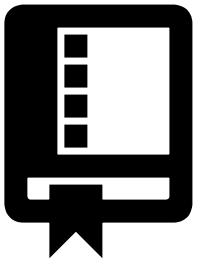
Advanced Modular Software Performance Monitoring

CPU profiling with Intel® VTune™ Amplifier XE

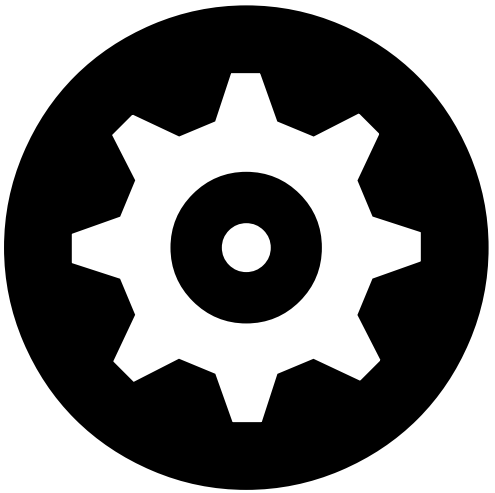
Alexander Mazurov
Ferrara University, CERN



I. Event Processing Software
II. Profilers
III. Intel® VTune™ Amplifier XE
IV. Gaudi Framework
V. Gaudi Intel Profiler Auditor
VI. Profiling examples

I. Event Processing Software

Physics events



The Higgs Boson

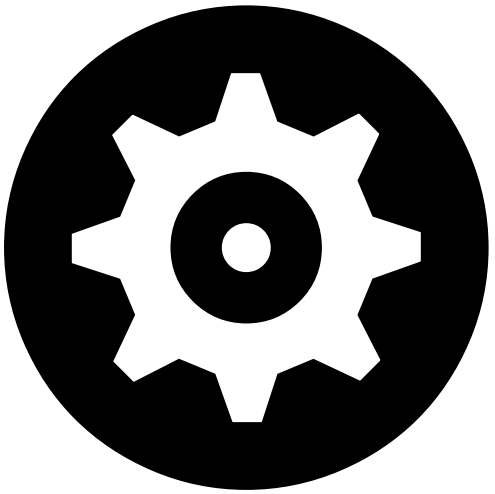
Simulation * Trigger * Analysis

LHCb High Level Trigger (HLT) Software

Moore

Detector events

10^6 events/sec



Events to storage

4500 events/sec

II. Profilers

Collect information related to how an application or system perform.

CPU Profiler

Measure frequency and duration of functions calls and/or code instructions.

Profiling Techniques

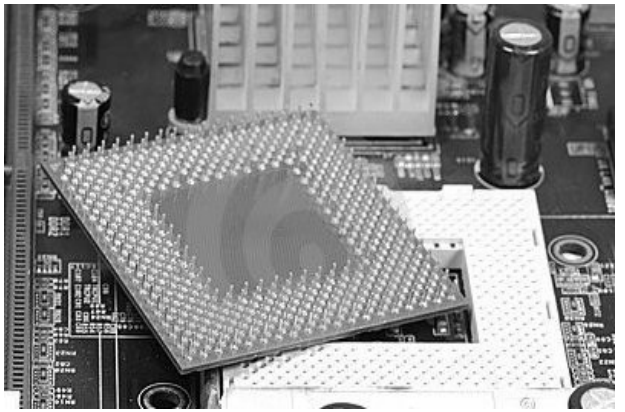
- Hardware counters
- Instrumenting the code

Hardware counters

Exploit hardware performance counters from Performance Monitoring Unit (PMU)

Counters:

- **Translation lookaside buffer (TLB) misses**
- **Cache misses**
- **Stall cycles**
- **Memory access latency**
- **…**



Perfmon2 * Intel VTune Amplifier

Instrumenting the code

- Statically:
  * Change code manually / automatically
  * Compiler assisted (gcc -pg)

- Dynamically (at runtime):
  * Change code in runtime
    - Valgrind
    - Google Performance Tools
    - Intel VTune Amplifier

III. VTune™ Amplifier XE

Performance Profiling Tool

- x86 (32 and 64-bit)
- GUI and CLI

VTune™ Features

Runtime instrumenting profiler

- User-mode sampling
- Hardware-based sampling
- Concurrency and locks and waits analysis
- Threading timeline
- Attach to a running process
- Source view

How user-mode sampling works?

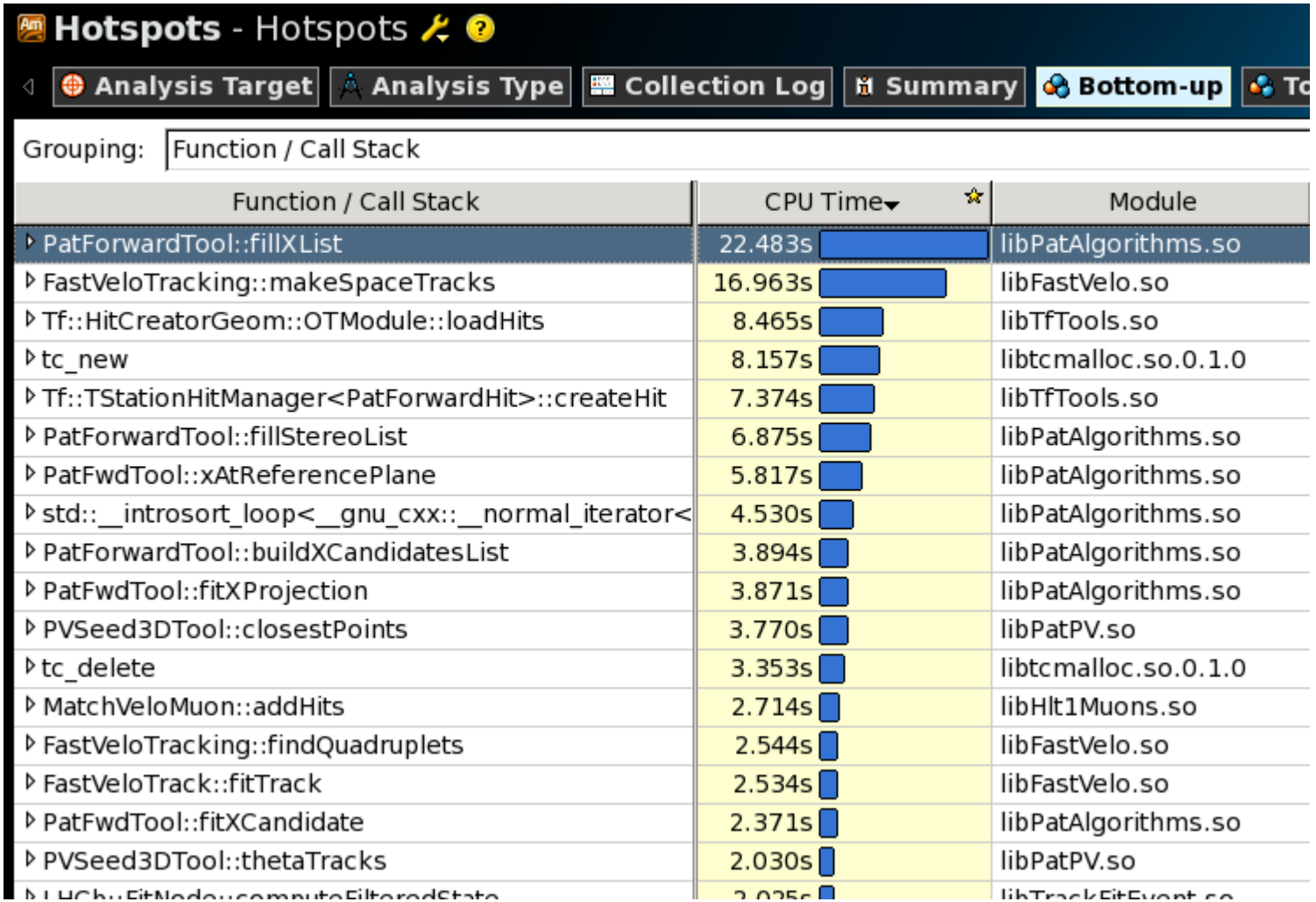
1) Interupts a process
2) Collect samples of all active instruction addresses
3) Restore a call sequence upon each sample.

User-mode analysis types

- Hotspots
- Concurrency
- Locks and Waits

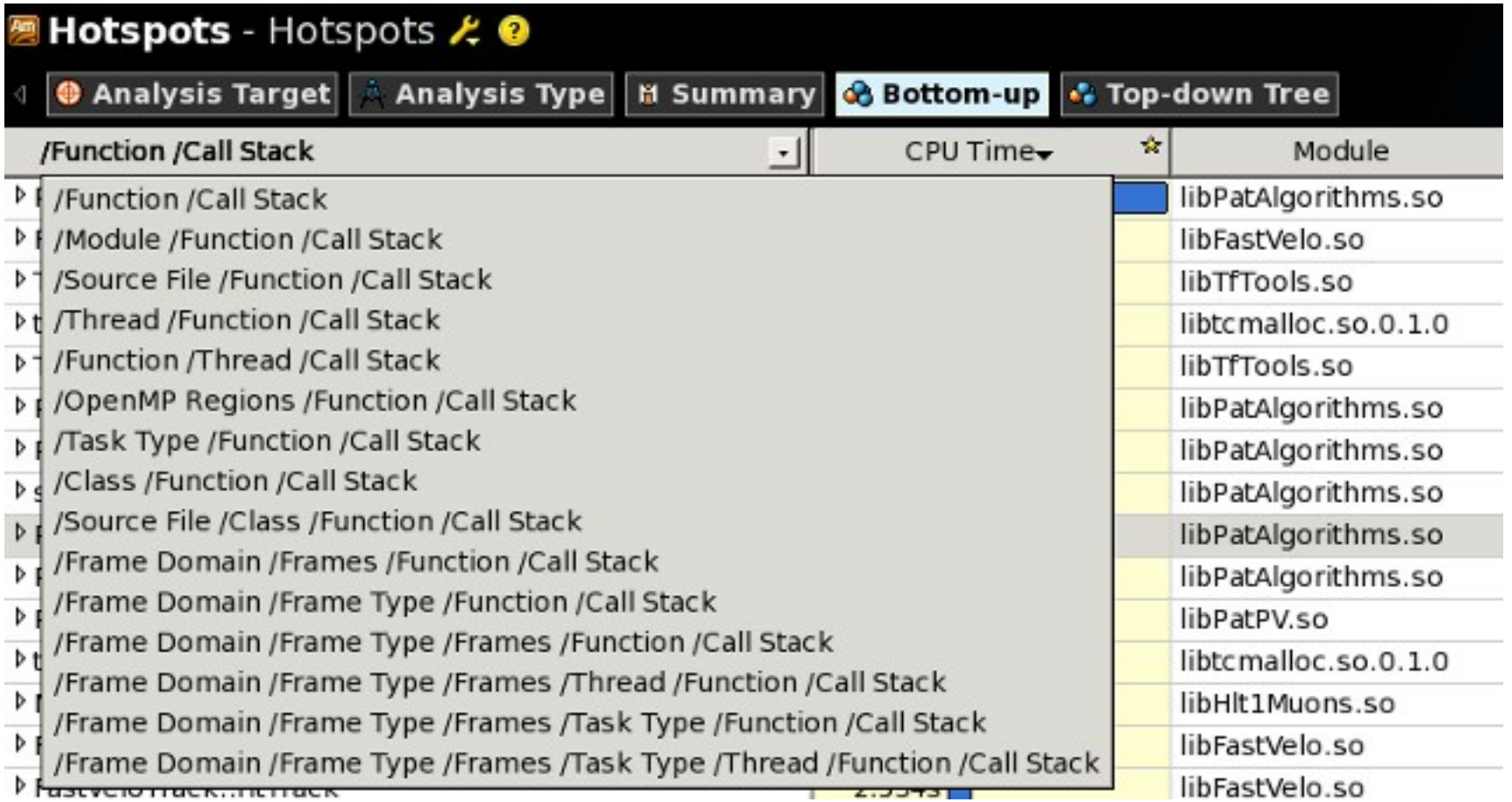
User-mode sampling

Hotspots analysis:



Function / Call Stack	CPU Time	Module
PatForwardTool::fillXList	22.483s	libPatAlgorithms.so
FastVeloTracking::makeSpaceTracks	16.963s	libFastVelo.so
Tf::HitCreatorGeom::OTModule::loadHits	8.465s	libTfTools.so
tc_new	8.157s	libtcmalloc.so.0.1.0
Tf::TStationHitManager<PatForwardHit>::createHit	7.374s	libTfTools.so
PatForwardTool::fillStereoList	6.875s	libPatAlgorithms.so
PatFwdTool::xAtReferencePlane	5.817s	libPatAlgorithms.so
std::__introsort_loop<__gnu_cxx::__normal_iterator<	4.530s	libPatAlgorithms.so
PatForwardTool::buildXCandidatesList	3.894s	libPatAlgorithms.so
PatFwdTool::fitXProjection	3.871s	libPatAlgorithms.so
PVSeed3DTool::closestPoints	3.770s	libPatPV.so
tc_delete	3.353s	libtcmalloc.so.0.1.0
MatchVeloMuon::addHits	2.714s	libHlt1Muons.so
FastVeloTracking::findQuadruplets	2.544s	libFastVelo.so
FastVeloTrack::fitTrack	2.534s	libFastVelo.so
PatFwdTool::fitXCandidate	2.371s	libPatAlgorithms.so
PVSeed3DTool::thetaTracks	2.030s	libPatPV.so

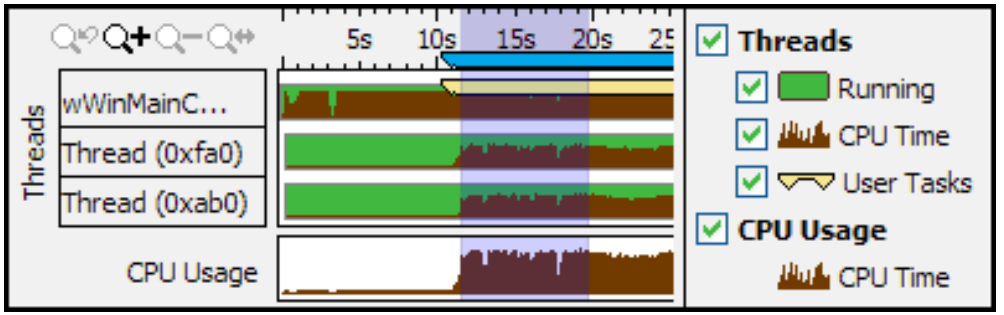
Group results



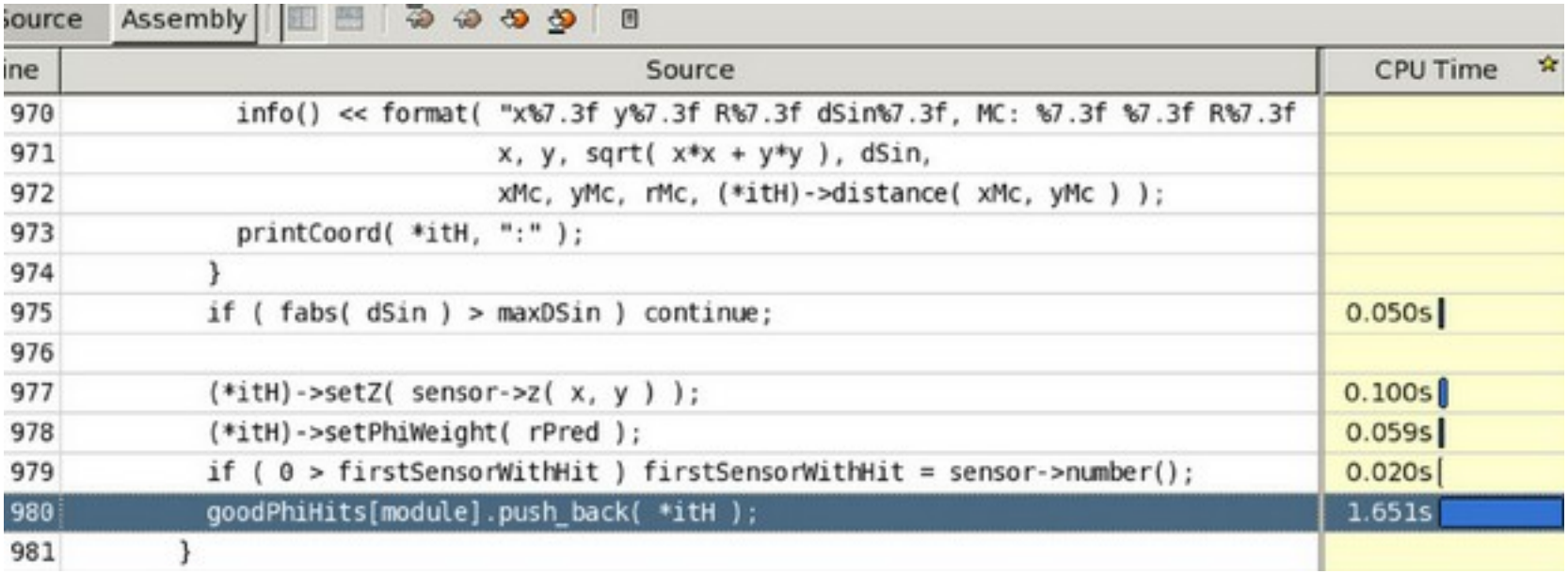
Call Stack

Call Stack	CPU Time:Total	Module
▽Selection::Line::Stage::execute	96.3%	libSelectionLine.so
Algorithm::isEnabled	0.0%	libGaudiKernel.so
StatusCode::~StatusCode	0.0%	libGaudiKernel.so
▽GaudiAlgorithm::sysExecute	96.3%	libGaudiAlgLib.so
▽Algorithm::sysExecute	96.3%	libGaudiKernel.so
▷Gaudi::Guards::AuditorGuard::AuditorGuard	0.4%	libGaudiKernel.so
▽GaudiSequencer::execute	95.4%	libGaudiAlgLib.so
▽GaudiAlgorithm::sysExecute	95.4%	libGaudiAlgLib.so
▽Algorithm::sysExecute	95.4%	libGaudiKernel.so
▷L0DUFromRawAlg::execute	3.0%	libL0DU.so
▷LoKi::HltUnit::execute	90.6%	libLoKiTrigger.so
▷LoKi::L0Filter::execute	0.4%	libLoKiHlt.so

Filter by timeline



CPU time by code line



Source | Assembly

ine	Source	CPU Time
970	info() << format("x%7.3f y%7.3f R%7.3f dSin%7.3f, MC: %7.3f %7.3f R%7.3f	
971	x, y, sqrt(x*x + y*y), dSin,	
972	xMc, yMc, rMc, (*itH)->distance(xMc, yMc));	
973	printCoord(*itH, ":");	
974	}	
975	if (fabs(dSin) > maxDSin) continue;	0.050s
976		
977	(*itH)->setZ(sensor->z(x, y));	0.100s
978	(*itH)->setPhiWeight(rPred);	0.059s
979	if (0 > firstSensorWithHit) firstSensorWithHit = sensor->number();	0.020s
980	goodPhiHits[module].push_back(*itH);	1.651s
981	}	

Debug mode (-g)

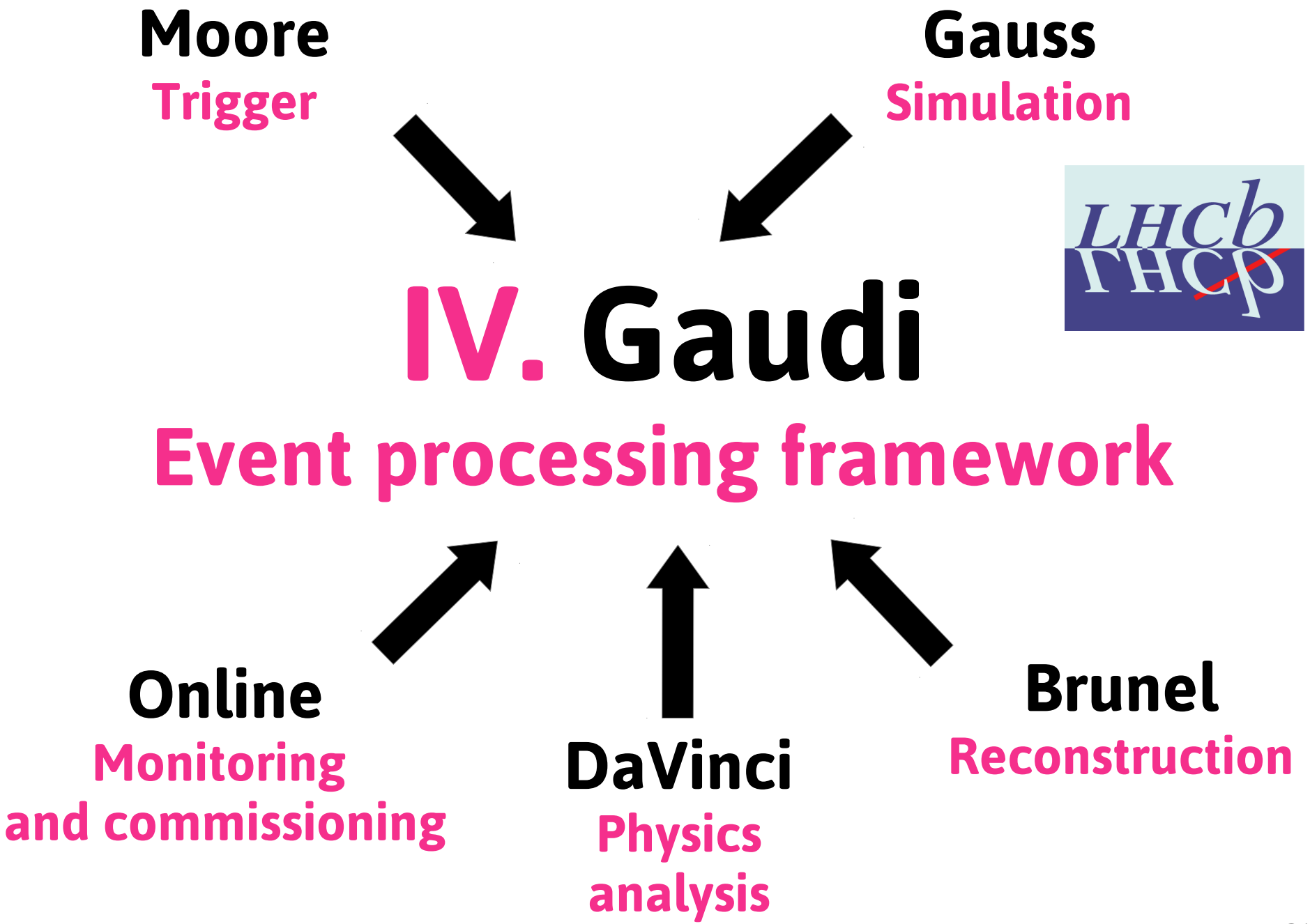
Sampling Accuracy

User-mode sampling is a statistical method and does not provide a 100% accurate results.

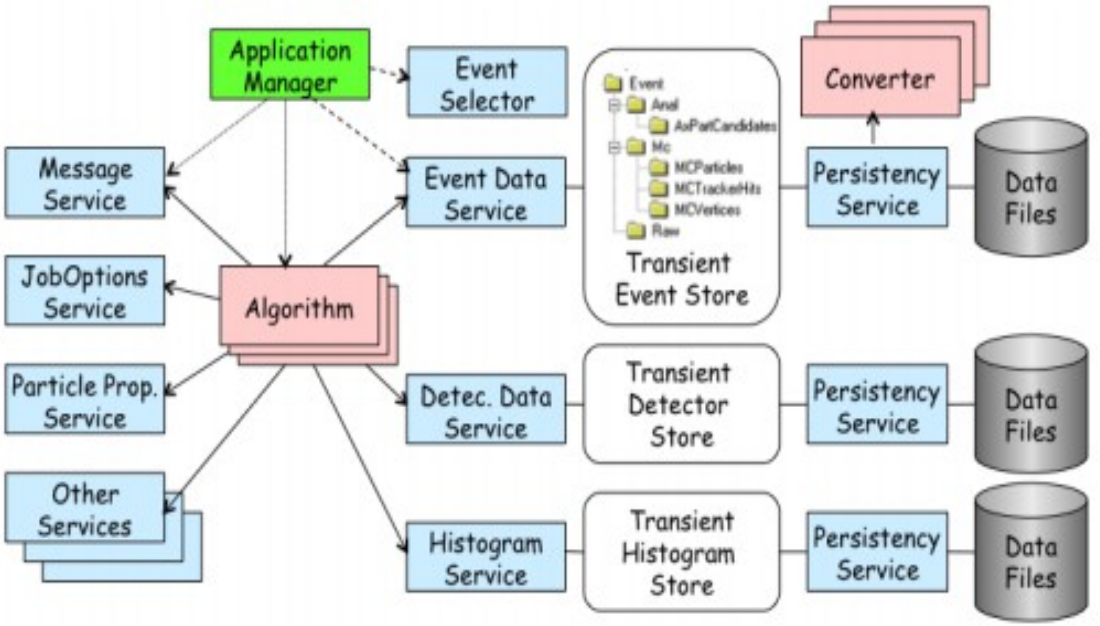
Accuracy depends on:

- **Duration of the collection**
- **Speed of processor**
- **Amount of software activity**
- **Sampling interval**
  - * **recommended value is 10 ms**
  - * **profiling is only 5% slower**

Integrating VTune™ Amplifier to Event Processing Framework


Moore
Trigger
Gauss
Simulation
LHCb
IV. Gaudi
Event processing framework
Online
Monitoring
and commissioning
DaVinci
Physics
analysis
Brunel
Reconstruction

Gaudi Architecture



Algorithms * Services * Tools

Moore Event Loop

Hlt1DiMuonHighMassFilterSequence
  Hlt1DiMuonHighMassStreamer
    FastVeloHlt
    MuonRec
    Velo2CandidatesDiMuonHighMass
  GECLooseUnit
    createITLiteClusters
    createVeloLiteClusters

Algorithms Sequence

How to profile algorithms?

V. Gaudi Intel Profiling Auditor

VTune™ User API

+

Gaudi Auditors API

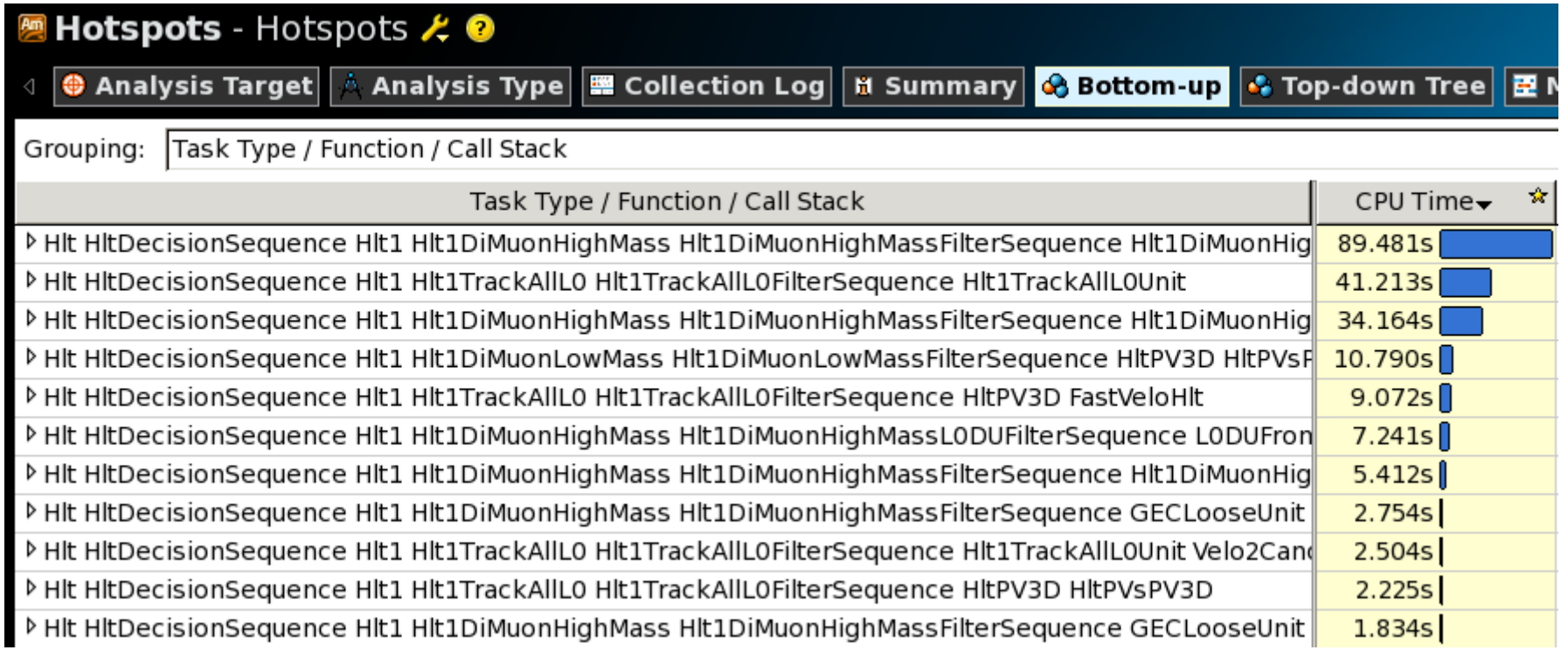
VTune™ User API

- Start/Pause profiling
- Mark profiling regions

Gaudi Auditors API

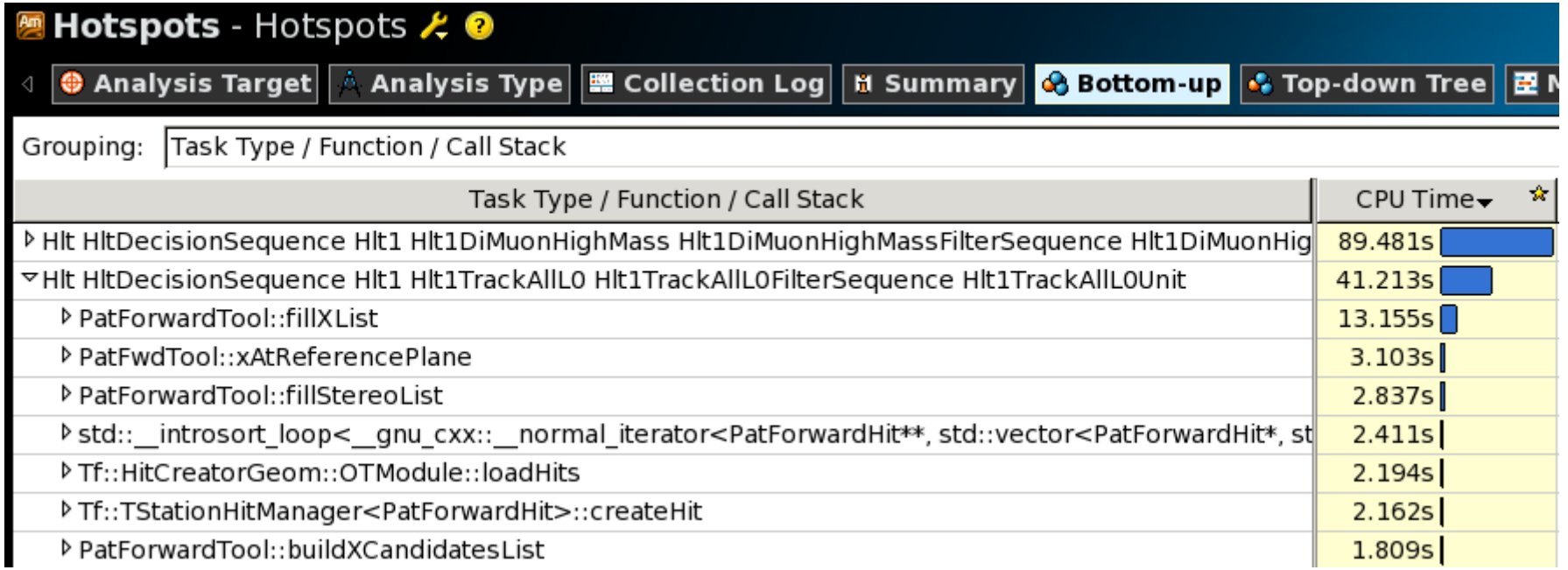
Callback functions

Algorithms profiling (I)



CPU time per sequence branch

Algorithms profiling (II)



Gaudi configuration

```
from Configurables import IntelProfilerAuditor
profiler = IntelProfilerAuditor()
profiler.StartFromEventN = 5000
profiler.StopAtEventN = 15000
AuditorSvc().Auditors +=   [profiler]
```

Run:

$> intelprofiler -o /collected/data job.py

Analyze (GUI):

$> amplxe-gui /collecter/data/r001hs

Analyze (CLI):

$> amplxe-cl -reports hotspots -r /collecter/data/r001hs

VI. Profiling examples

1. Memory allocation functions
2. Measuring profiling accuracy
3. Custom reports

1. Memory allocation functions

operatornew from libstdc++ library:

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module
PatForwardTool::fillXList	23.460s	libPatAlgorithms.so
FastVeloTracking::makeSpaceTracks	19.788s	libFastVelo.so
operatornew	18.696s	libstdc++.so.6
Tf::HitCreatorGeom::OTModule::loadHits	10.870s	libTfTools.so
Tf::TStationHitManager<PatForwardHit>::createHit	7.981s	libTfTools.so

tc_new from tcmalloc library:

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module
PatForwardTool::fillXList	22.483s	libPatAlgorithms.so
FastVeloTracking::makeSpaceTracks	16.963s	libFastVelo.so
Tf::HitCreatorGeom::OTModule::loadHits	8.465s	libTfTools.so
tc_new	8.157s	libtcmalloc.so.0.1.0
Tf::TStationHitManager<PatForwardHit>::createHit	7.374s	libTfTools.so

tc_new uses twice less time then operatornew

2. Measuring profiling accuracy

Intel Profiling Auditor
vs .
Timing Auditor

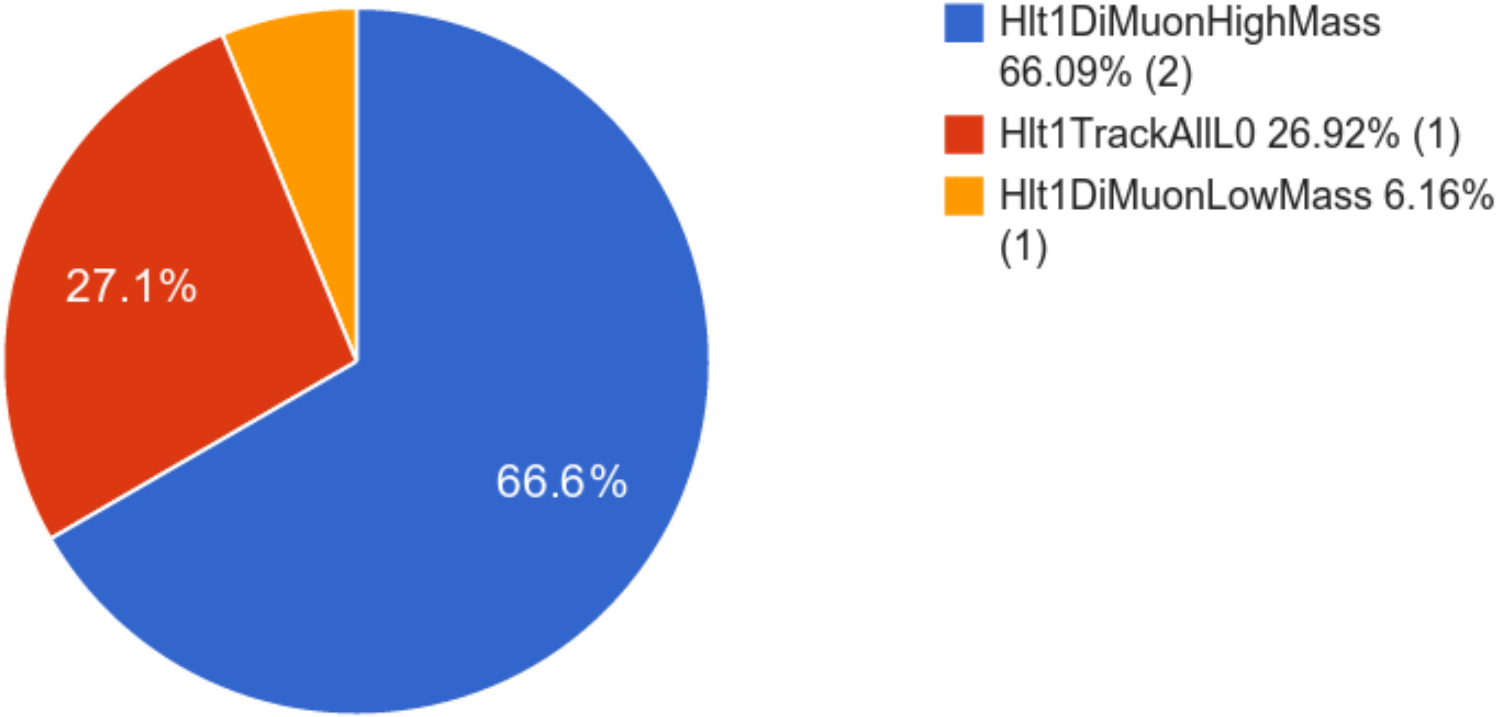
Measures the absolute time of algorithm's run

1000 events

Algorithm name	Timer (%)	Profiler (%)	**Difference**
L0Muon	100	100	-
Hlt1TrackAllL0Unit	35.872	35.147	**0.725**
FastVeloHlt	29.648	28.235	**1.413**
L0Calo	30.478	29.736	**0.742**
HltPVsPV3D	2.491	2.25	**0.241**

3. Custom reports

Build reports using CSV files exported from VTune Amplifier



Conclusions

Intel® VTune™ Amplifier XE:

+ Various analysis types and reports
+ Rich User API
+ Reasonable overhead time