted Branches, types of calls.

Current results

- Cycles per instruction: 1.5 (ideal < 1, ok ~ 1.0)
- Too many branches
  - A branch every 40 instructions
  - Little time for CPU to do work, between loading and saving registers
- High cost for shared library calls
  - Indirect calls due to position independent code
  - Each call to another library typically requires 3 branches
- Real cost also for virtual methods
  - CPU cannot predict target address of branch

Daniele's Results for G4 *FullCMS* simulation

Run on *Nehalem*

- The results for the FullCMS simulation (10 events) are available at:

  http://dkruse.web.cern.ch/dkruse/2010_09_21_Geant4_FullCMS_slc5_amd64/

- The simulation ran on a single-processor quad-core *Nehalem* machine with *HyperThreading*, *TurboBoost*, *C-States* ***all disabled***

- Same random seed for each run to have identical runs!

CMS CPU/Mem Profiling with IgProf

G4 ESTEC Meeting

Peter Elmer/Princeton University (CMS experiment/CERN)

[For Lassi Tuura/Giulio Eulisse (FNAL) and Matti Kortelainen (Helsinki Institute of Physics]

In recent months, we (CMS) have mostly been doing things other than G4 performance studies, however there has been some progress on two tools we have been using for our studies (igprof and ptuview). The goal was to make a proper presentation at CHEP, however I'll give a quick "preview" here.

The main things which have happened in the past 6 months or so are:

- Lassi has finished his investigations into 64bit stack traces and problems with libunwind and has fed his patches back to libunwind. Resolving some problems will require switching to gcc45 and (for most pratical cases) may require rebuilding libc/libm to get complete tracebacks.
- igprof is now available and built (even within CMS) as a standalone external (see recipe below)
- As part of the learning process for PTU (see John's talk earlier in this session), we put together a web-based viewer for the data from PTU (normally available only through the eclipse-based PTU tool).

ptview demo

Hotspot view - function

View all events

◀ All program cycles	1e+11	
◀ Executing	42.57 %	
▶ Port utilization		
Microcode sequencer		1.82 %
Wasted work (of uops)		21.48 %
Mispredicted branches (of all branches)		3.73 %
Indirect branches (of all branches)		7.46 %
◀ Stalled	57.43 %	
▶ Memory latency total		
Cycles the divider is busy		3.20 %
▶ Load ordering stalls		
▶ Bandwidth		
▶ Front End total		

CPU_CLK_UNHALTED.THREAD	Executing	UOPS_DECODED.MS_CYCLES_ACTIVE		UOPS_ISSUED.ANY		BR_MISP_RETIRED.ALL_BRANCHES		BR_INST_EXEC.INDIRECT_NON	
1.6e+10	59.43 %	0	0.00 %	2.7e+10	24.74 %	2.3e+08	5.29 %	2.7e+08	8.21 %
1.4e+10	34.20 %	1.7e+08	1.20 %	1.8e+10	38.29 %	7.8e+07	3.58 %	2.2e+08	10.59 %
1.2e+10	71.42 %	0	0.00 %	2.5e+10	47.60 %	1.1e+08	3.28 %	0	0.00 %
1.1e+10	22.63 %	5.6e+07	0.49 %	7.4e+09	42.70 %	2.6e+07	2.58 %	0	0.00 %
5.5e+09	50.25 %	2e+08	3.55 %	7e+09	-19.94 %	2e+07	1.09 %	2e+08	11.53 %

+ … + function + module + library

Conclusions and Action Items

On performance tests:

- Define a set of profiling tests (with metrics) to benchmark a given version of G4 and identify opportunities to improve performance. CMS (full showers option) is one of the best candidates.
- Decide target (ref, candidate, public/patch) releases to profile, frequency. Assign responsibilities.
- Ideally we would like to run fully automated applications and store the information in a database to track history. We should start with the currently available semi-automated versions that produce web browsable outputs.

Conclusions and Action Items (Cont.)

On profilers:

- IgProf (NE-CMS) and FAST (FNAL) should be used to run performance tests.
- Perfmon (HP) and PTU (Intel) study interaction between the code and the hardware. We should experiment and learn how to use them to detect performance bottlenecks.

Conclusions and Action Items (Cont.)

On code reviews:

- Bertini Cascade. M. Kelsey (SLAC)'s changes in memory handling reduced memory churn by 5-8% in CMS/ATLAS simulations … with a significant increase in the CPU time though.
- Message from ATLAS/CMS: improve EM package performance. We need to write the charge for the team to do the code review. K. Genser (FNAL), who else?
- What packages are next? Hadronic cross-sections, pre-compound/de-excitation?

Conclusions and Action Items (Cont.)

On collaboration with users:

- Space : geometry and physics are presently of greater concern than absolute computational efficiency (speed/memory).
- Medical: there are memory issues due to large number of materials and voxels, speed issues due to the need to iterate for treatment plans (~10 min).
- ATLAS: recent ~15% CPU time improvement without cost in physics with Nystrom stepper.
- CMS: moving to 64bits platform/gcc45. Maintain good communication on profiling activities.

Backup Slides

List of Top Problems to Investigate

1. Memory Allocation
    * Navigation (G. Cosmo working on a fix - ATLAS, CMS testing)
    * Bertini (Mike Kelsey working on mem/speed improvements)
2. EM Physics Package
    * Optimization of parameters in applications
    * Revisit physics algorithms in Geant4 code: optimizations, approximations
    * Multiple scattering
    * Code review
3. Navigation speed and memory use in Voxel geometries and when handling large numbers of materials (brought up by the medical community)
4. Ion-ion inelastic models speed and memory use (medical).
5. Propagation in Magnetic Fields
    * Code review (done - no low hanging fruit from the programming side)
    * Testing/validation/profiling new steppers (ATLAS is testing Nystrom)
6. Hadronic cross-sections
    * Code review
7. Precompound/de-excitation
    * Code optimization. Many log/power functions are called. Many classes.

Challenge: define, staff, initiate remaining projects