

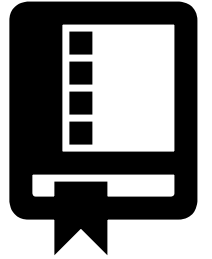
Advanced Modular Software Performance Monitoring

CPU profiling with
Intel® VTune™ Amplifier XE



Alexander Mazurov
Ferrara University, CERN

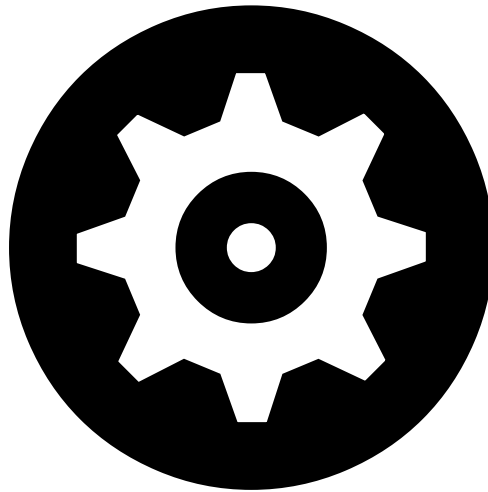




- I. Event Processing Software**
- II. Profilers**
- III. Intel® VTune™ Amplifier XE**
- IV. Gaudi Framework**
- V. Gaudi Intel Profiler Auditor**
- VI. Profiling examples**

I. Event Processing Software

**Physics
events**



**The Higgs
Boson**



Simulation * Trigger * Analysis

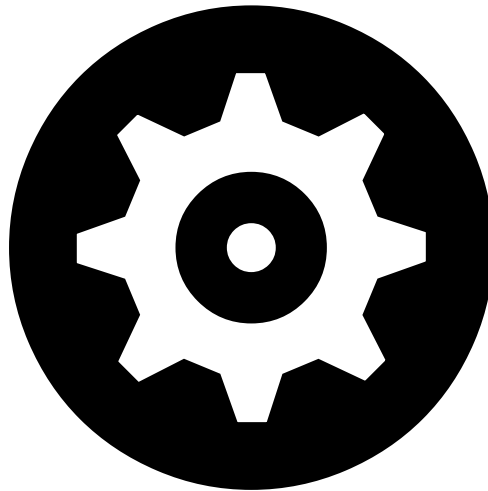
LHCb High Level Trigger (HLT) Software

Moore

**Detector
events**



10⁶ 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

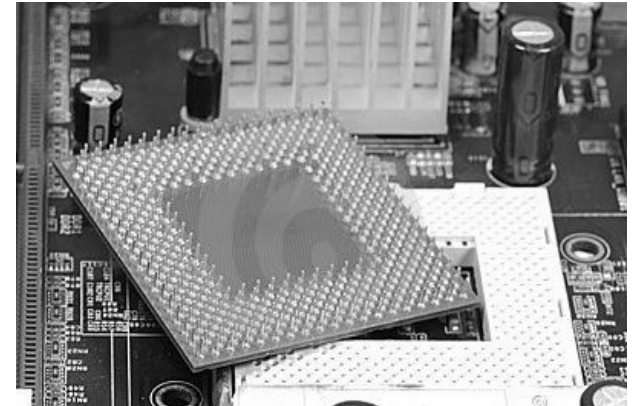
- **Hardware counters**
- **Instrumenting the code**

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?

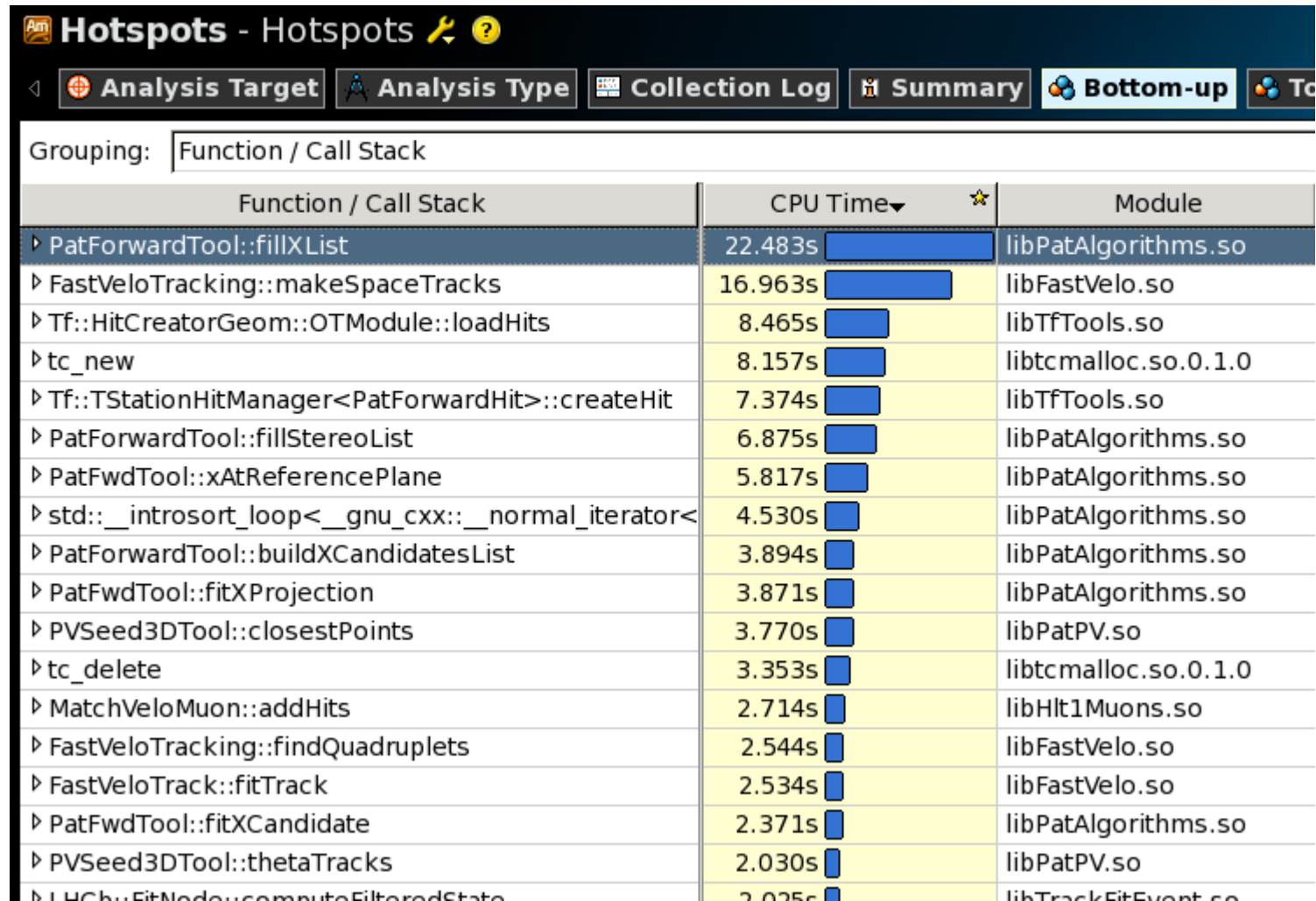
- 1) **Interrupts** a process
- 2) **Collect** samples of all active instruction addresses
- 3) **Restore** a call sequence upon each sample.

User-mode analysis types

- **Hotspots**
- **Concurrency**
- **Locks and Waits**

User-mode sampling

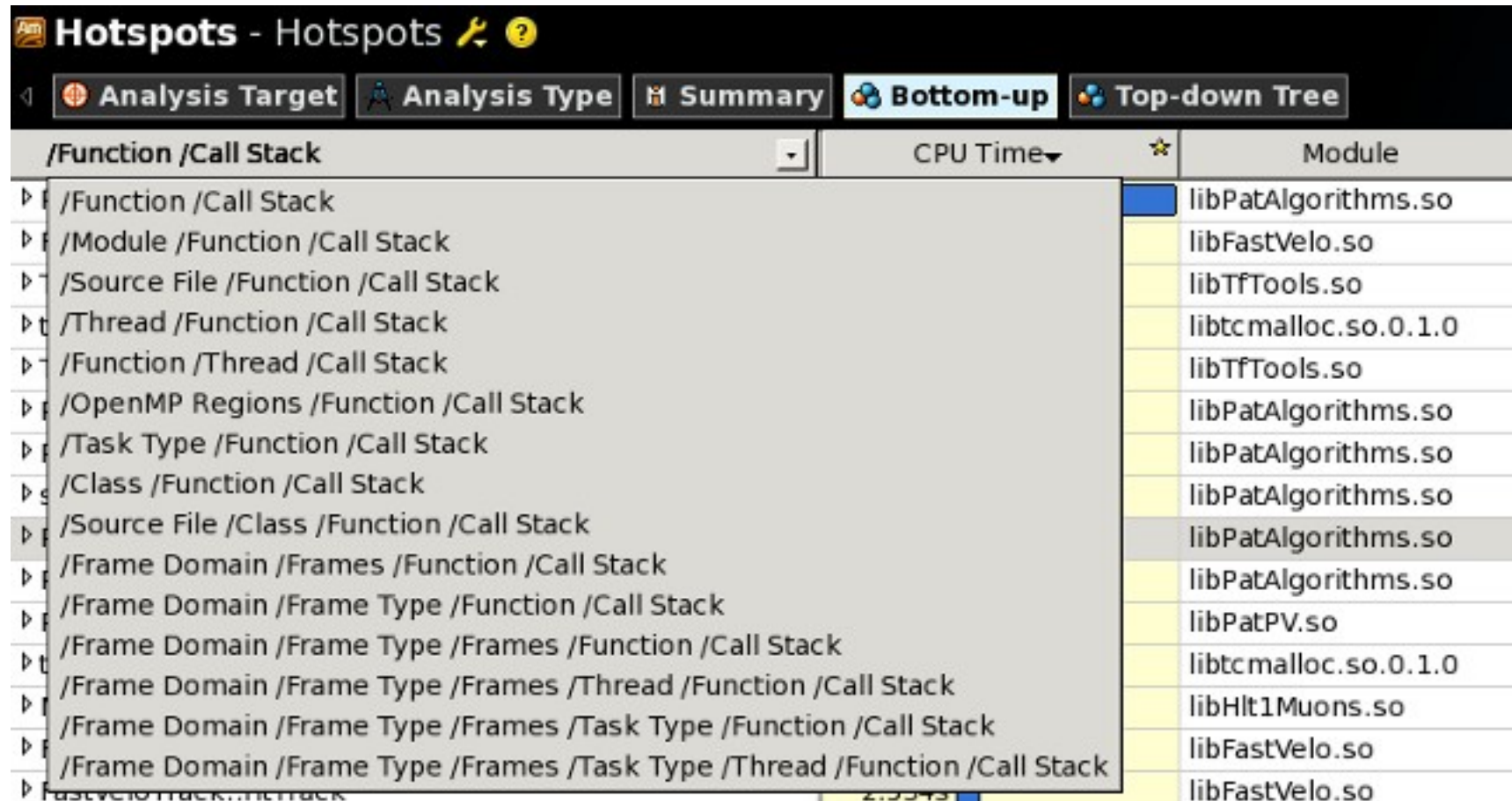
Hotspots analysis:



The screenshot shows the 'Hotspots' analysis tool interface. The title bar reads 'Hotspots - Hotspots'. Below the title bar are several tabs: 'Analysis Target', 'Analysis Type', 'Collection Log', 'Summary', 'Bottom-up', and 'Tools'. The 'Bottom-up' tab is selected. Below the tabs, there is a 'Grouping:' dropdown menu set to 'Function / Call Stack'. The main area displays a table with three columns: 'Function / Call Stack', 'CPU Time', and 'Module'. The table lists various functions and their corresponding CPU times, with blue bars representing the time spent. The top function is 'PatForwardTool::fillXList' with a CPU time of 22.483s, followed by 'FastVeloTracking::makeSpaceTracks' at 16.963s, and 'Tf::HitCreatorGeom::OTModule::loadHits' at 8.465s.

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
▸ LHCb::FitModule::computeFilteredState	2.025s	libTrackFitEvent.so







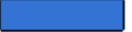
Group results



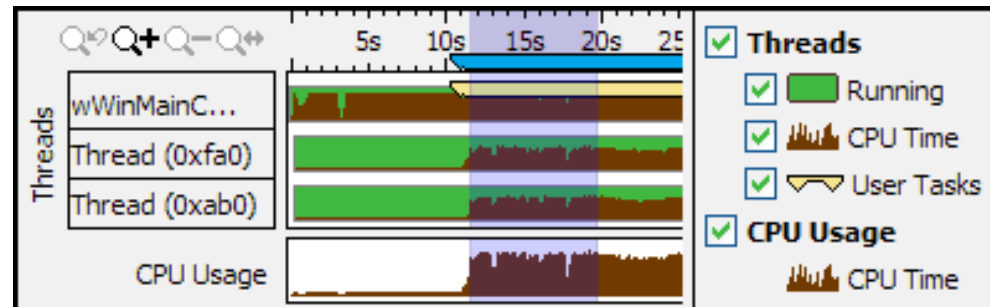
The screenshot shows the 'Hotspots' application window. The title bar reads 'Hotspots - Hotspots'. Below the title bar are several tabs: 'Analysis Target', 'Analysis Type', 'Summary', 'Bottom-up', and 'Top-down Tree'. The 'Bottom-up' tab is selected. The main content area displays a table with columns for 'Function / Call Stack', 'CPU Time', and 'Module'. A context menu is open over the table, listing various analysis categories. The table data is as follows:

Function / Call Stack	CPU Time	Module
/Function / Call Stack		libPatAlgorithms.so
/Module /Function / Call Stack		libFastVelo.so
/Source File /Function / Call Stack		libTfTools.so
/Thread /Function / Call Stack		libtcmalloc.so.0.1.0
/Function /Thread / Call Stack		libTfTools.so
/OpenMP Regions /Function / Call Stack		libPatAlgorithms.so
/Task Type /Function / Call Stack		libPatAlgorithms.so
/Class /Function / Call Stack		libPatAlgorithms.so
/Source File /Class /Function / Call Stack		libPatAlgorithms.so
/Frame Domain /Frames /Function / Call Stack		libPatAlgorithms.so
/Frame Domain /Frame Type /Function / Call Stack		libPatPV.so
/Frame Domain /Frame Type /Frames /Function / Call Stack		libtcmalloc.so.0.1.0
/Frame Domain /Frame Type /Frames /Thread /Function / Call Stack		libHlt1Muons.so
/Frame Domain /Frame Type /Frames /Task Type /Function / Call Stack		libFastVelo.so
/Frame Domain /Frame Type /Frames /Task Type /Thread /Function / Call Stack		libFastVelo.so

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
▸ LODUFromRawAlg::execute	3.0%	libLODU.so
▸ LoKi::HltUnit::execute	90.6% 	libLoKiTrigger.so
▸ LoKi::L0Filter::execute	0.4% [libLoKiHlt.so

Filter by timeline



CPU time by code line

line	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



IV. Gaudi

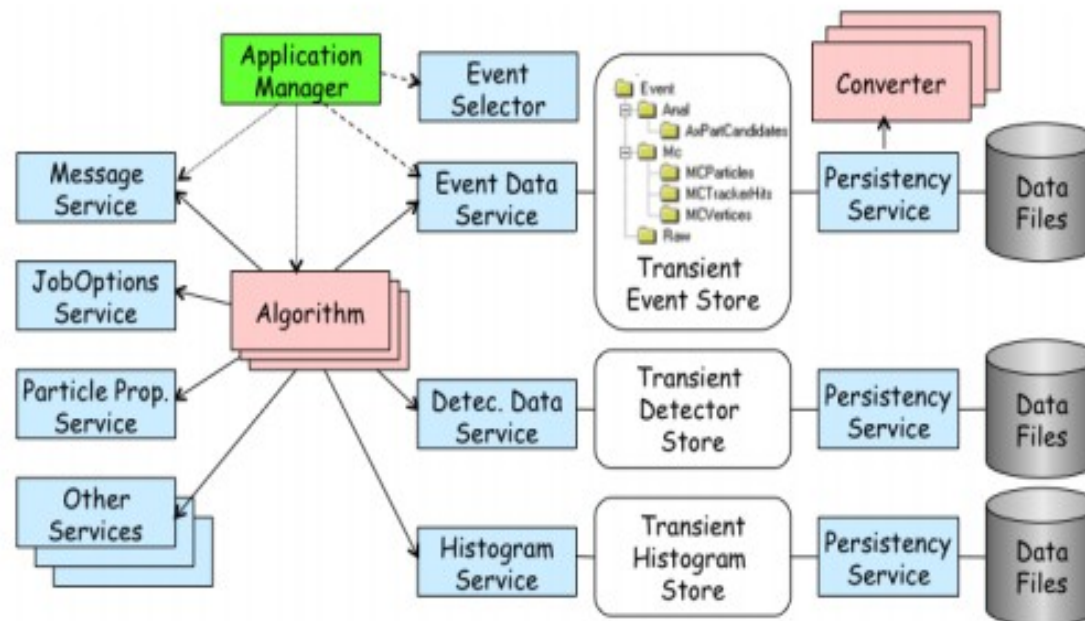
Event processing framework

Online
Monitoring
and commissioning

DaVinci
Physics
analysis

Brunel
Reconstruction

Gaudi Architecture



Algorithms * Services * Tools

Moore Event Loop

Hlt1DiMuonHighMassFilterSequence
Hlt1DiMuonHighMassStreamer
FastVeloHlt
MuonRec
Velo2CandidatesDiMuonHighMass
GECloseUnit
createTLiteClusters
createVeloLiteClusters

Algorithms
Sequence

How to profile algorithms?

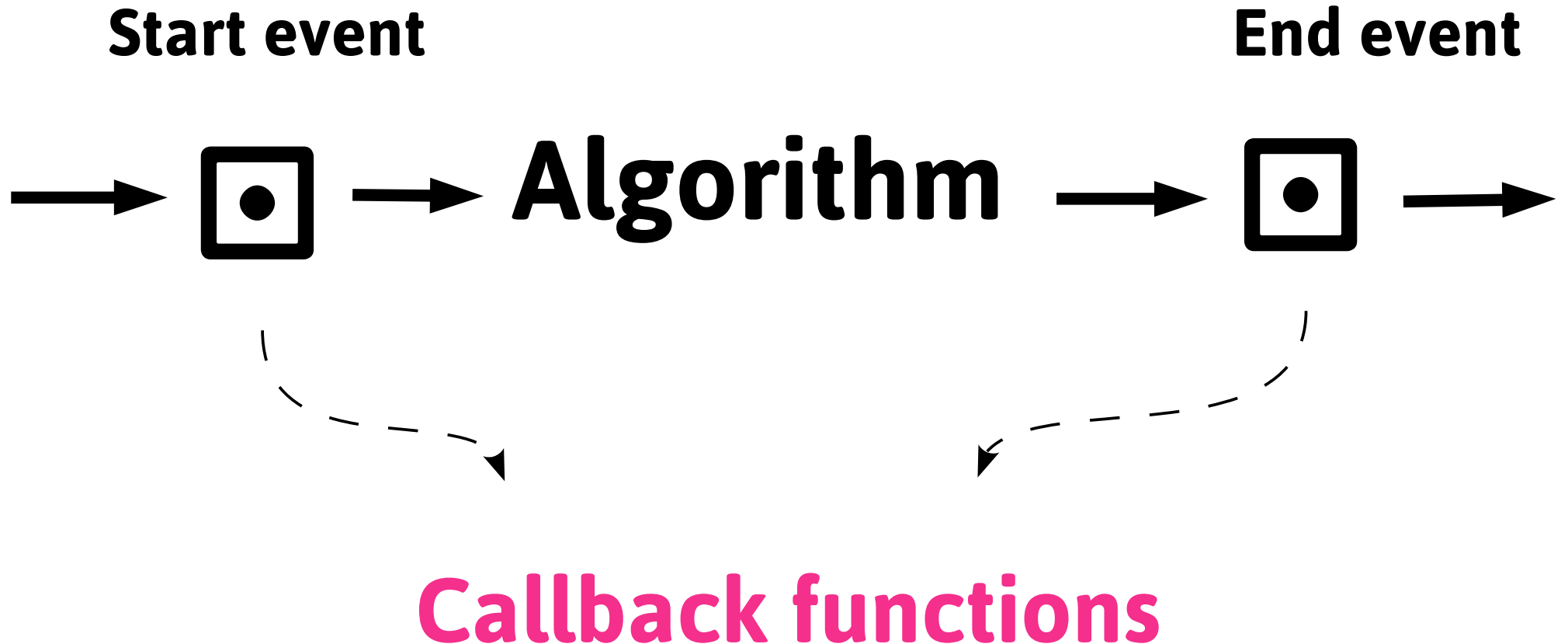
V. Gaudi Intel Profiling Auditor

VTune™ User API
+
Gaudi Auditors API

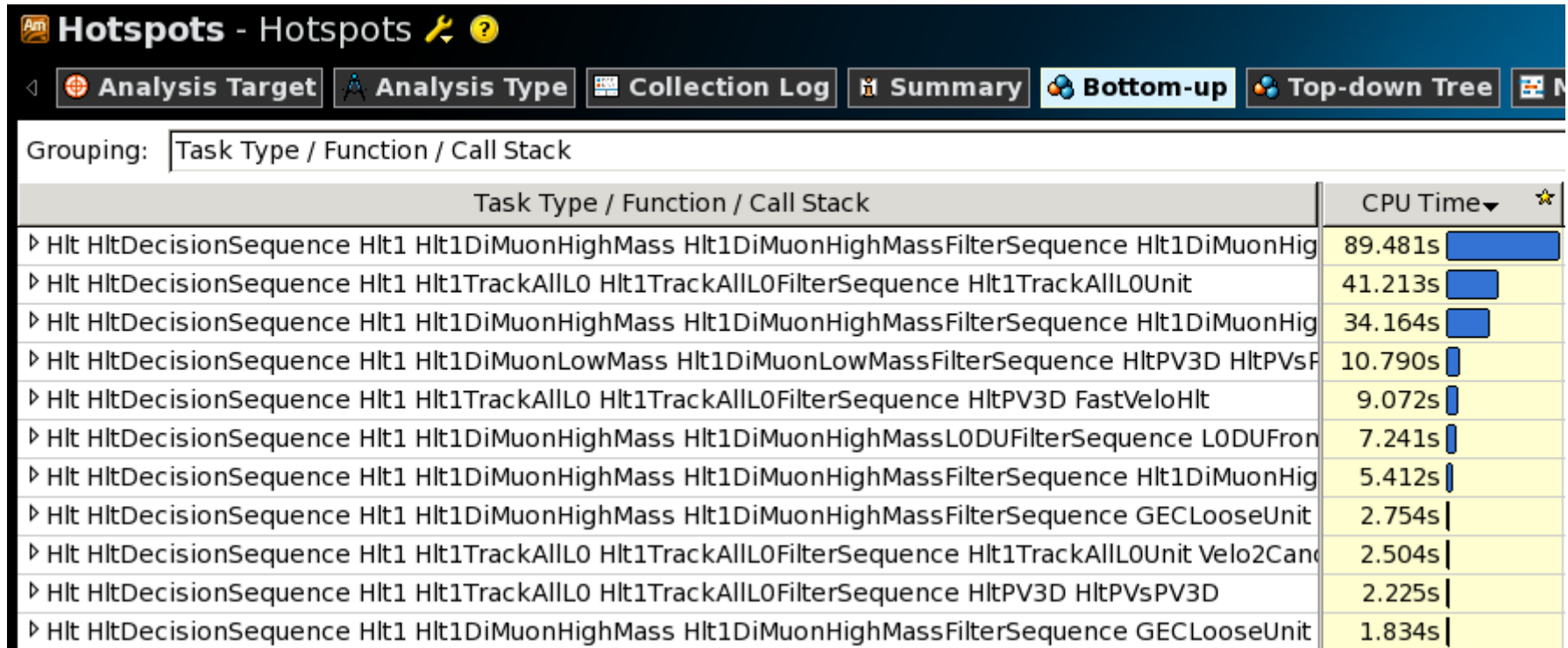
VTune™ User API

- Start/Pause profiling**
- Mark profiling regions**

Gaudi Auditors API



Algorithms profiling (I)



The screenshot shows the 'Hotspots' application interface. The title bar reads 'Hotspots - Hotspots'. Below the title bar is a navigation menu with buttons for 'Analysis Target', 'Analysis Type', 'Collection Log', 'Summary', 'Bottom-up', and 'Top-down Tree'. The 'Bottom-up' button is selected. Below the navigation menu is a 'Grouping:' dropdown menu set to 'Task Type / Function / Call Stack'. The main content area is a table with two columns: 'Task Type / Function / Call Stack' and 'CPU Time'. The table contains 10 rows of data, each representing a different sequence branch. The CPU time values are listed in seconds, and each row has a corresponding blue bar chart representing the time. The first row has the highest CPU time at 89.481s, and the last row has the lowest at 1.834s.

Task Type / Function / Call Stack	CPU Time
‣ Hlt HltDecisionSequence Hlt1 Hlt1DiMuonHighMass Hlt1DiMuonHighMassFilterSequence Hlt1DiMuonHig	89.481s
‣ Hlt HltDecisionSequence Hlt1 Hlt1TrackAllL0 Hlt1TrackAllL0FilterSequence Hlt1TrackAllL0Unit	41.213s
‣ Hlt HltDecisionSequence Hlt1 Hlt1DiMuonHighMass Hlt1DiMuonHighMassFilterSequence Hlt1DiMuonHig	34.164s
‣ Hlt HltDecisionSequence Hlt1 Hlt1DiMuonLowMass Hlt1DiMuonLowMassFilterSequence HltPV3D HltPVsP	10.790s
‣ Hlt HltDecisionSequence Hlt1 Hlt1TrackAllL0 Hlt1TrackAllL0FilterSequence HltPV3D FastVeloHlt	9.072s
‣ Hlt HltDecisionSequence Hlt1 Hlt1DiMuonHighMass Hlt1DiMuonHighMassL0DUFilterSequence L0DUFron	7.241s
‣ Hlt HltDecisionSequence Hlt1 Hlt1DiMuonHighMass Hlt1DiMuonHighMassFilterSequence Hlt1DiMuonHig	5.412s
‣ Hlt HltDecisionSequence Hlt1 Hlt1DiMuonHighMass Hlt1DiMuonHighMassFilterSequence GECLooseUnit	2.754s
‣ Hlt HltDecisionSequence Hlt1 Hlt1TrackAllL0 Hlt1TrackAllL0FilterSequence Hlt1TrackAllL0Unit Velo2Can	2.504s
‣ Hlt HltDecisionSequence Hlt1 Hlt1TrackAllL0 Hlt1TrackAllL0FilterSequence HltPV3D HltPVsPV3D	2.225s
‣ Hlt HltDecisionSequence Hlt1 Hlt1DiMuonHighMass Hlt1DiMuonHighMassFilterSequence GECLooseUnit	1.834s

CPU time per sequence branch

Algorithms profiling (II)

Hotspots - Hotspots

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Top-down Tree

Grouping: Task Type / Function / Call Stack

Task Type / Function / Call Stack	CPU Time
▶ Hlt HltDecisionSequence Hlt1 Hlt1DiMuonHighMass Hlt1DiMuonHighMassFilterSequence Hlt1DiMuonHig	89.481s
▼ Hlt HltDecisionSequence Hlt1 Hlt1TrackAllLO Hlt1TrackAllLOFilterSequence Hlt1TrackAllLOUnit	41.213s
▶ PatForwardTool::fillXList	13.155s
▶ PatFwdTool::xAtReferencePlane	3.103s
▶ PatForwardTool::fillStereoList	2.837s
▶ std::__introsort_loop<__gnu_cxx::__normal_iterator<PatForwardHit**, std::vector<PatForwardHit*, st	2.411s
▶ Tf::HitCreatorGeom::OTModule::loadHits	2.194s
▶ Tf::TStationHitManager<PatForwardHit>::createHit	2.162s
▶ PatForwardTool::buildXCandidatesList	1.809s

Gaudi configuration

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

Run:

```
$> intelprofler -o /collected/data job.py
```

Analyze (GUI):

```
$> amplxe-gui /collector/data/r001hs
```

Analyze (CLI):

```
$> amplxe-cl -reports hotspots -r /collector/data/r001hs
```




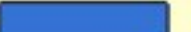
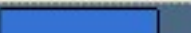
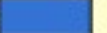

VI. Profiling examples

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

1. Memory allocation functions








operatornew from libstdc++ library:

Grouping:

Function / Call Stack	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	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

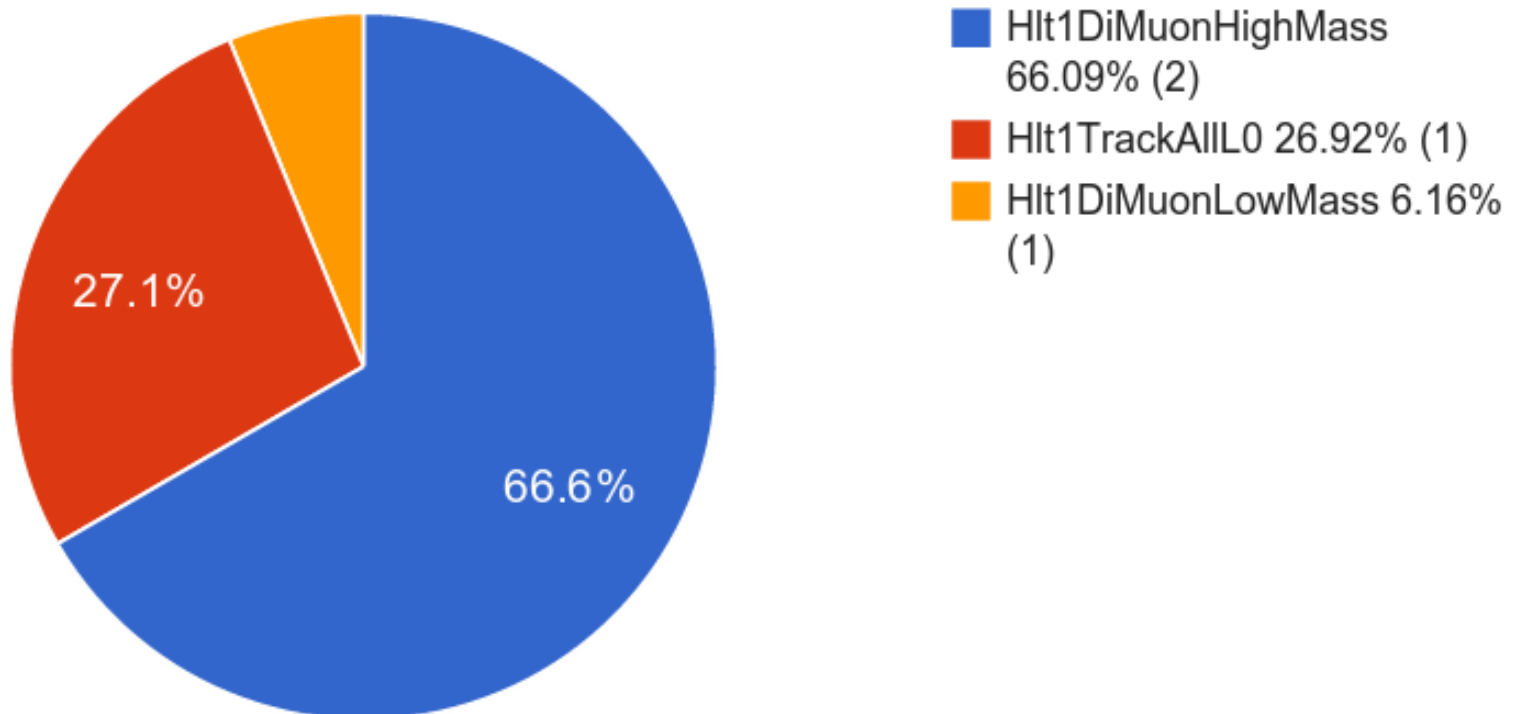
Measures the absolute time of
algorithm's run

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**