




TAU Performance System®

Examples of Performance Analysis of Geant4 Electromagnetic Processes

Boyana Norris

Associate Professor
Computer and Information Science
University of Oregon



Talk outline

- ❑ Overview of measurement methods
- ❑ TAU introduction
- ❑ Example use of TAU: Geant4 EM review
- ❑ Ongoing work and future directions

Performance measurement methodologies

- ❑ Ideally we want a low-overhead method that gives sufficiently accurate and detailed information
- ❑ Sampling based (a.k.a. statistical sampling)
 - Timer or hardware interrupts are used to stop a thread of execution and interrogate the program counter, callstack, and other context **at runtime**
 - **Nonintrusive** – source or binary not modified in any way
 - Usually (but not always) **low** overhead, can be **imprecise**
 - Can capture **fine-grained** behavior (small functions called frequently)
- ❑ Probe-based (a.k.a. direct)
 - **Intrusive** -- Code inserted into program (manually or through automated instrumentation)
 - Can have very **high** overhead
 - Can have very **low** overhead
 - **Precise** for most measurements except **time**

Performance measurement tools

❑ Measurement and analysis tools (portable)

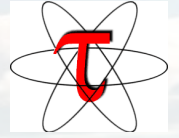
- Sampling: TAU, HPCToolkit, Fast, IgProf
- Probe-based with binary instrumentation: Pardyn, TAU, HPCToolkit
- Probe-based with source instrumentation: TAU, Scalasca

❑ Meta-tools

- Open|SpeedShop (based on Dyninst, libmonitor, PAPI)
- PerfExpert (TACC)
- Autoperf (UO)

❑ Wikipedia – apparently hosting a quiet war among tools

TAU Performance System[®]



- ❑ Tuning and Analysis Utilities (19+ year project)
- ❑ Comprehensive performance profiling and tracing
 - Integrated, scalable, flexible, portable
 - Targets all parallel programming/execution paradigms
- ❑ Integrated performance toolkit
 - Instrumentation, measurement, analysis, visualization
 - Widely-ported performance profiling / tracing system
 - Performance data management and data mining
 - Open source (BSD-style license)
- ❑ Easy to integrate in application frameworks

<http://tau.uoregon.edu>

TAU direct observation: Events

□ Event types

- Interval events (begin/end events)
 - ◆ Measures exclusive & inclusive durations between events
 - ◆ Metrics monotonically increase
- Atomic events (trigger with data value)
 - ◆ Used to capture performance data state
 - ◆ Shows extent of variation of triggered values (min/max/mean)

□ Code events

- Routines, classes, templates
- Statement-level blocks, loops

TAU sampling

- ❑ When an interrupt occurs, a TAU handler routine is activated and performs the following steps:
 - Determine program counter where the program was executing
 - Check if executing TAU code and process sample accordingly
 - Unwind the calling stack a certain *unwind degree* (configurable)
- ❑ The current program counter context is provided by the signaling mechanism
- ❑ If call stack information is not required, a *flat profile* of the time spent for each line of code is recorded

User options

❑ Probing:

- select events,
- enable event path profiling for a specified depth

❑ Sampling:

- set timer interrupt period,
- set depth call stack unwinding

EM review performance study

- ❑ Focus on
 - G4VProcess and G4VEMModel class hierarchies
 - G4PhysicsVector and G4Physics2DVector container classes and their descendants
- ❑ Biggest challenges (no tool could do all this when we started)
 - OO-centric analysis
 - ◆ View and analyze performance based on classes ✓
 - ◆ Virtual functions – attribution to concrete implementations ✓
 - Inlined functions ✓
 - Accurately measure small functions that are called a lot – with limited overhead (still a problem) ✗
 - Ability to do analysis only on portions of the code of interest (not necessarily the most time-consuming parts) ✓✗

Direct instrumentation options in TAU

- ❑ Source Code Instrumentation
 - Automatic instrumentation using pre-processor based on static analysis of source code (PDT), creating an instrumented copy
 - Compiler generates instrumented object code
 - Manual instrumentation
- ❑ Library-level instrumentation
 - Statically or dynamically linked wrapper libraries: MPI, I/O, memory, ...
 - Wrapping external libraries where source is not available
- ❑ Runtime pre-loading and interception of library calls
- ❑ Binary Code instrumentation
 - Rewrite the binary, runtime instrumentation
- ❑ Virtual Machine, Interpreter, OS level instrumentation

EM review experiments

- ❑ Performed automated instrumentation of Geant4 v. 10.0
 - Sampling
 - Complete: all functions
 - Selective: only explicitly specified functions
- ❑ HW: Intel Xeon X5650 2.67GHz 12-core dual processor nodes with 70GB of DRAM (per node). Compiler: gcc , RelWithDebInfo
- ❑ Collected two types of profiles
 - Flat: per-function information without any context
 - Callpath: per-function information for each calling context
- ❑ Types of data collected
 - Wall-clock time
 - CPU hardware performance counters, e.g., instruction counts, cache misses
- ❑ Results: TAUdb database: geant4 at taudb.nic.uoregon.edu

Hardware counters

Category	Metric	Description
Time	P.WALL.CLOCK.TIME	Wall-clock time
Memory	L1.DCM	Level1 data cache misses
Memory	L2.DCM	Level2 data cache misses
Memory	L2.DCA	Level2 total number of Level2 data accesses
Memory	L1.ICM	Level1 instruction cache misses
Memory	L2.ICM	Level2 instruction cache misses
Memory	L2.TCM	Total Level2 cache misses (data and instructions)
Memory	L3.TCA	Total Level3 cache accesses
Memory	LD.INS	Load instructions
Memory	TLB.DM	Data translation lookaside buffer misses
Memory	TLB.IM	Instruction translation lookaside buffer misses
Memory	TLB.TL	Total translation lookaside buffer misses
Jumps	BR.MSP	Number of mispredicted branches
Jumps	BR.INS	Total number of branch instructions
Compute	DP.OPS	Number of double precision floating-point operations
Compute	FP.OPS	Number of single precision floating-point operations
General	VEC.DP	Vector/SIMD instructions executed (double precision)
General	VEC.SP	Vector/SIMD instructions executed (single precision)
General	TOT.CYC	Total cycles
General	TOT.IIS	Total instructions issued
General	TOT.INS	Total instructions executed
General	RES.STL	Total resource stalls (any reason)

Tool overheads (sampling only)

❑ SimplifiedCalo

○ TAU and HPCToolkit:

- ◆ callpaths
- ◆ sampling period=10,000μs

Tool	HEP Event, 50 events	HEP Event, pythia_pp_ttbar (1000 events)
No profiling	241	458
FAST	237	452
HPCToolkit	245	470
TAU sampling	241	459

Probe-based vs Sampling-based

- ❑ Overhead depends on the code structure
 - In some SimplifiedCalo experiments, probe-based is between 2x and 200x slower than sampling-based.
 - In other codes, e.g., some of the NAS Parallel Benchmarks (right), probe-based can be faster.

BT	<i>Clean</i>	<i>Sampling</i>	<i>Select</i>	<i>Hybrid</i>	<i>Full</i>
1	246.20	248.71	250.61	244.97	2355.43
2	125.24	125.38	125.82	124.95	1644.89
4	64.90	64.67	65.26	64.49	875.45
8	37.65	37.77	37.99	38.19	1033.80
12	30.15	31.37	30.43	30.48	1002.12

EP	<i>Clean</i>	<i>Sampling</i>	<i>Select</i>	<i>Hybrid</i>	<i>Full</i>
1	61.46	64.40	61.43	63.17	61.89
2	30.96	31.81	30.88	31.76	31.03
4	15.53	16.01	15.51	15.90	15.58
8	8.10	8.34	8.11	8.33	8.15
12	5.44	5.54	5.43	5.55	5.42

FT	<i>Clean</i>	<i>Sampling</i>	<i>Select</i>	<i>Hybrid</i>	<i>Full</i>
1	51.68	51.24	52.62	51.43	56.46
2	26.62	26.68	26.83	26.63	29.44
4	14.00	14.06	14.01	13.95	15.27
8	9.42	9.57	9.42	9.50	9.96
12	8.76	9.05	9.09	9.12	9.76

LU-HP	<i>Clean</i>	<i>Sampling</i>	<i>Select</i>	<i>Hybrid</i>	<i>Full</i>
1	248.02	278.16	253.27	280.81	251.80
2	121.24	125.90	125.75	125.57	127.26
4	64.22	68.47	69.24	68.70	69.64
8	37.81	42.43	43.52	44.05	45.11
12	29.92	34.30	36.26	36.16	37.61

How to compile Geant4

❑ For **probe-based** measurement:

- For full instrumentation, simply configure with:

```
cmake -DCMAKE_CXX_COMPILER=tau_cxx.sh \  
      -DCMAKE_CC_COMPILER=tau_cc.sh ...
```

- For selective instrumentation, create a **select.tau** file, then define the **TAU_OPTIONS** environment variable accordingly, for example:

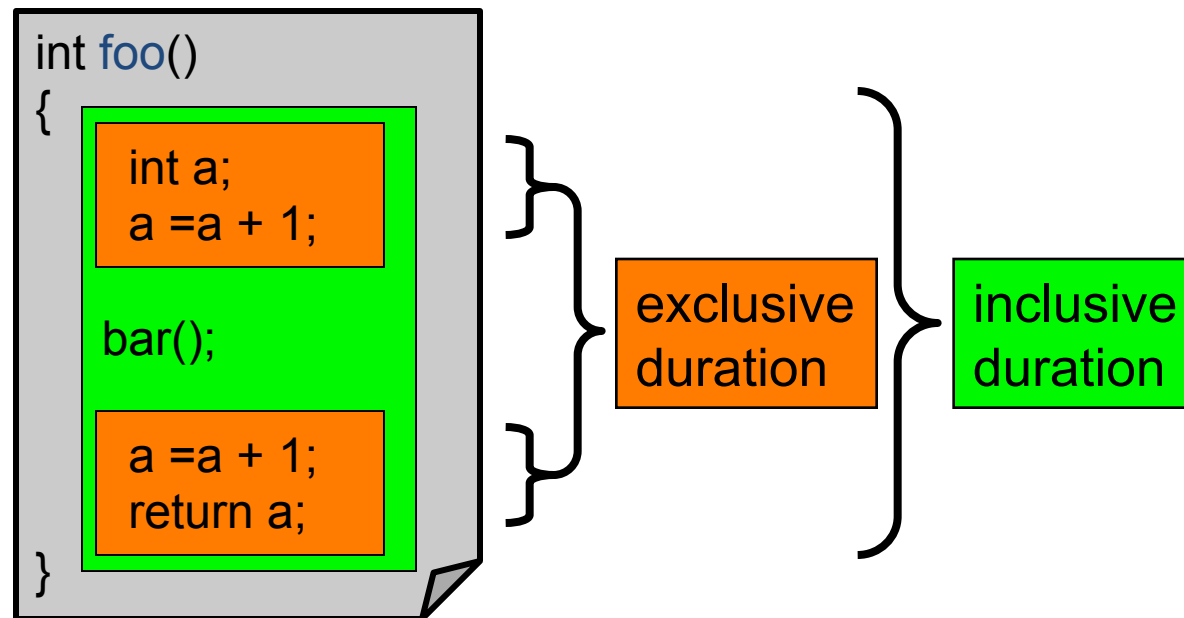
```
export TAU_OPTIONS='--optVerbose --optRevert \  
--optTauSelectFile="$HOME/Geant4/select.tau"'
```

❑ For **sampling-based** measurement

- Configure and compile as usual (ensure **-g** is used)

Inclusive vs. exclusive measurements

- ❑ Performance with respect to code regions
- ❑ Exclusive measurements for region only
- ❑ Inclusive measurements includes child regions



Summary of EM Review results

Class name Function name	Performance hardware counter measurements (%Tot)				
	Stalls Inc. (Excl.)	L2 TCM Inc. (Excl.)	TLB DM Inc. (Excl.)	TLB IM Inc. (Excl.)	Time Inc. (Excl.)
G4PhysicsVector Value	8.46 (8.46)	1.67 (1.67)	4.13 (4.13)	1.46 (1.46)	2.68 (2.68)
G4PhysicsLogVector FindBinLocation	-	-	-	-	-
G4VProcess SubtractNumberOfInteractionLengthLeft	-	-	-	-	-
G4VEmProcess PostStepGetPhysicalInteractionLength	4.07 (1.49)	2.20 (1.62)	3.09 (1.95)	1.93 (1.45)	2.51 (1.82)
GetCurrentLambda*	<0.01	<0.01	<0.01	<0.01	<0.01
PostStepDoIt	4.56 (0.68)	2.93 (1.31)	1.83 (0.64)	2.57 (1.16)	1.68 (0.72)
G4VMultipleScattering AlongStepDoIt*	2.1 (0.2)	1.9 (0.4)	-	1.7 (0.3)	1.8 (0.2)
AlongStepGetPhysicalInteractionLength*	4.8 (0.2)	4.2 (0.4)	-	2.6 (0.2)	2.3 (0.3)
G4VEnergyLossProcess PostStepGetPhysicalInteractionLength*	1.8 (0.4)	1.9 (1.0)	-	0.6 (0.5)	1.2 (0.7)
AlongStepDoIt*	2.8 (0.3)	1.0 (0.4)	-	0.8 (0.3)	4.2 (0.3)
GetLambdaForScaledEnergy	-	-	-	-	-
AlongStepGetPhysicalInteractionLength*	0.5 (0.2)	0.5 (0.2)	-	<0.1	0.4 (0.3)
G4UrbanMscModel ComputeGeomPathLength	1.04 (0.61)	0.56 (0.48)	0.78 (0.55)	1.93 (1.79)	0.62 (0.47)
ComputeTruePathLengthLimit	3.67 (0.99)	2.90 (1.99)	2.52 (0.91)	6.02 (3.78)	2.37 (1.19)
SampleCosineTheta	1.84 (1.78)	0.61 (0.58)	0.36 (0.21)	0.49 (0.48)	0.31 (0.21)
SampleScattering	3.20 (1.22)	1.32 (0.67)	0.77 (0.25)	1.18 (0.65)	0.73 (0.31)
Classes and functions added during the review					
G4Physics2DVector Value	0.40 (0.39)	0.36 (0.32)	0.40 (0.26)	0.03 (0.03)	0.34 (0.24)
FindBinLocation	<0.01	0.05 (0.05)	0.13 (0.13)	<0.01	0.01 (0.01)
G4Poisson*	<0.01	0.22	-	<0.01	0.33
G4UniversalFluctuation SampleFluctuations	8.10 (8.10)	0.53 (0.53)	0.18 (0.18)	1.10 (1.10)	0.13 (0.13)

FLOP-inefficient functions

- Functions that do a lot of floating-point computations and also are responsible for a significant fraction of the stall cycles (for any reason):

$$FP_{ineff} = w \frac{Ins_{fp}}{Ins_{total}} \frac{Cycles_{stall}}{Cycles_{total}}$$

- w is the function cycles (exclusive) fraction of total application cycles
- Ins_{fp} is the number of floating-point instructions
- Ins_{total} is the total number of instructions.
- $Cycles_{stall}$ are the function's stall cycles
- $Cycles_{total}$ are the total stall cycles

FLOP-inefficient functions (cont.)

Class name Function name	FP_{ineff} Value (Excl.)	Number of Calls	% of Total
G4UniversalFluctuation			60
SampleFluctuations	1.25e+00	1.56e+07	60
G4UrbanMscModel			16
SampleCosineTheta	1.98e-01	1.44e+07	9.5
SampleScattering	9.31e-02	1.44e+07	4.5
ComputeTruePathLengthLimit	2.68e-02	4.57e+07	1.3
G4PhysicsVector			11.3
Value	2.35e-01	3.71e+08	11.3
G4VEmProcess			2.2
PostStepGetPhysicalInteractionLength	2.99e-02	1.61e+08	1.4
G4VEnergyLossProcess			1.8
PostStepGetPhysicalInteractionLength	3.35e-02	9.38e+07	1.6

Lightweight functions

- ❑ Lightweight functions: relatively few floating-point instructions per call but take a significant fraction ($> 0.5\%$) of the overall execution time.

Class Function	Total Instr.	Number of Calls	FP Instr. per Call	% of Total Time
<code>G4VEmProcess</code>	1.56e+10			2.97
<code>PostStepGetPhysicalInteractionLength</code>	1.14e+10	1.61e+08	70.64	1.82
<code>PostStepDoIt</code>	3.89e+09	3.05e+07	127.44	0.72

- The FLOP-inefficient

`G4VEmProcess::PostStepGetPhysicalInteractionLength` is also among the lightweight functions because each call does relatively few operations (but there are many calls).

- ❑ Instrumentation-based data (accurate function call counts)

PostStepDoIt and PostStepGetPhysicalInteractionLength

❑ Which types of objects invoke them most?

○ PostStepDoIt (`G4VEmProcess` classes)

Class name	Count	% of total
G4ComptonScattering	20668199	14.91%
G4PhotoElectricEffect	8916255	6.43%
G4GammaConversion	852255	0.62%
G4eplusAnnihilation	79927	0.06%
G4CoulombScattering	999	0.0007%

○ PostStepDoIt (outside `G4VEmProcess` hierarchy)

Class name	Count	% of total
G4Transportation	83117473	59.97%
G4eMultipleScattering	16078703	11.60%
G4eBremsstrahlung	8476856	6.12%
G4eIonisation	450700	0.32%

PostStepGetPhysicalInteractionLength

- Called in `G4VProcess::PostStepGPIL`

Class name	Count	% of total
G4CoulombScattering	4.692927e+07	29.10%
G4GammaConversion	3.618821e+07	22.44%
G4PhotoElectricEffect	3.618821e+07	22.44%
G4ComptonScattering	3.618821e+07	22.44%
G4eplusAnnihilation	5.753727e+06	3.57%

Impact of changes: 10 vs 10xyd

- ❑ $Improvement = (T_{v10} - T_{xyd}) / T_{v10}$
 - positive: improvement
 - negative: degradation

- ❑ The measurements represent *exclusive* values (only for the function indicated) and were obtained with low-overhead sampling using a period of 1,000 microseconds.

- ❑ Better approach would be to use hybrid sampling (but it was too new and review was done)

Impact of changes: 10 vs 10xyd

Time (milliseconds); exclusive; sampling; includes initialization					Total L2 misses; Exclusive		
%total v10	%total xyd	Time (μs) v10	Time (μs) xyd	% Improv.	L2.TCM v10	L2.TCM xyd	% Improv.
Total		226666	232211	-2.45%	2.38E+09	2.25E+09	5.37%
UNRESOLVED /lib64/libm-2.12.so							
6.2%	5.4%	13940	12456	10.65%	1.47E+08	1.22E+08	16.83%
4Log.hhG4Log [G]							
4.4%	4.3%	10052	9886	1.65%	1.06E+08	9.71E+07	8.67%
G4PhysicsVector::SplineInterpolation()							
4.4%	4.3%	9967	9963	0.00%	1.05E+08	9.77E+07	7.09%
G4SteppingManager::DefinePhysicalStepLength()							
2.4%	2.3%	5422	5339	1.53%	5.70E+07	5.22E+07	8.46%
CLHEP::Hep3Vector::operator=()							
2.1%	2.2%	4755	4993	-0.50%	4.98E+07	4.86E+07	2.39%
4Exp.hhG4Exp [G]							
2.0%	2.0%	4597	4560	0.80%	4.89E+07	4.52E+07	7.55%
G4UniversalFluctuation::SampleFluctuations()							
1.9%	1.9%	4417	4395	0.49%	4.70E+07	4.37E+07	6.98%
CLHEP::MTwistEngine::flat()							
1.7%	2.0%	3876	4575	-18.03%	4.10E+07	4.50E+07	-9.91%
G4PhysicsVector::FindBinLocation()							
1.7%	1.8%	3766	4240	-12.59%	3.97E+07	4.15E+07	-4.56%
G4Transportation::AlongStepGetPhysicalInteractionLength()							
1.4%	0.9%	3255	2191	32.69%	3.42E+07	2.13E+07	37.65%
G4Navigator::ComputeStep()							
1.4%	1.2%	3230	2870	11.15%	3.38E+07	2.80E+07	17.17%

Impact of changes: 10 vs 10xyd (cont.)

Time (milliseconds); exclusive; sampling; includes initialization					Total L2 misses; Exclusive		
%total v10	%total xyd	Time (μ s) v10	Time (μ s) xyd	% Improv.	L2-TCM v10	L2-TCM xyd	% Improv.
Total		226666	232211	-2.45%	2.38E+09	2.25E+09	5.37%
G4ParticleChange::CheckIt()							
1.3%	1.3%	2992	3072	-2.67%	3.14E+07	2.99E+07	4.63%
CLHEP::Hep3Vector::rotateUz()							
1.3%	1.3%	2945	2916	0.98%	3.11E+07	2.87E+07	7.94%
G4NormalNavigation::ComputeStep()							
1.2%	1.1%	2719	2576	5.26%	2.87E+07	2.52E+07	12.23%
4Log.hhget_log_px [G]							
1.2%	1.2%	2617	2693	-2.90%	2.77E+07	2.65E+07	4.30%
G4VEmProcess::PostStepGetPhysicalInteractionLength()							
1.1%	1.4%	2515	3225	-28.23%	2.64E+07	3.15E+07	-19.26%
G4PhysicsVector::Value()							
1.0%	1.2%	2353	2747	-16.74%	2.48E+07	2.67E+07	-7.78%
G4SteppingManager::InvokeAlongStepDoItProcs()							
1.0%	1.2%	2309	2672	-15.72%	2.42E+07	2.59E+07	-7.19%
G4SteppingManager::Stepping()							
1.0%	1.1%	2288	2495	-9.05%	2.40E+07	2.44E+07	-1.50%
G4UrbanMscModel::ComputeGeomPathLength()							
1.0%	0.8%	2287	1887	17.49%	2.40E+07	1.84E+07	23.16%
G4VEnergyLossProcess::PostStepGetPhysicalInteractionLength()							
1.0%	1.0%	2251	2228	1.02%	2.37E+07	2.17E+07	8.32%
G4AffineTransform::TransformPoint()							
1.0%	0.9 %	2238	2154	3.75%	2.36E+07	2.09E+07	11.33%

Different versions (10.0 vs 10.02.b01): -6.28%

SimplifiedCalo, Standard, 50GeV, 500 events

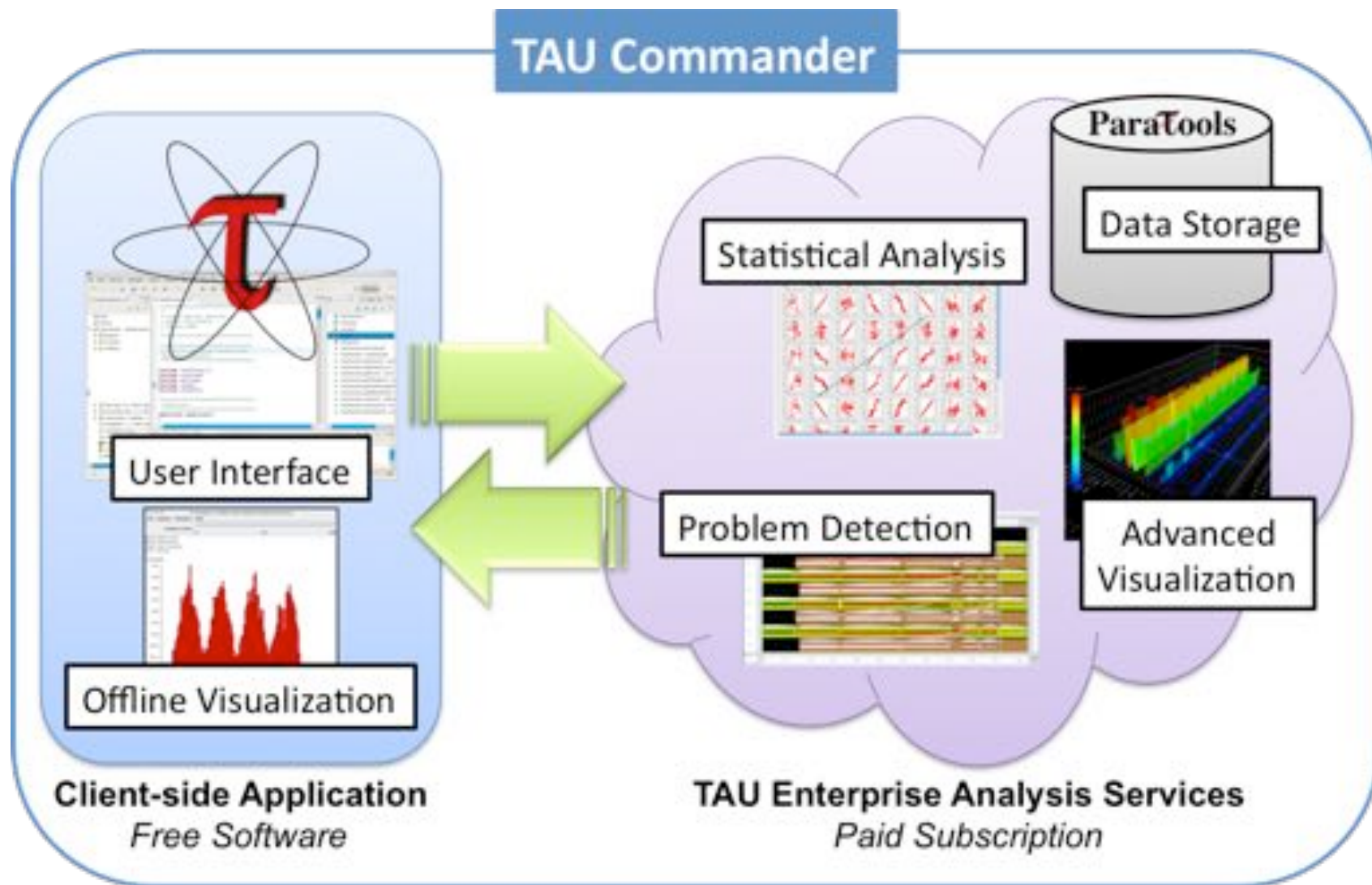
Function	Difference	%Total in.	% Change
G4PhysicsVector::Spin	6.66E+05	4.57E+00	6.68E-02
G4UniversalFluctuation	-1.87E+06	1.89E+00	-4.24E-01
G4PhysicsVector::FindBinLocation(double)	-3.52E+05	1.66E+00	-9.34E-02
G4PhysicsVector::Value(double, unsigned long&)	-3.90E+05	1.25E+00	-1.66E-01
G4VEmProcess::PostStepGetPhysicalInteraction	-1.01E+06	1.19E+00	-4.03E-01
G4PhysicsVector::FindBin(double, unsigned long)	-1.55E+05	9.27E-01	-9.41E-02
G4UrbanMscModel::ComputeGeomPathLength(double)	-2.35E+05	8.75E-01	-1.03E-01
G4UrbanMscModel::SampleCosineTheta(double, doub	1.68E+05	6.38E-01	1.08E-01
G4VMscModel::GetTransportMeanFreePath(G4Par	-3.03E+05	5.81E-01	-2.69E-01
G4UrbanMscModel::StartTracking(G4Track*)	-1.07E+06	4.82E-01	-1.42E+00
G4VParticleChange::CheckIt(G4Track const&)	-6.87E+05	4.47E-01	-7.47E-01
G4VEmProcess::PostStepDolt(G4Track const&, G	-2.15E+07	4.47E-01	-1.94E+01

Increases in level 1 and 2 cache misses, number of instructions and TLB instruction misses

Work in progress in TAU-related tools

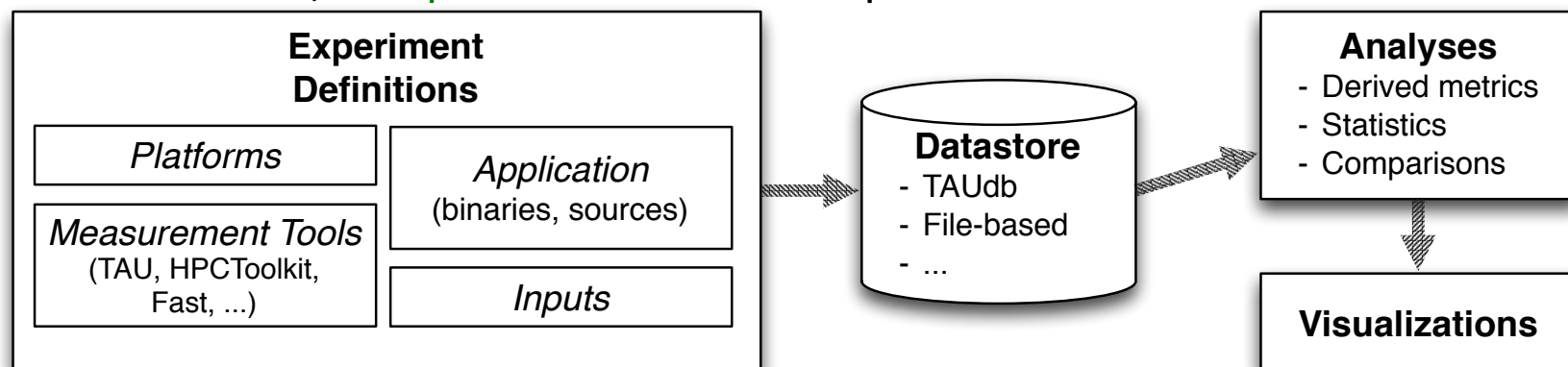
- ❑ Make performance experiments easier, faster, and more reproducible
 - Autoperf
 - ◆ mostly done for multithreaded code running on multicores
 - ◆ currently focusing on GPUs and Intel MIC
 - Integration with IDEs such as Eclipse as part of the Roofline Toolkit subproject in SUPER
- ❑ Multi- and many core analysis
- ❑ Automation of iterative analysis – defining multi-experiment workflows
 - Right now separate, manually created Autoperf configurations required

TAU Commander



Autoperf

- ❑ *Simple* tool for performance experiments and associated analysis
- ❑ Adds a layer of abstraction over *existing* performance tools
- ❑ Automates tedious and error-prone tasks
 - Selecting performance counters (**minimize** # of experiments required)
 - Setting up the **environment** for each tool, managing **batch** jobs
 - Generating **selective** profiling configuration based on sampling results
 - Configuring access to **databases**, uploading data
 - **Reusable** and **extensible** analyses that normal humans should be able to understand; **comparisons** across multiple code versions



Autoperf example

- ❑ Base version: -O0
- ❑ Compare -O2 and -O3 improvement

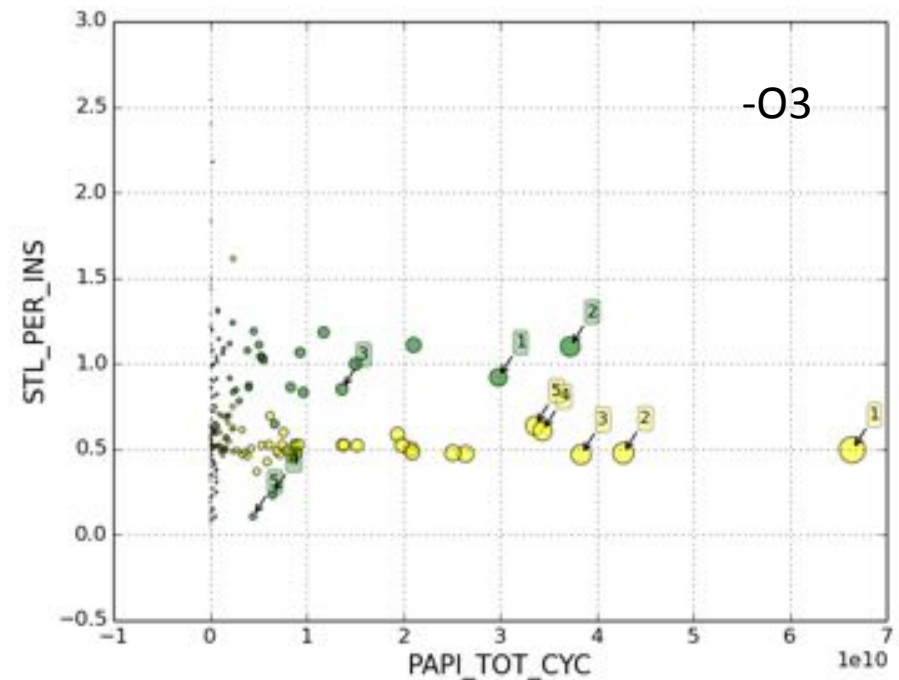
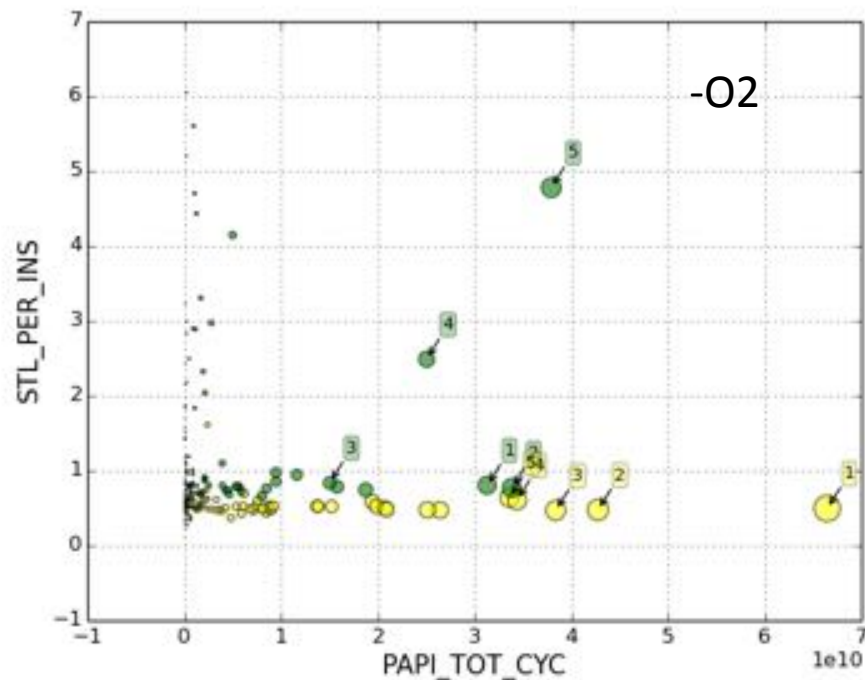
Table 1: Top five functions (total cycles)

Stall Cycles ($\times 10^{10}$)	Total Cycles ($\times 10^{10}$)	Function Name
3.13	6.65	GeantTrack_v::ComputeTransportLength
2.01	4.28	GeantTrack_v::PropagateInVolumeSingle
1.80	3.84	WorkloadManager::TransportTracks
1.55	3.44	GeantTrack_v::AddTracks
1.48	3.36	TOPLEVEL

Xiaoguang Dai, Boyana Norris, Allen Malony, “Autoperf: Workflow Support for Performance Experiments,” *Workshop on Challenges in Performance Methods for Software Development (WOSP-C'15)*, Austin, TX, 2015.

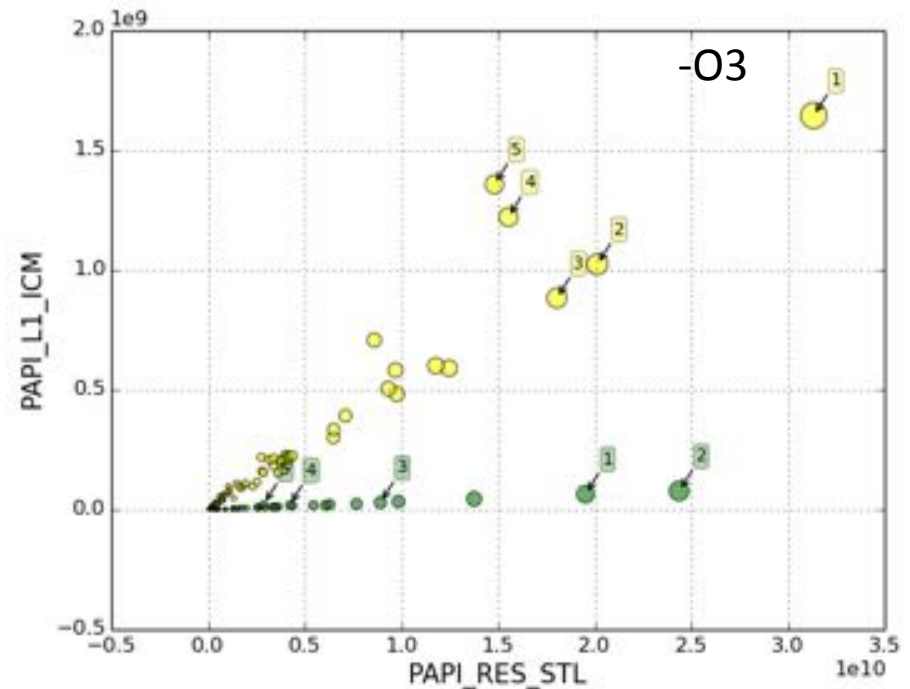
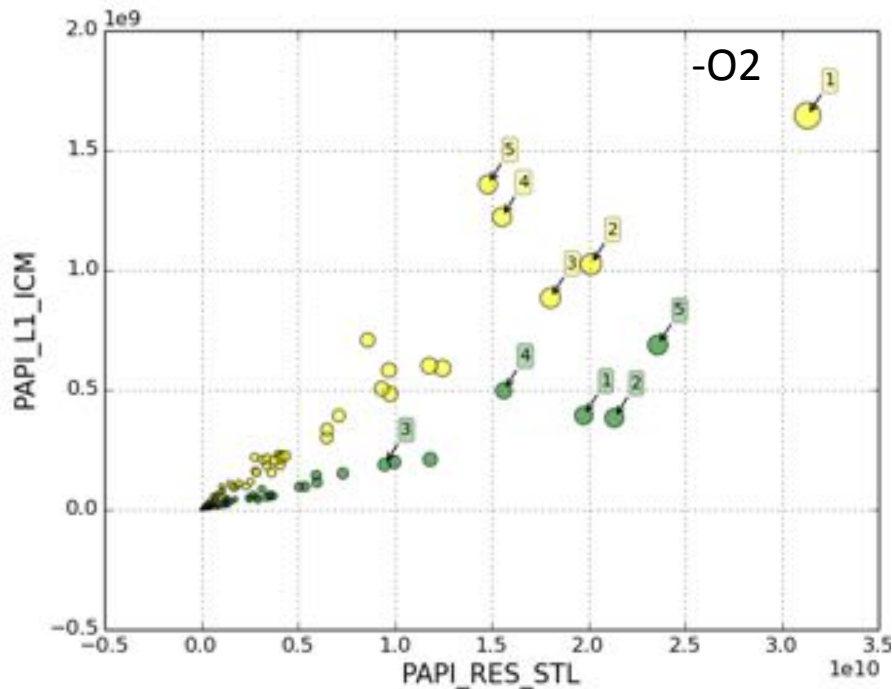
Example: Effect of optimizations

- ❑ Stalls per instruction vs total cycles—O2 unexpectedly increases stalls per instruction in two of the functions



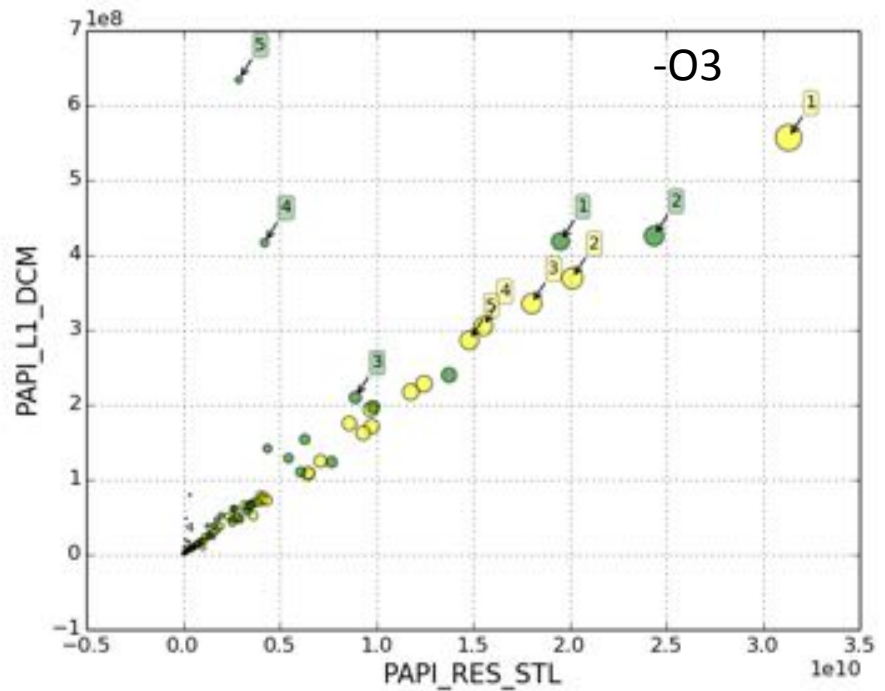
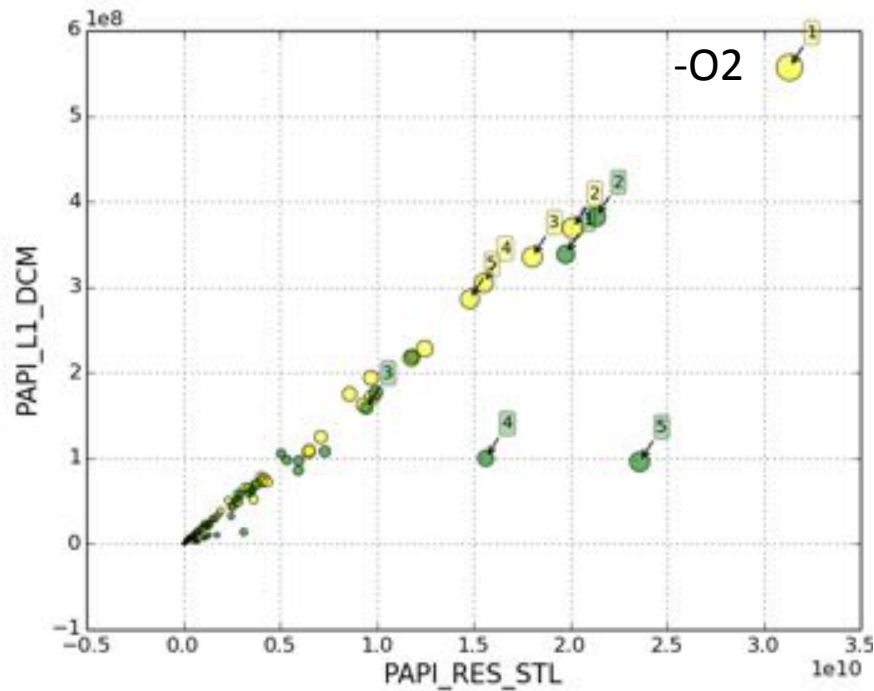
Example: Effect of optimizations

- ❑ L1 instruction cache misses vs total cycles—O3 eliminates most instruction misses, but with little impact on stalls, showing that they are not a major contributor to stalls



Example: Effect of optimizations

- L1 data cache misses vs total cycles—O3 increases L1 misses in some functions, but with little effect on overall stalls



Summary

- ❑ TAU provides comprehensive measurement capabilities
- ❑ A number of new meta-tools being developed to make it easier to create fully automated performance experiments