Comparison of Profiling Results for Run 3 and High Pileup LHC Simulation and Reconstruction

Mentee: Bongiwe Mkwananzi(Bethune-Cookman University)
Mentor: Patrick Gartung (Fermilab)

Abstract

Abstract: The performance of CMS simulation and reconstruction software will be critical given the resource constraints on CPU and memory for the high luminosity LHC. Profiling the CPU and memory usage of the simulation and reconstruction software with every release is essential to ensure that performance remains stable or improves. Several profilers are available for profiling CMS software including Igprof and Intel Vtune. This project involves profiling with both profilers for each new release of CMSSW.

Overview

- Introduction
- Work to do
- Method
- Process
- Results
- Discussion
- What I learnt
- References

Introduction

- A profiler is software that:

  -Records snapshots of code performance on CPU.

  -Reports the sum of time spent in functions and their children.

  -Reports the sum of memory allocated and used in functions and their children.

Profilers to be used:

Ignominous Profiler:

- Igprof - fast and lightweight, handle loaded shared libraries, threads and sub-processes.
- It currently works on Linux (`ia32`, `x86_64`). Eons ago it worked also on Mac OS X (`PPC`).
- IgProf can be run in one of three modes: as a performance profiler, as a memory profiler, or in instrumentation mode.
- When used as a performance profiler it provides statistical sampling based performance profiles of the application.
- When used as a memory profiler information about both memory leaks and the total dynamic memory allocations are available.
- It can also be used to obtain a profile the live memory allocations in the heap at any given instant during the application run, although this requires a small code modification to signal from within your application the appropriate time to obtain the profile.

Continuation of profilers to be used…

Intel Vtune:

Vtune - Analysis and tuning tool that provides various examinations of performances.

Use VTune Profiler to locate or determine:

- The most time-consuming (hot) functions in your application and/or on the whole system
- Sections of code that do not effectively utilize available processor time
- The best sections of code to optimize for sequential performance and for threaded performance
- Synchronization objects that affect the application performance
- Whether, where, and why your application spends time on input/output operations
- Whether your application is CPU or GPU bound and how effectively it offloads code to the GPU
- The performance impact of different synchronization methods, different numbers of threads, or different algorithms
- Thread activity and transitions
- Hardware-related issues in your code such as data sharing, cache misses, branch misprediction, and others.

N.B. For this analysis, I took the descriptive analysis approach as the data changes on a daily basis. I selected data from the 31st day of July(1100 hours CERN time) as the sample.

Work to do

- Connect to the CMSLPC cluster.

- Learn how to run CMS software.

- Learn how to run the profiler on CMS software.

- Compare the text output of the profiler for each release
  -What are the top 5 functions for CPU usage.

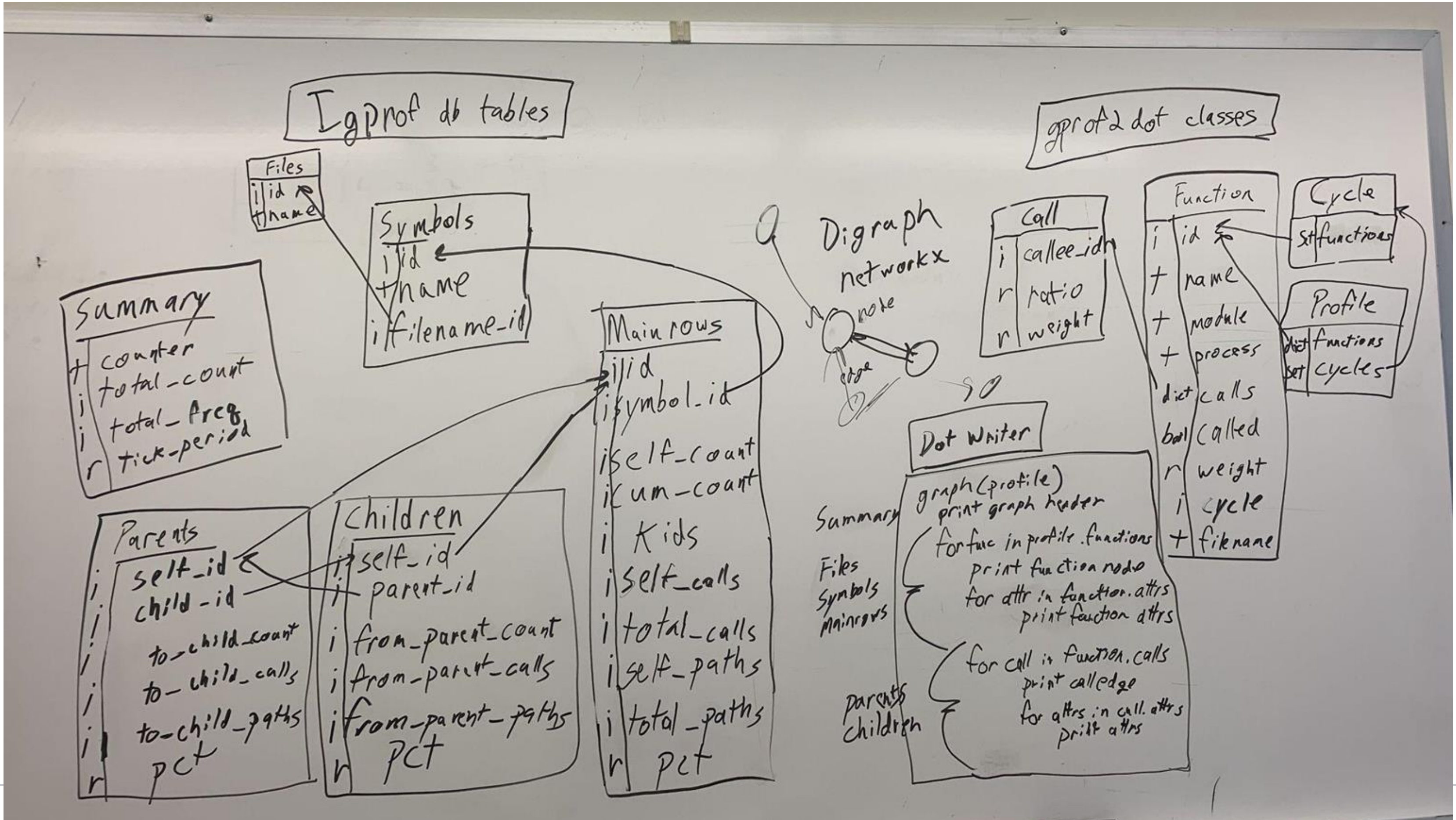
Method

- Connect to the CMSLPC cluster.
- Set up the CMS environment and Vtune profiler for every session.
- Create a CMSSW integration build release project area in the nobackup directory.

The following steps are done for Run 3 and HL LHC

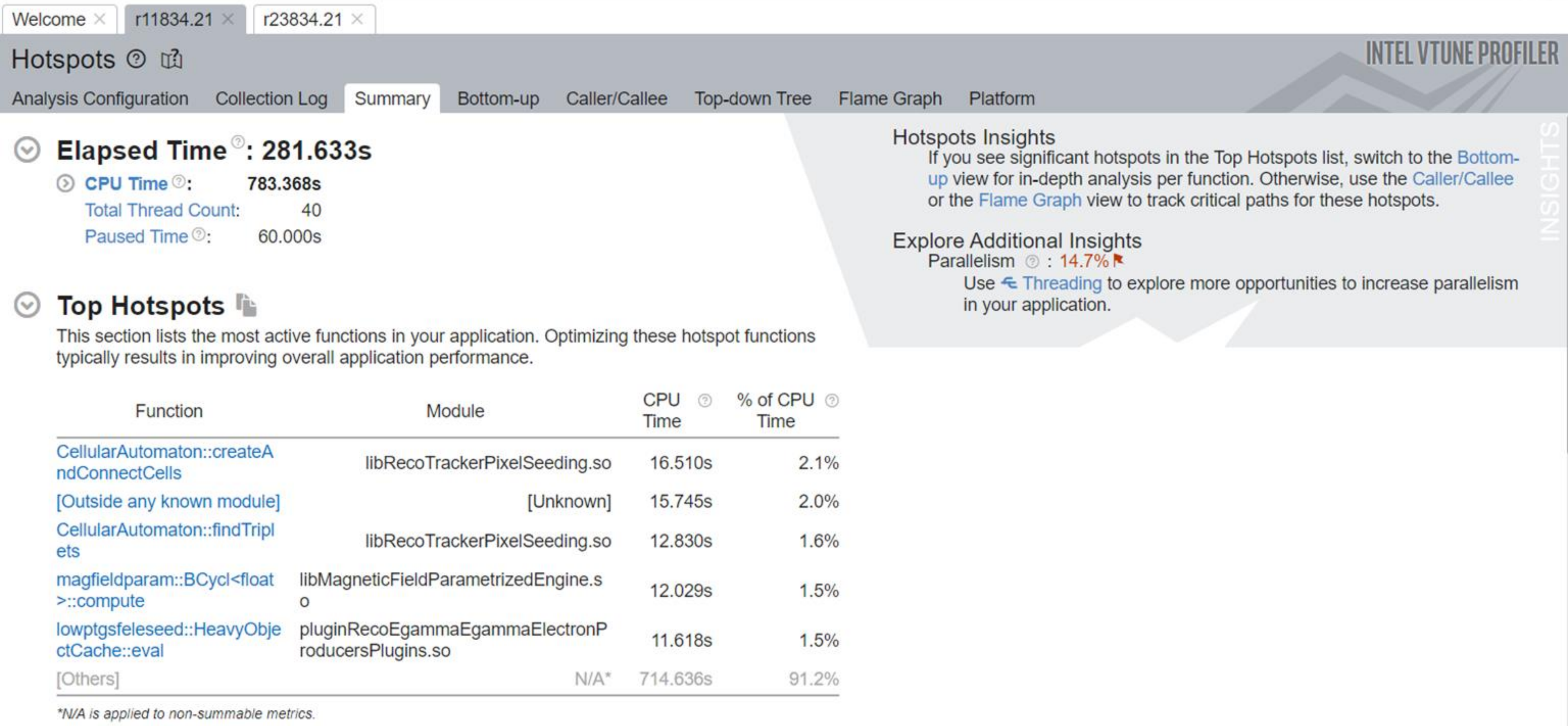
- Copy the configuration files necessary for a reconstruction job for {insert name of workflow}.
- Check that vtune created the profile.
- Generate a vtune hotspots report to get the top functions by CPU usage.
- Generate a vtune gprof_cc report to the callgraph of reconstruction.
- Generate a Vtune call stacks report to get the call stacks of reconstruction
- Generate a gprof2dot dump of the gprof_cc text report
- Get the igprof reports directly.

Process



Igprof db tables
Files
i id
t name
Symbols
i id
t name
i filename_id
Summary
t counter
i total_count
i total_freq
r tick_period
Mainrows
i id
i symbol_id
i self_count
i cum_count
i Kids
i self_calls
i total_calls
i self_paths
i total_paths
r pct
Parents
i self_id
i child_id
i to_child_count
i to_child_calls
i to_child_paths
r pct
Children
i self_id
i parent_id
i from_parent_count
i from_parent_calls
i from_parent_paths
r pct
Digraph
networkx
node
edge
gprof2dot classes
Call
i callee_id
r ratio
r weight
Function
i id
t name
t module
t process
dict calls
bool called
r weight
i cycle
t filename
Cycle
set functions
Profile
dict functions
set cycles
Dot Writer
graph(profile)
print graph header
for func in profile.functions
print function node
for attr in function.attrs
print function attrs
for call in function.calls
print calledge
for attrs in call.attrs
print attrs
Summary
Files
Symbols
Mainrows
Parents
Children

Results: Vtune Hotspots Report



Elapsed Time: 281.633s

CPU Time:	783.368s
Total Thread Count:	40
Paused Time:	60.000s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
CellularAutomaton::createAndConnectCells	libRecoTrackerPixelSeeding.so	16.510s	2.1%
[Outside any known module]	[Unknown]	15.745s	2.0%
CellularAutomaton::findTriplets	libRecoTrackerPixelSeeding.so	12.830s	1.6%
magfieldparam::BCycl<float>::compute	libMagneticFieldParametrizedEngine.so	12.029s	1.5%
lowptgsfeleseed::HeavyObjectCache::eval	pluginRecoEgammaEgammaElectronProducersPlugins.so	11.618s	1.5%
[Others]	N/A*	714.636s	91.2%

*N/A is applied to non-summable metrics.

Hotspots Insights

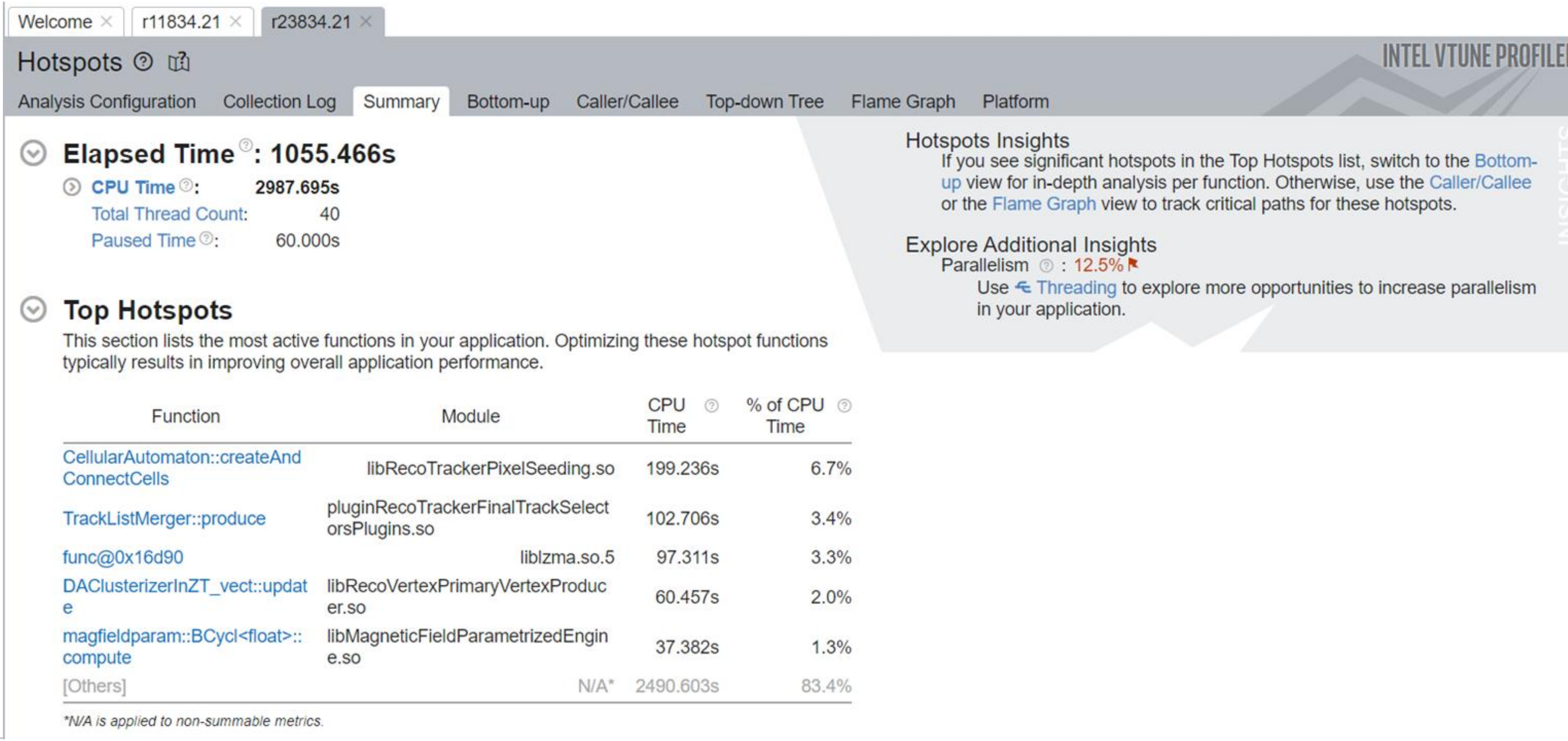
If you see significant hotspots in the Top Hotspots list, switch to the Bottom-up view for in-depth analysis per function. Otherwise, use the Caller/Callee or the Flame Graph view to track critical paths for these hotspots.

Explore Additional Insights

Parallelism : 14.7%

Use Threading to explore more opportunities to increase parallelism in your application.

Results: Vtune Hotspots report



Elapsed Time: 1055.466s

CPU Time:	2987.695s
Total Thread Count:	40
Paused Time:	60.000s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
CellularAutomaton::createAndConnectCells	libRecoTrackerPixelSeeding.so	199.236s	6.7%
TrackListMerger::produce	pluginRecoTrackerFinalTrackSelectorsPlugins.so	102.706s	3.4%
func@0x16d90	liblzma.so.5	97.311s	3.3%
DAClusterizerInZT_vect::update	libRecoVertexPrimaryVertexProducer.so	60.457s	2.0%
magfieldparam::BCycl<float>::compute	libMagneticFieldParametrizedEngine.so	37.382s	1.3%
[Others]	N/A*	2490.603s	83.4%

*N/A is applied to non-summable metrics.

Hotspots Insights

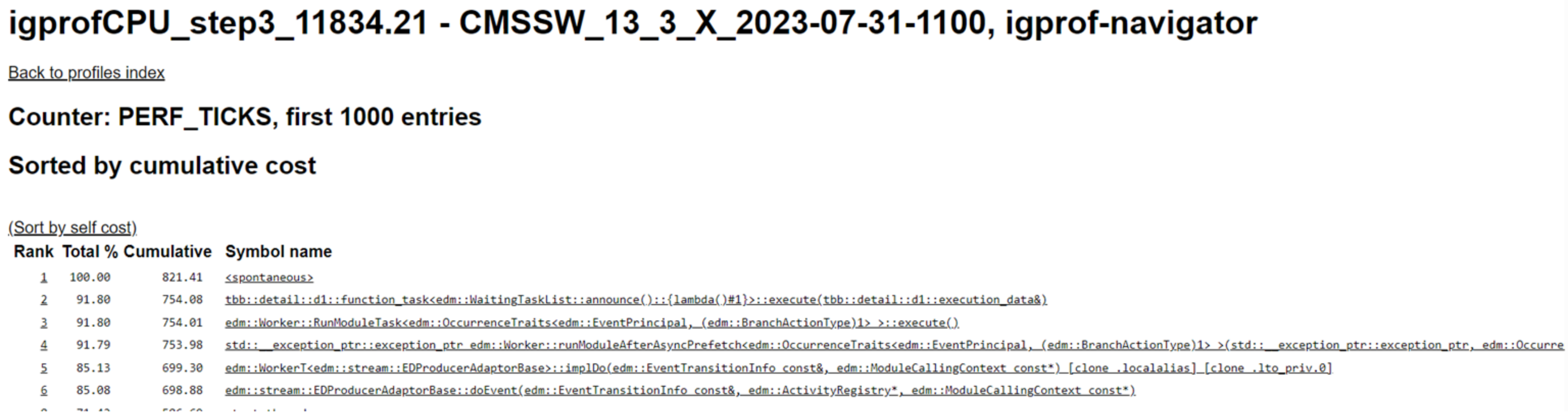
If you see significant hotspots in the Top Hotspots list, switch to the Bottom-up view for in-depth analysis per function. Otherwise, use the Caller/Callee or the Flame Graph view to track critical paths for these hotspots.

Explore Additional Insights

Parallelism : 12.5%

Use Threading to explore more opportunities to increase parallelism in your application.

Results: Vtune gprof_cc report



igprofCPU_step3_11834.21 - CMSSW_13_3_X_2023-07-31-1100, igprof-navigator

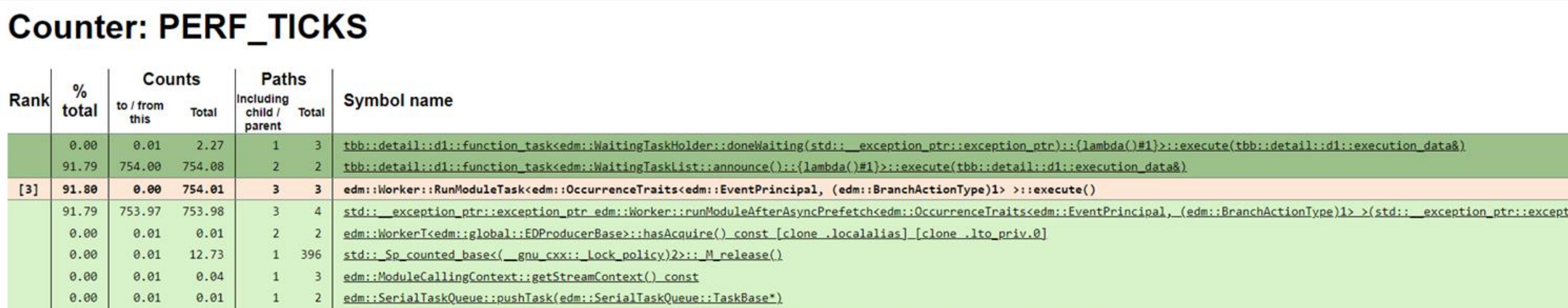
Back to profiles index

Counter: PERF_TICKS, first 1000 entries

Sorted by cumulative cost

(Sort by self cost)

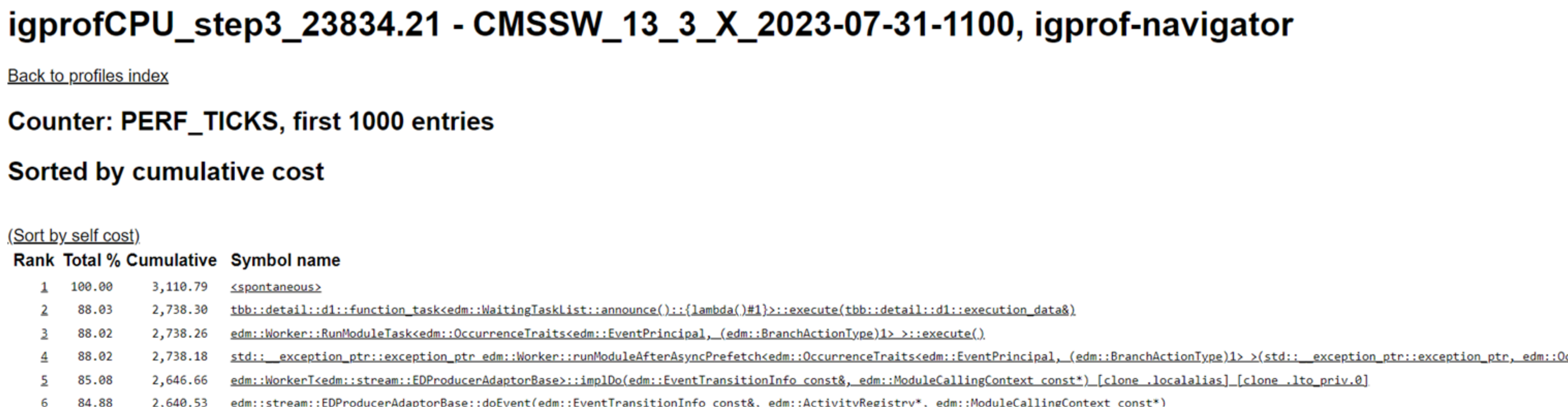
Rank	Total %	Cumulative	Symbol name
1	100.00	821.41	<spontaneous>
2	91.80	754.08	tbb::detail::d1::function_task<edm::WaitingTaskList::announce()::{lambda()#1}>::execute(tbb::detail::d1::execution_data&)
3	91.80	754.01	edm::Worker::RunModuleTask<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >::execute()
4	91.79	753.98	std::__exception_ptr::exception_ptr edm::Worker::runModuleAfterAsyncPrefetch<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >(std::__exception_ptr::exception_ptr, edm::Occurre
5	85.13	699.30	edm::WorkerT<edm::stream::EDProducerAdaptorBase>::implDo(edm::EventTransitionInfo const&, edm::ModuleCallingContext const*) [clone .localalias] [clone .lto_priv.0]
6	85.08	698.88	edm::stream::EDProducerAdaptorBase::doEvent(edm::EventTransitionInfo const&, edm::ActivityRegistry*, edm::ModuleCallingContext const*)



Counter: PERF_TICKS

Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	0.00	0.01	2.27	1	3	tbb::detail::d1::function_task<edm::WaitingTaskHolder::doneWaiting(std::__exception_ptr::exception_ptr)::{lambda()#1}>::execute(tbb::detail::d1::execution_data&)
	91.79	754.00	754.08	2	2	tbb::detail::d1::function_task<edm::WaitingTaskList::announce()::{lambda()#1}>::execute(tbb::detail::d1::execution_data&)
[3]	**91.80**	**0.00**	**754.01**	**3**	**3**	**edm::Worker::RunModuleTask<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >::execute()**
	91.79	753.97	753.98	3	4	std::__exception_ptr::exception_ptr edm::Worker::runModuleAfterAsyncPrefetch<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >(std::__exception_ptr::except
	0.00	0.01	0.01	2	2	edm::WorkerT<edm::global::EDProducerBase>::hasAcquire() const [clone .localalias] [clone .lto_priv.0]
	0.00	0.01	12.73	1	396	std::_Sp_counted_base<(__gnu_cxx::_Lock_policy)2>::_M_release()
	0.00	0.01	0.04	1	3	edm::ModuleCallingContext::getStreamContext() const
	0.00	0.01	0.01	1	2	edm::SerialTaskQueue::pushTask(edm::SerialTaskQueue::TaskBase*)

Results: Vtune gprof_cc report



igprofCPU_step3_23834.21 - CMSSW_13_3_X_2023-07-31-1100, igprof-navigator

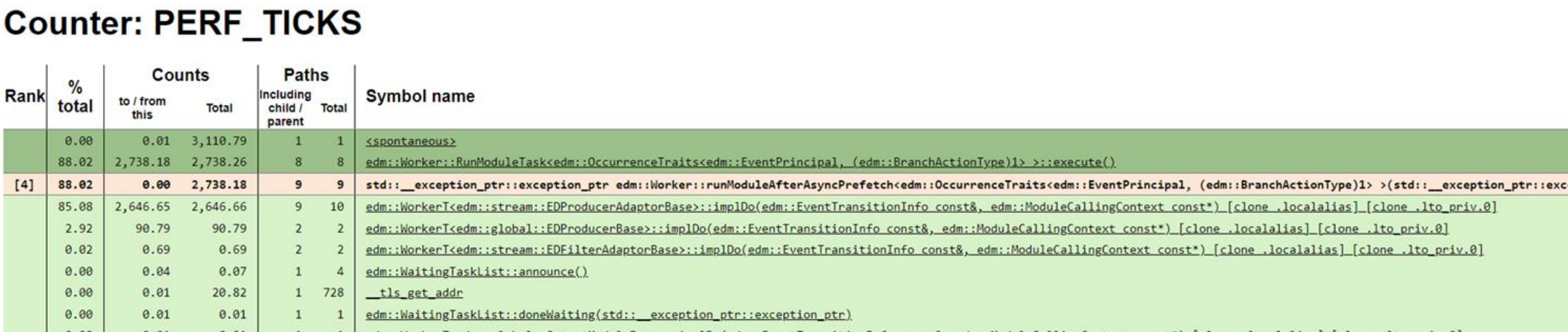
Back to profiles index

Counter: PERF_TICKS, first 1000 entries

Sorted by cumulative cost

(Sort by self cost)

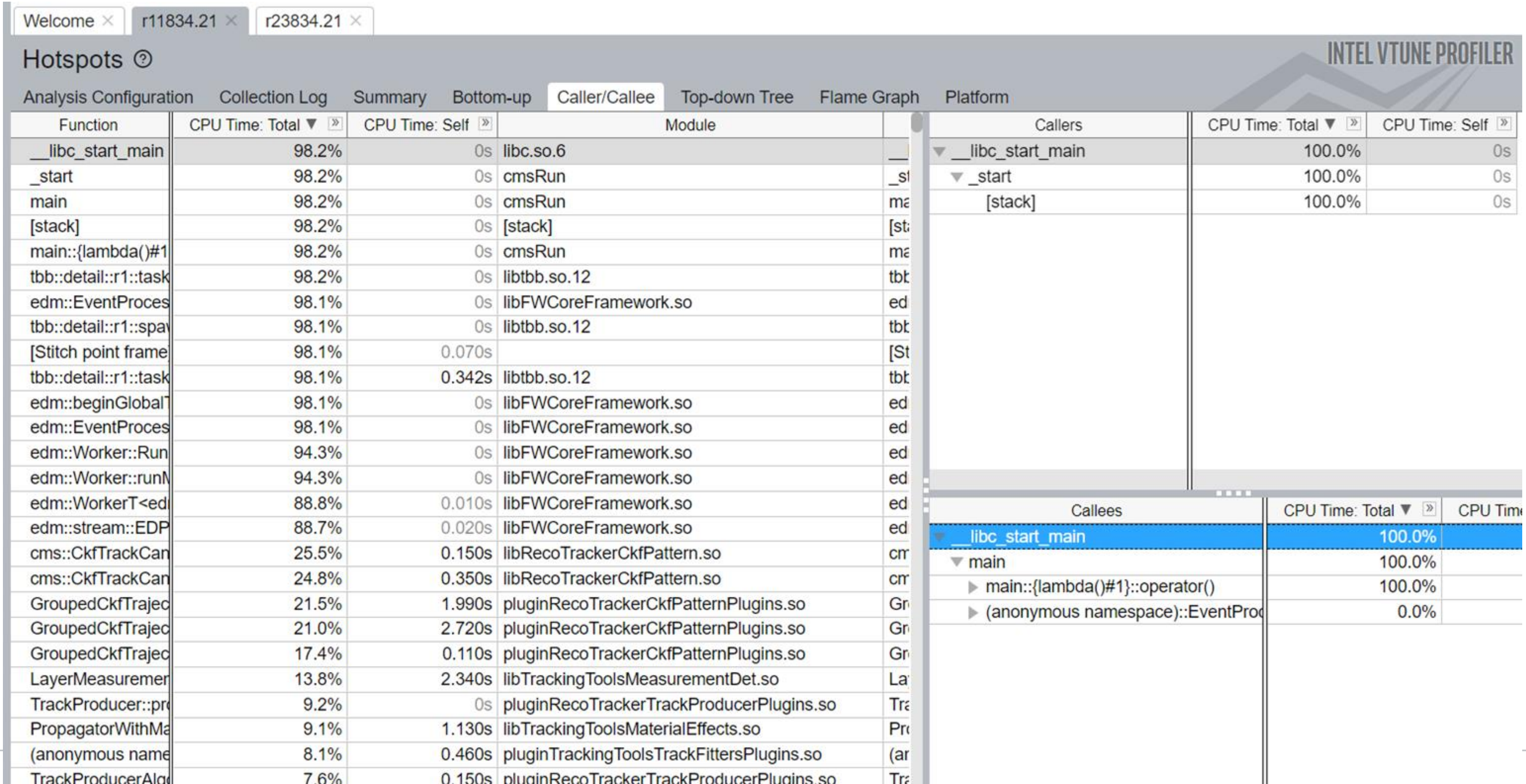
Rank	Total %	Cumulative	Symbol name
1	100.00	3,110.79	<spontaneous>
2	88.03	2,738.30	tbb::detail::d1::function_task<edm::WaitingTaskList::announce()::{lambda()#1}>::execute(tbb::detail::d1::execution_data&)
3	88.02	2,738.26	edm::Worker::RunModuleTask<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >::execute()
4	88.02	2,738.18	std::__exception_ptr::exception_ptr edm::Worker::runModuleAfterAsyncPrefetch<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >(std::__exception_ptr::exception_ptr, edm::Oc
5	85.08	2,646.66	edm::WorkerT<edm::stream::EDProducerAdaptorBase>::implDo(edm::EventTransitionInfo const&, edm::ModuleCallingContext const*) [clone .localalias] [clone .lto_priv.0]
6	84.88	2,640.53	edm::stream::EDProducerAdaptorBase::doEvent(edm::EventTransitionInfo const&, edm::ActivityRegistry*, edm::ModuleCallingContext const*)



Counter: PERF_TICKS

Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	0.00	0.01	3,110.79	1	1	<spontaneous>
	88.02	2,738.18	2,738.26	8	8	edm::Worker::RunModuleTask<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >::execute()
[4]	88.02	0.00	2,738.18	9	9	std::__exception_ptr::exception_ptr edm::Worker::runModuleAfterAsyncPrefetch<edm::OccurrenceTraits<edm::EventPrincipal, (edm::BranchActionType)1> >(std::__exception_ptr::exc
	85.08	2,646.65	2,646.66	9	10	edm::WorkerT<edm::stream::EDProducerAdaptorBase>::implDo(edm::EventTransitionInfo const&, edm::ModuleCallingContext const*) [clone .localalias] [clone .lto_priv.0]
	2.92	90.79	90.79	2	2	edm::WorkerT<edm::global::EDProducerBase>::implDo(edm::EventTransitionInfo const&, edm::ModuleCallingContext const*) [clone .localalias] [clone .lto_priv.0]
	0.02	0.69	0.69	2	2	edm::WorkerT<edm::stream::EDFilterAdaptorBase>::implDo(edm::EventTransitionInfo const&, edm::ModuleCallingContext const*) [clone .localalias] [clone .lto_priv.0]
	0.00	0.04	0.07	1	4	edm::WaitingTaskList::announce()
	0.00	0.01	20.82	1	728	__tls_get_addr
	0.00	0.01	0.01	1	1	edm::WaitingTaskList::doneWaiting(std::__exception_ptr::exception_ptr)

Results: Vtune call stacks report

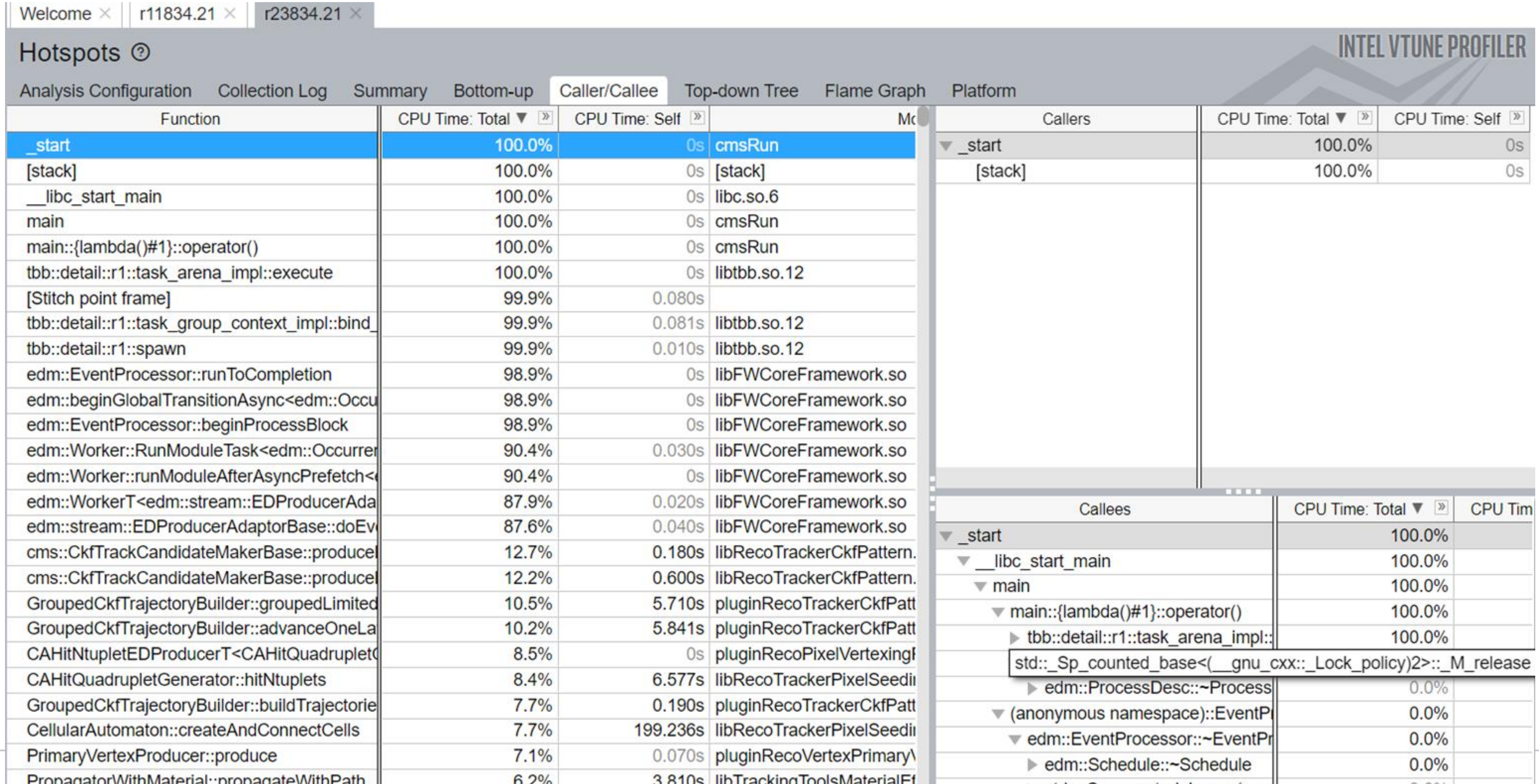


Function	CPU Time: Total ▼	CPU Time: Self	Module	
__libc_start_main	98.2%	0s	libc.so.6	__
_start	98.2%	0s	cmsRun	_st
main	98.2%	0s	cmsRun	ma
[stack]	98.2%	0s	[stack]	[st
main::{lambda()#1	98.2%	0s	cmsRun	ma
tbb::detail::r1::task	98.2%	0s	libtbb.so.12	tbb
edm::EventProces	98.1%	0s	libFWCoreFramework.so	ed
tbb::detail::r1::spa	98.1%	0s	libtbb.so.12	tbb
[Stitch point frame]	98.1%	0.070s		[St
tbb::detail::r1::task	98.1%	0.342s	libtbb.so.12	tbb
edm::beginGlobal	98.1%	0s	libFWCoreFramework.so	ed
edm::EventProces	98.1%	0s	libFWCoreFramework.so	ed
edm::Worker::Run	94.3%	0s	libFWCoreFramework.so	ed
edm::Worker::runM	94.3%	0s	libFWCoreFramework.so	ed
edm::WorkerT<edi	88.8%	0.010s	libFWCoreFramework.so	ed
edm::stream::EDP	88.7%	0.020s	libFWCoreFramework.so	ed
cms::CkfTrackCan	25.5%	0.150s	libRecoTrackerCkfPattern.so	cm
cms::CkfTrackCan	24.8%	0.350s	libRecoTrackerCkfPattern.so	cm
GroupedCkfTrajec	21.5%	1.990s	pluginRecoTrackerCkfPatternPlugins.so	Gr
GroupedCkfTrajec	21.0%	2.720s	pluginRecoTrackerCkfPatternPlugins.so	Gr
GroupedCkfTrajec	17.4%	0.110s	pluginRecoTrackerCkfPatternPlugins.so	Gr
LayerMeasuremen	13.8%	2.340s	libTrackingToolsMeasurementDet.so	La
TrackProducer::pr	9.2%	0s	pluginRecoTrackerTrackProducerPlugins.so	Tra
PropagatorWithMa	9.1%	1.130s	libTrackingToolsMaterialEffects.so	Pro
(anonymous name	8.1%	0.460s	pluginTrackingToolsTrackFittersPlugins.so	(an
TrackProducerAlg	7.6%	0.150s	pluginRecoTrackerTrackProducerPlugins.so	Tra

Callers	CPU Time: Total ▼	CPU Time: Self
▼ __libc_start_main	100.0%	0s
▼ _start	100.0%	0s
[stack]	100.0%	0s

Callees	CPU Time: Total ▼	CPU Time
▼ __libc_start_main	100.0%	
▼ main	100.0%	
▶ main::{lambda()#1}::operator()	100.0%	
▶ (anonymous namespace)::EventProc	0.0%	

Results: Vtune call stacks report

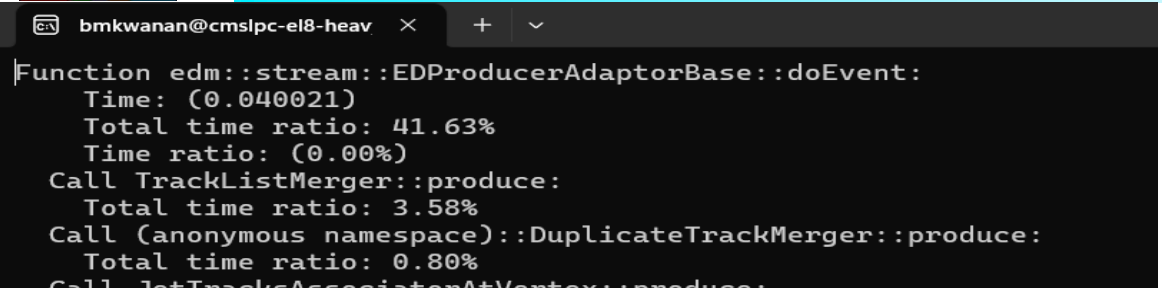


Function	CPU Time: Total ▼	CPU Time: Self	Mc
_start	100.0%	0s	cmsRun
[stack]	100.0%	0s	[stack]
__libc_start_main	100.0%	0s	libc.so.6
main	100.0%	0s	cmsRun
main::{lambda()#1}::operator()	100.0%	0s	cmsRun
tbb::detail::r1::task_arena_impl::execute	100.0%	0s	libtbb.so.12
[Stitch point frame]	99.9%	0.080s	
tbb::detail::r1::task_group_context_impl::bind_	99.9%	0.081s	libtbb.so.12
tbb::detail::r1::spawn	99.9%	0.010s	libtbb.so.12
edm::EventProcessor::runToCompletion	98.9%	0s	libFWCoreFramework.so
edm::beginGlobalTransitionAsync<edm::Occu	98.9%	0s	libFWCoreFramework.so
edm::EventProcessor::beginProcessBlock	98.9%	0s	libFWCoreFramework.so
edm::Worker::RunModuleTask<edm::Occurre	90.4%	0.030s	libFWCoreFramework.so
edm::Worker::runModuleAfterAsyncPrefetch<	90.4%	0s	libFWCoreFramework.so
edm::WorkerT<edm::stream::EDProducerAda	87.9%	0.020s	libFWCoreFramework.so
edm::stream::EDProducerAdaptorBase::doEv	87.6%	0.040s	libFWCoreFramework.so
cms::CkfTrackCandidateMakerBase::produceI	12.7%	0.180s	libRecoTrackerCkfPattern.
cms::CkfTrackCandidateMakerBase::produceI	12.2%	0.600s	libRecoTrackerCkfPattern.
GroupedCkfTrajectoryBuilder::groupedLimited	10.5%	5.710s	pluginRecoTrackerCkfPatt
GroupedCkfTrajectoryBuilder::advanceOneLa	10.2%	5.841s	pluginRecoTrackerCkfPatt
CAHitNtupletEDProducerT<CAHitQuadrupletC	8.5%	0s	pluginRecoPixelVertexingI
CAHitQuadrupletGenerator::hitNtuplets	8.4%	6.577s	libRecoTrackerPixelSeedi
GroupedCkfTrajectoryBuilder::buildTrajectorie	7.7%	0.190s	pluginRecoTrackerCkfPatt
CellularAutomaton::createAndConnectCells	7.7%	199.236s	libRecoTrackerPixelSeedi
PrimaryVertexProducer::produce	7.1%	0.070s	pluginRecoVertexPrimaryV
PropagatorWithMaterial::propagateWithPath	6.2%	3.810s	libTrackingToolsMaterialEf

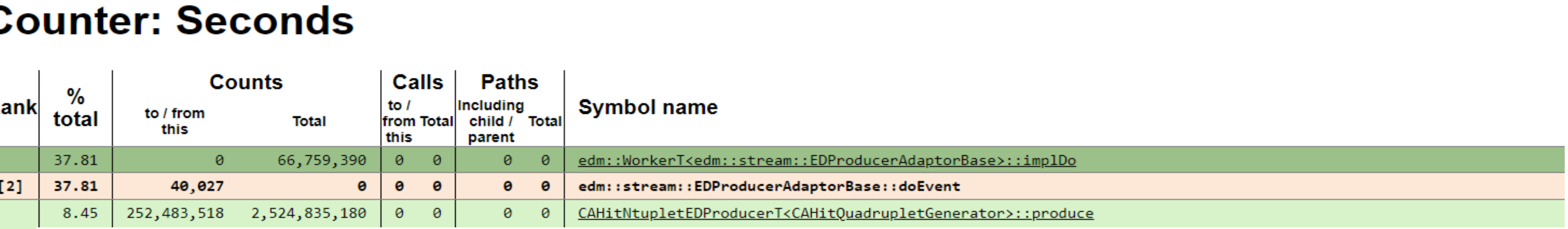
Callers	CPU Time: Total ▼	CPU Time: Self
▼ _start	100.0%	0s
[stack]	100.0%	0s

Callees	CPU Time: Total ▼	CPU Tim
▼ _start	100.0%	
▼ __libc_start_main	100.0%	
▼ main	100.0%	
▼ main::{lambda()#1}::operator()	100.0%	
▶ tbb::detail::r1::task_arena_impl::	100.0%	
std::_Sp_counted_base<(__gnu_cxx::_Lock_policy)2>::_M_release		
▶ edm::ProcessDesc::~Process	0.0%	
▼ (anonymous namespace)::EventPr	0.0%	
▼ edm::EventProcessor::~EventPr	0.0%	
▶ edm::Schedule::~Schedule	0.0%	

Results: gprof2dot dump of the gprof_cc text report

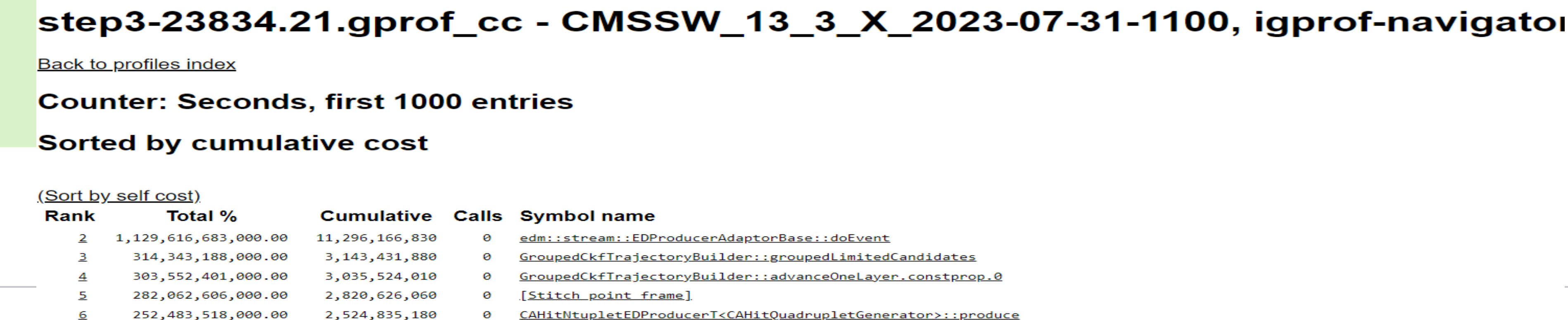


```
Function edm::stream::EDProducerAdaptorBase::doEvent:
    Time: (0.040021)
    Total time ratio: 41.63%
    Time ratio: (0.00%)
  Call TrackListMerger::produce:
    Total time ratio: 3.58%
  Call (anonymous namespace)::DuplicateTrackMerger::produce:
    Total time ratio: 0.80%
  Call JetTracksAssociatorAtVertex::produce:
```



Counter: Seconds

ank	% total	Counts		Calls		Paths		Symbol name
		to / from this	Total	to / from this	Total	Including child / parent	Total	
	37.81	0	66,759,390	0	0	0	0	edm::WorkerT<edm::stream::EDProducerAdaptorBase>::implDo
[2]	37.81	40,027	0	0	0	0	0	edm::stream::EDProducerAdaptorBase::doEvent
	8.45	252,483,518	2,524,835,180	0	0	0	0	CAHitNtupletEDProducerT<CAHitQuadrupletGenerator>::produce



step3-23834.21.gprof_cc - CMSSW_13_3_X_2023-07-31-1100, igprof-navigator

Back to profiles index

Counter: Seconds, first 1000 entries

Sorted by cumulative cost

(Sort by self cost)

Rank	Total %	Cumulative	Calls	Symbol name
2	1,129,616,683,000.00	11,296,166,830	0	edm::stream::EDProducerAdaptorBase::doEvent
3	314,343,188,000.00	3,143,431,880	0	GroupedCkfTrajectoryBuilder::groupedLimitedCandidates
4	303,552,401,000.00	3,035,524,010	0	GroupedCkfTrajectoryBuilder::advanceOneLayer.constprop.0
5	282,062,606,000.00	2,820,626,060	0	[Stitch point frame]
6	252,483,518,000.00	2,524,835,180	0	CAHitNtupletEDProducerT<CAHitQuadrupletGenerator>::produce

Discussion

- Parallelism refers to how efficiently the code is threaded and the identification of threading issues that impact performance. In simply terms, parallelism refers to what vtune could note of the threads in that % of time. According to the results, Run 3 recorded Parallelism is 14.7% and for HL LHC is 12.5%.
- Run 3 and HL LHC have two identical functions, namely Cellular Automation::createAndConnectCells and magfieldparam::BCycl<float>::compute (but they have different CPU time).
- HL LHC has more tracks hence the functions consume more CPU time compared to Run 3.

What I learnt

- Segmentation faults can happen in IB and that’s the best place to catch them before they go to production.
- Importance of Critical thinking.
- Attention to detail as some programming languages are case sensitive.
- Working in Python, SQL, Pandas- Python Data Analysis Library.
- I’m interested in Data Science.
- Susy is not only a Hebrew girl name but an abbreviation for a Super symmetrical particle (SUSY).
- I just know of the tip of the iceberg when it comes to Physics.

References

Introduction (intel.com)

IgProf, The Ignominous Profiler

Configure VTune parallelism? - Intel Community