Code that Outperforms

Intel® oneAPI Analyzers

Intel VTune Profiler and Intel Advisor



Agenda



Intel[®] oneAPI Overview

Introduction to the Intel oneAPI Base and HPC Toolkits



Intel VTune Profiler and Intel Advisor Overview

Overview of the oneAPI analyzers



MandelbrotOMP Sample Configuration

Configure the sample used in the exercises



CPU Profiling Exercises

Running the sample on midway3 with Intel Advisor and Intel VTune Profiler.



GPU Profiling Demo

Demo profiling the iso3dfd sample on Intel DevCloud with Intel Advisor and Intel VTune Profiler

Programming Challenges for Multiple Architectures

Growth in specialized workloads

Variety of data-centric hardware required

Separate programming models and toolchains for each architecture are required today

Software development complexity limits freedom of architectural choice

Applicat	ion Workloads I	Need Diverse H	ardware
(\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	Vector	Spatial	Matrix
	Middleware &	Frameworks	
		-	
CPU programming model	GPU programming model	FPGA programming model	Other accel. programming models
programming	programming	programming	programming

3

ONEAPI One Programming Mod

One Programming Model for Multiple Architectures and Vendors

Freedom to Make Your Best Choice

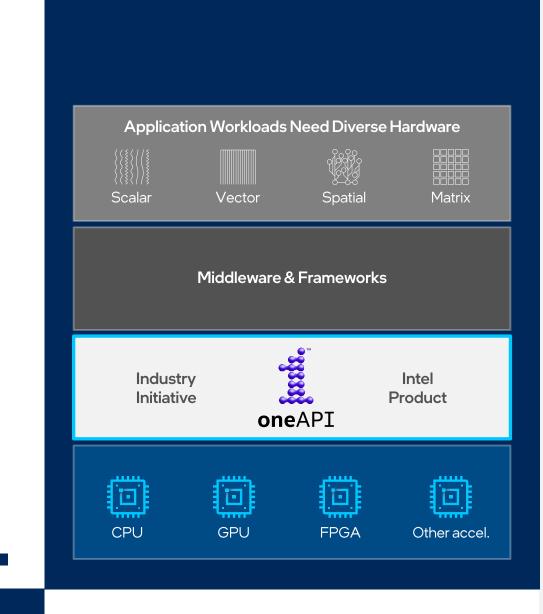
Choose the best accelerated technology the software doesn't decide for you

Realize all the Hardware Value

Performance across CPU, GPUs, FPGAs, and other accelerators

Develop & Deploy Software with Peace of Mind

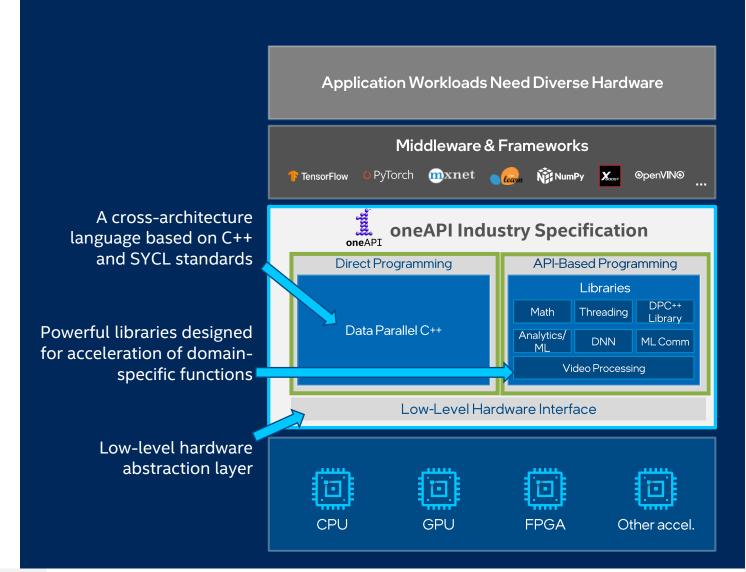
- Open industry standards provide a safe, clear path to the future
- Compatible with existing languages and programming models including C++, Python, SYCL, OpenMP, Fortran, and MPI



oneAPI Industry Initiative Break the Chains of Proprietary Lock-in

Open to promote community and industry collaboration

Enables code reuse across architectures and vendors



The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary programming models

Data Parallel C++

Standards-based, Cross-architecture Language

DPC++ = ISO C++ and Khronos SYCL and community extensions

Freedom of Choice: Future-Ready Programming Model

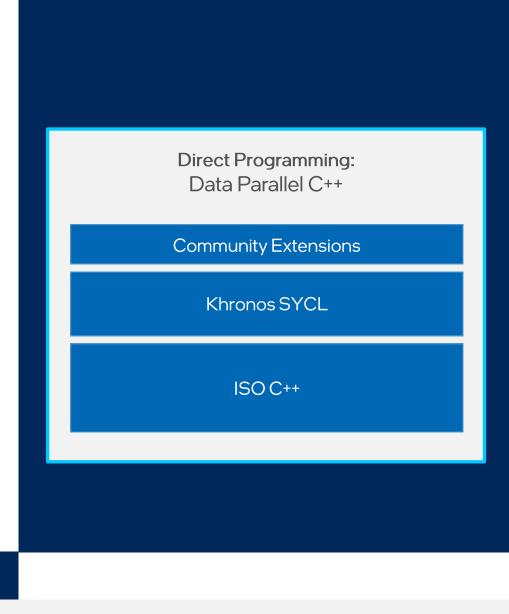
- Allows code reuse across hardware targets
- Permits custom tuning for a specific accelerator
- Open, cross-industry alternative to proprietary language

DPC++ = ISO C++ and Khronos SYCL and community extensions

- Delivers C++ productivity benefits, using common, familiar C and C++ constructs
- Adds SYCL from the Khronos Group for data parallelism and heterogeneous programming

Community Project Drives Language Enhancements

- Provides extensions to simplify data parallel programming
- Continues evolution through open and cooperative development



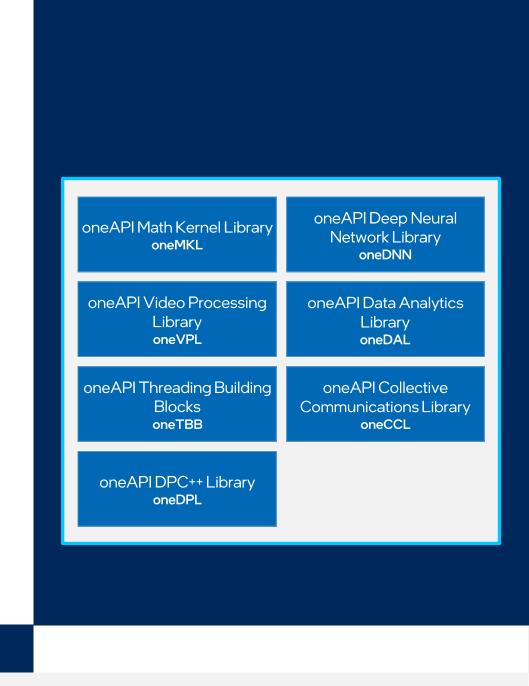
Powerful oneAPI Libraries

Realize all the Hardware Value

Designed for acceleration of key domain-specific functions

Freedom of Choice

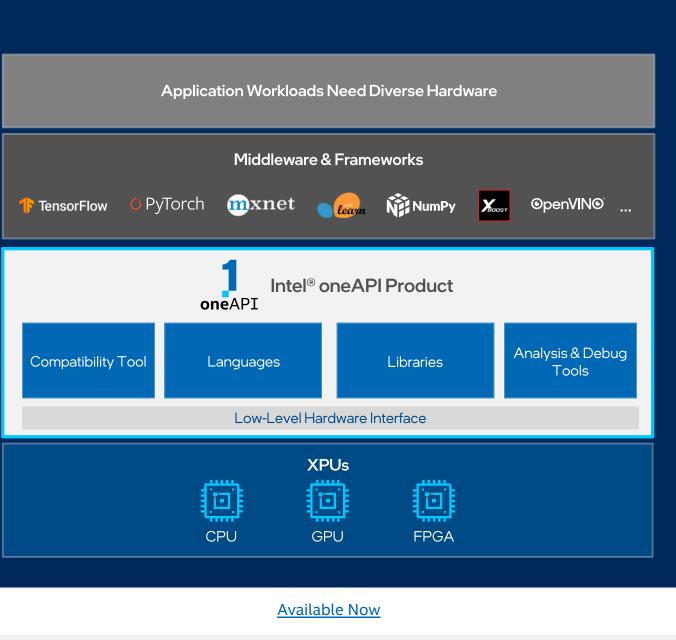
Pre-optimized for each target platform for maximum performance



Intel[®] oneAPI Product Built on Intel's Rich Heritage of CPU Tools Expanded to XPUs

A complete set of advanced compilers, libraries, and porting, analysis and debugger tools

- Accelerates compute by exploiting cutting-edge hardware features
- Interoperable with existing programming models and code bases (C++, Fortran, Python, OpenMP, etc.), developers can be confident that existing applications work seamlessly with oneAPI
- Eases transitions to new systems and accelerators—using a single code base frees developers to invest more time on innovation



Latest version is 2021.1

Visit <u>software.intel.com/oneapi</u> for more details Some capabilities may differ per architecture and custom-tuning will still be required. Other accelerators to be supported in the future.

8

Intel[®] oneAPI Base Toolkit Accelerate Data-centric Workloads

A core set of core tools and libraries for developing high-performance applications on Intel[®] CPUs, GPUs, and FPGAs.

Who Uses It?

- A broad range of developers across industries
- Add-on toolkit users since this is the base for all toolkits

Top Features/Benefits

- Data Parallel C++ compiler, library and analysis tools
- DPC++ Compatibility tool helps migrate existing code written in CUDA
- Python distribution includes accelerated scikit-learn, NumPy, SciPy libraries
- Optimized performance libraries for threading, math, data analytics, deep learning, and video/image/signal processing

Intel® oneAPI Base Toolkit Direct Programming Intel® oneAPI DPC++/C++ Compiler Intel® DPC++ Compatibility Tool Intel® oneAPI Math Kernel Library-oneMKL

Intel[®] Distribution for Python

Intel[®] FPGA Add-on

for oneAPI Base Toolkit

Intel® oneAPI Data Analytics Library - oneDAL

Intel® oneAPI Threading Building Blocks - oneTBB

Intel® oneAPI Video Processing Library - oneVPL

Intel® oneAPI Collective Communications Library oneCCL

Intel® oneAPI Deep Neural Network Library - oneDNN

Intel[®] Integrated Performance Primitives - Intel[®] IPP



Analysis & debug Tools

Intel[®] VTune[™] Profiler

Intel[®] Advisor

Intel[®] Distribution for GDB

9

Intel® oneAPI Tools for HPC Intel® OneAPI HPC Toolkit

Deliver Fast Applications that Scale

What is it?

A toolkit that adds to the Intel[®] oneAPI Base Toolkit for building high-performance, scalable parallel code on C++, SYCL, Fortran, OpenMP & MPI from enterprise to cloud, and HPC to AI applications.

Who needs this product?

- OEMs/ISVs
- C++, Fortran, OpenMP, MPI Developers

Why is this important?

- Accelerate performance on Intel[®] Xeon[®] and Core[™] Processors and Intel[®] Accelerators
- Deliver fast, scalable, reliable parallel code with less effort built on industry standards

Intel[®] oneAPI Base & HPC Toolkits

Direct Programming	API-Based Programming	Analysis & debug Tools
Intel® C++ Compiler Classic	Intel [®] MPI Library	Intel [®] Inspector
Intel® Fortran Compiler Classic	Intel® oneAPI DPC++ Library oneDPL	Intel® Trace Analyzer & Collector
Intel [®] Fortran Compiler	Intel® oneAPI Math Kernel Library - oneMKL	Intel® Cluster Checker
Intel® oneAPI DPC++/C++ Compiler	Intel® oneAPI Data Analytics Library - oneDAL	Intel® VTune™ Profiler
Intel® DPC++ Compatibility Tool	Intel® oneAPI Threading Building Blocks - oneTBB	Intel [®] Advisor
Intel [®] Distribution for Python	Intel® oneAPI Video Processing Library - oneVPL	Intel® Distribution for GDB
Intel® FPGA Add-on for oneAPI Base Toolkit	Intel® oneAPI Collective Communications Library oneCCL	intel
	Intel® oneAPI Deep Neural Network Library - oneDNN	1
Intel® oneAPI HPC Toolkit + Intel® oneAPI Base Toolkit	Intel® Integrated Performance Primitives – Intel® IPP	OneAPI

Intel Analysis Tools for GPU Compute Analysis

Intel[®] Advisor

Offload Advisor

- Identify high-impact opportunities to offload
- Detect bottlenecks and key bounding factors
- Get your code ready even before you have the hardware by modeling performance, headroom, and bottlenecks

Roofline Analysis

- See performance headroom against hardware limitations
- Determine performance optimization strategy by identifying bottlenecks and which optimizations will pay off the most
- Visualize optimization progress

Flow Graph Analyzer

• Visualize your CPU/GPU code and get recommendations for the CPU device

Intel[®] VTune[™] Profiler

Offload Performance Tuning

- Explore code execution on your platform's various CPU and GPU cores
- Correlate CPU and GPU activity
- Identify whether your application is GPU- or CPU-bound

GPU Compute/Media Hotspots

- Analyze the most time-consuming GPU kernels, characterize GPU usage based on GPU hardware metrics
- GPU code performance at the source-line level and kernel-assembly level

Intel[®] VTune[™] Profiler Overview

Optimize Performance Intel® VTune™ Profiler

Get the Right Data to Find Bottlenecks

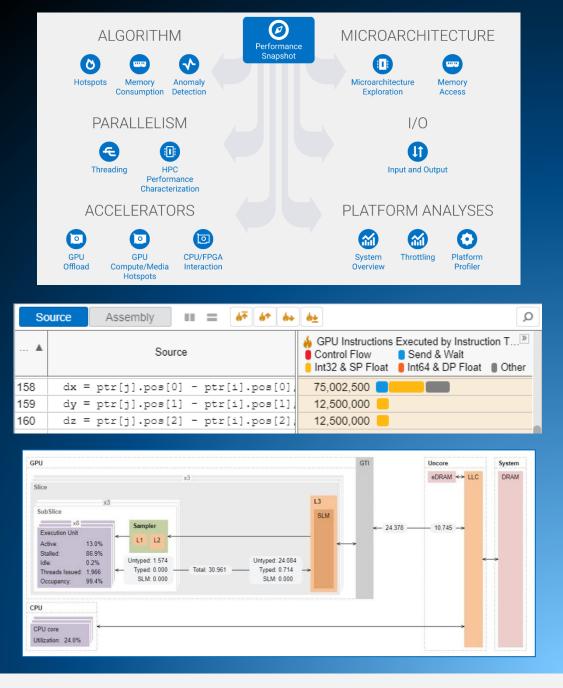
- A suite of profiling for CPU, GPU, FPGA, threading, memory, cache, storage, offload, power...
- Application or system-wide analysis
- DPC++, C, C++, Fortran, Python*, Go*, Java*, or a mix
- Linux, Windows, FreeBSD, Android, Yocto and more
- Containers and VMs

Analyze Data Faster

- Collect data HW/SW sampling and tracing w/o recompilation
- See results on your source, in architecture diagrams, as a histogram, on a timeline...
- Filter and organize data to find answers

Work Your Way

- User interface or command line
- Profile locally and remotely
- GUI (desktop or web) or command line

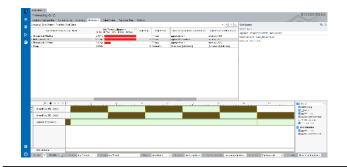


Rich Set of Profiling Capabilities

	Contact Inc.	THE DEALERS CONCLUSING NO.	PARTINE TRACTORY PRAT			Col Name		0.0
■ 3.5	- opening						STERN ARE LEADED.	
						THE REPORT OF THE PARTY		
							Note Alam grant close	dependence Marth
							where the second second	
						erroriet: polities m	meren de con marches, en	alexander construction
						second according to second	and an only the other all	and an a fill of a standard
						constitution and	www.com/witebilit.com/wite	of contractions and contractions
							100-Combile Montal 8	
						CONTRACTOR OF ANY		
	when the state of a state of a		_			Internal Automation	6	
press accr. etc.								
a I De Neder Calentin	artue:		- 28					8 (8.4
a I De Neder Calentin	- = #			1		· · · · · · · · · · · · · · · · · · ·		8
анциндин Би Алан стан ст	- = #					¥		Salary Salary
анциндин Би Алан стан ст	- = #				12	· · · · · · · · · · · · · · · · · · ·		Salay a Salay na Salay na
анциндин Би Алан стан ст	- = #				12	¥		Second Second
анцина, уна Би Са анг 1 анг 1	- = #		···· '¥	1		· · · · · · · · · · · · · · · · · · ·		Salay a Salay na Salay na
анциндин Би Алан стан ст	- = #					· · · · · · · · · · · · · · · · · · ·		Sabys Sabys
анциндин Би Алан стан ст	- = #					7		S How y S HOW Pa S Denne Commun C With April S OVER States S HOW The
анцина, уна Би Са анг 1 анг 1	- = #		···· 'Y			7		S How y S HOW Pa S Denne Commun C With April S OVER States S HOW The
анциндин Би Алан стан ст	- = #							S How y S HOW Pa S Denne Commun C With April S OVER States S HOW The

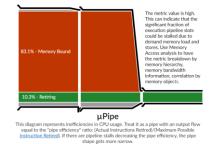
Algorithm Optimization

- ✓ Hotspots
- \checkmark Anomaly Detection
- ✓ Memory Consumption



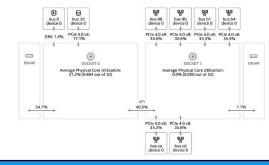
Parallelism

✓ Threading
 ✓ HPC Performance Characterization



Microarch.&Memory Bottlenecks

- ✓ Microarchitecture Exploration
- ✓ Memory Access



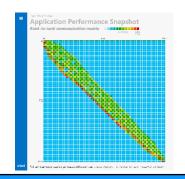
Platform & I/O

- ✓ Input and Output
- ✓ System Overview
- ✓ Platform Profiler



Accelerators / xPU

- ✓ GPU Offload
- ✓ GPU Compute / Media Hotspots
- ✓ CPU/FPGA Interaction



Multi-Node

✓ Application Performance Snapshot

Find Answers Fast

Intel[®] VTune[™] Profiler

Adjust Data Grouping

Function / Call Stack Source Function / Function / Call Stack

Sync Object / Function / Call Stack

Sync Object / Thread / Function / Call Stack ... (Partial list shown)

Double Click Function to View Source Click [>] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

> Zoom In And Filter On Selection Filter In by Selection Remove All Filters

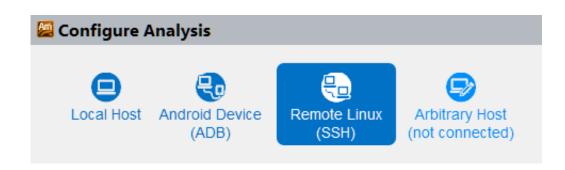
Grouping: Function / Call \$	Collection Log Summary Bottom-u Stack	p Caller/Callee	Top-dow	n Tree Platf		• 🛠 D		
		CPU	Time					
Function / Call Stack		>		Overhead Time				
Tuncton / Can Stack	Effective Time by Utilization ▼ Idle Poor Ok Ideal Over	Spin Time	Creation	Scheduling	Reduction	Atomics		
grid_intersect	4.0875	0s	0s	0s	0s	0s		
 sphere_intersect 	3.748s	0s	0s	0s	0s	0s		
grid_intersect	3.748s	0s	0s	0s	0s	0s		
intersect_objects	3.580s	0s	0s	0s	0s	0s		
▶ < grid_intersect ←	0.168s 📙	2.021s	0s	0s	0s	0s		
func@0x69e19df0	2.467s	Os	Os	0s	0s	0s		
OMP Worker Thread #						-		
CPU Utilization					CPU Util			
	Any Proce V Ar Thread V	Any Modu 🔻	Any U 🔻 🛛	User functi 🔻	Function •	Show in		
FILTER T 100.0%	Party Flood Party Flood							

Interactive Remote Data Collection

Performance analysis of remote systems just got a lot easier

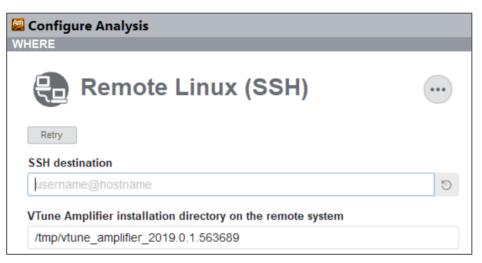
Interactive analysis

- 1) Configure SSH to a remote Linux* target
- 2) Choose and run analysis with the UI



Command line analysis

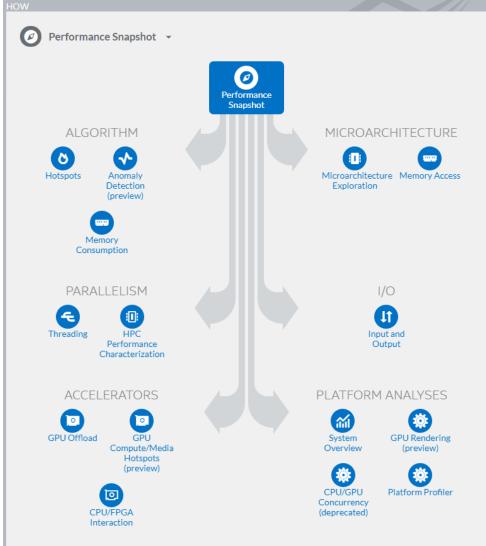
- Run command line remotely on Windows* or Linux* target
- 2) Copy results back to host and open in UI



Conveniently use your local UI to analyze remote systems

Analysis Types

- Performance Snapshot:
 - Used as a starting point to determine areas for deeper focus.
- Algorithm:
 - Hotspots: investigate call paths and find where your code spends the most time.
 - Anomaly Detection (preview): identify performance anomalies in frequently recurring intervals of code like loop iterations.
 - Memory Consumption: analyze memory consumption by app [Linux* only]
- Microarchitecture:
 - **Microarchitecture Exploration:** deep dive into the CPU pipeline stage and hardware units responsible for your hardware bottlenecks.
 - Memory Access: analyze CPU cache and main memory usage
- Parallelism:
 - Threading: visualize thread parallelism on available cores.
- I/O:
 - Input and Output: monitor utilization of the IO subsystems, CPU, and buses.
- Platform Analyses:
 - System Overview: monitors general behavior of the target system and identifies platform-level factors that limit performance
 - **Platform Profiler:** provides insights into overall system configuration, performance, and behavior.



More details: https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/set-up-project/analysis-types.html

Optimize Memory Access Memory Access Analysis - Intel® VTune™ Profiler

Tune data structures for performance

- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

Optimize NUMA latency & scalability

- True & false sharing optimization
- Auto detect max system bandwidth
- Easier tuning of inter-socket bandwidth

Easier install, Latest processors

- No special drivers required on Linux*
- Intel[®] Xeon Phi[™] processor MCDRAM (high bandwidth memory) analysis

Top Memory Objects by Latency

This section lists memory objects that introduced the highest latency to the overall application e								
Memory Object	Total Latency	Loads	Stores	LLC Miss Count ^②				
alloc_test.cpp:157 (30 MB)	65.6%	4,239,327,176	4,475,334,256	0				
alloc_test.cpp:135 (305 MB)	6.8%	411,212,336	441,613,248	0				
alloc_test.cpp:109 (305 MB)	6.3%	439,213,176	449,613,488	0				
alloc_test!I_data_init.436.0.6 (576 B)	5.2%	742,422,272	676,820,304	0				
[vmlinux]	4.6%	173,605,208	116,003,480	0				
[Others]	11.5%	1,533,646,008	1,674,450,232	0				

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

Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object /	Stores	LLC Miss Count 🔻			
Allocation Stack	Stores	Local DRAM Access Count	Remote DRAM Access Count		
doTriad\$omp\$parallel_for@2	40,307,609,1	2,439,273,176	2,430,472,912		
▶ triad!c (152 MB)	19,200,576	1,821,654,648	1,864,855,944		
triad!b (152 MB)	10,400,312	615,218,456	560,816,824		
[Unknown]	7,200,216	2,400,072	3,200,096		
triad!doTriad (2 MB)	15,200,456	0	0		
[Stack]	2,120,063,600	0	1,600,048		
triad!a (152 MB)	38,135,544,0	0	0		
update_blocked_averages	6,400,192	2,400,072	0		

Microarchitecture Exploration

Hierarchical view of the execution pipeline

- Pinpoint sections of the pipeline with performance problems flagged by VTune
- Hover over metrics for a detailed description

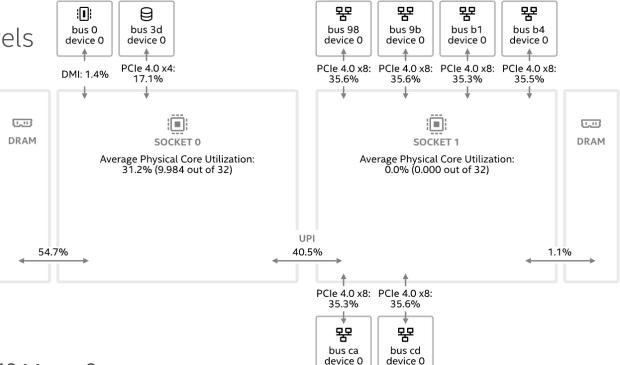
Visualize the pipeline at the function level in the bottom-up tab

Analysis Configuration	Collection L	.og Summa	ry Bottom-		Platform
Grouping: Function / Call	Stack			~ × P %	Microarchitecture Usage: 27.0% 🏲 of Pipeline Slots 🖄 Θ
			Back	-End Bound	
Function / Call Stack			Memory Bou	nd	
	L1 Bound 🖻	L2 Bound	L3 Bound	DRAM Bound	
grid_intersect	11.4%	0.0%	13.9%	6.3%	
sphere_intersect	14.6%	1.5%	2.9%	2.9%	
grid_bounds_intersect	100.0%	0.0%	20.2%	0.0%	
func@0x4b2be3a0	0.0%	0.0%	0.0%	0.0%	Memory Bound: 34.98%
pos2grid	0.0%	0.0%	0.0%	0.0%	This part of µPipe is fraction of Memory
tri_intersect	0.0%	0.0%	0.0%	0.0%	Bound. The metric value is high. This can indicate
func@0x14016b349	0.0%	0.0%	0.0%	0.0%	that the significant fraction of execution
Raypnt	0.0%	0.0%	0.0%	0.0%	pipeline slots could be stalled due to demand memory load and stores. Use Memory
func@0x10046130	0.0%		0.0%		Access analysis to have the metric
func@0x10076012	90.6%	0.0%	0.0%	0.0%	breakdown by memory hierarchy, memory bandwidth information, correlation by
libm_sse2_sqrt_precise	0.0%	94.7%	0.0%	0.0%	memory objects.
libm_sse2_pow_precise	100.0%	0.0%	0.0%	100.0%	Rí 27.0% of Pipe
> func@0x140168968	0.0%	0.0%	0.0%	0.0%	Front-End Bound: 5.0% of Pipe
TBB Scheduler Interna	0.0%	0.0%	0.0%	0.0%	Bad Speculation: 14.4% 🏲 of Pipe
shader	0.0%	0.0%	0.0%	0.0%	Branch Mispredict: 0.0% of Pipe
func@0x6b102230	0.0%	0.0%	0.0%	0.0%	Machine Clears: 14.4% No of Pipe
light_intersect	100.0%	0.0%	0.0%	0.0%	Back-End Bound: 53.6% P of Pipe
intersect objects	100.0%	0.0%	0.0%	0.0% 🗡	Memory Bound: 35.0% 🏲 of Pipe
< >	<			(1)	L1 Bound: 14.6% P of Cloc

Elapsed Time $^{\odot}$: 4.000s $^{\circ}$	-0
Clockticks: 9,475,2	200,000
Instructions Retired: 10,939,2	200,000
CPI Rata®.	<u></u> 966
MUX R This metric shows how often	113
Retirin: machine was stalled without missing the L1 data cache. The	5% of Pipeline Slots
Front-E L1 cache typically has the	3% of Pipeline Slots
⊗ Bad Spt shortest latency. However, in certain cases like loads blocker	.5% 🕅 of Pipeline Slots
Bra on older stores, a load might	.5% 🌂 of Pipeline Slots
Mac suffer a high latency even though it is being satisfied by th	0% of Pipeline Slots
Back-El L1.	.8% 🎙 of Pipeline Slots
⊘ Melme, y bound .	🗸
🕥 L1 Bound 🟠	13.7% 🎙 of Clockticks
L2 Bound ^②	0.0% of Clockticks
L3 Bound ² :	8.4% 🌂 of Clockticks
DRAM Bound [®] :	5.8% of Clockticks
Store Bound [®] :	0.0% of Clockticks
Ore Bound [®] :	14.4% 🌂 of Pipeline Slots
Total Thread Count:	14
Paused Time ^② :	Os

Intel® VTune™ Profiler Input and Output Analysis

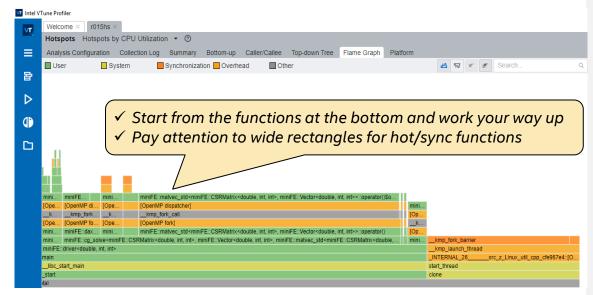
- Provides uncore- and device-centric view to locate performance bottlenecks in I/O-intensive apps at both HW and SW levels
- Two types of metrics to analyze:
 - Platform I/O: application- and deviceagnostic <u>hardware event-based metrics</u> for DRAM, PMEM, Intel UPI, PCIe, <u>Intel DDIO</u>, MMIO traffic consumption
 - API and OS I/O: <u>DPDK</u>, <u>SPDK</u>, <u>kernel I/O</u>
- Linux and FreeBSD targets are supported
- The full set of I/O metrics is available on Intel[®] Xeon[®] processors, including 3rd Generation Intel[®] Xeon[®] Scalable Processors



Easily Identify Hot Code Paths Flame Graphs* for Hotspots Analysis

Visualize Total Function CPU time spent

- Explore stacks and stack frames
 - Aggregation
 - Side-by-side visualization
 - Function bar as fraction of CPU Time
 - Different colors per function type
- Identify the time spent in each function and its callees
 Rich Experience with Intel[®] VTune[™] Profiler UI
- Select your visualization of choice
 - Flame or Icicle Graph
- Filter data by process, thread, time region, and more
- Jump to the function source code via stack pane

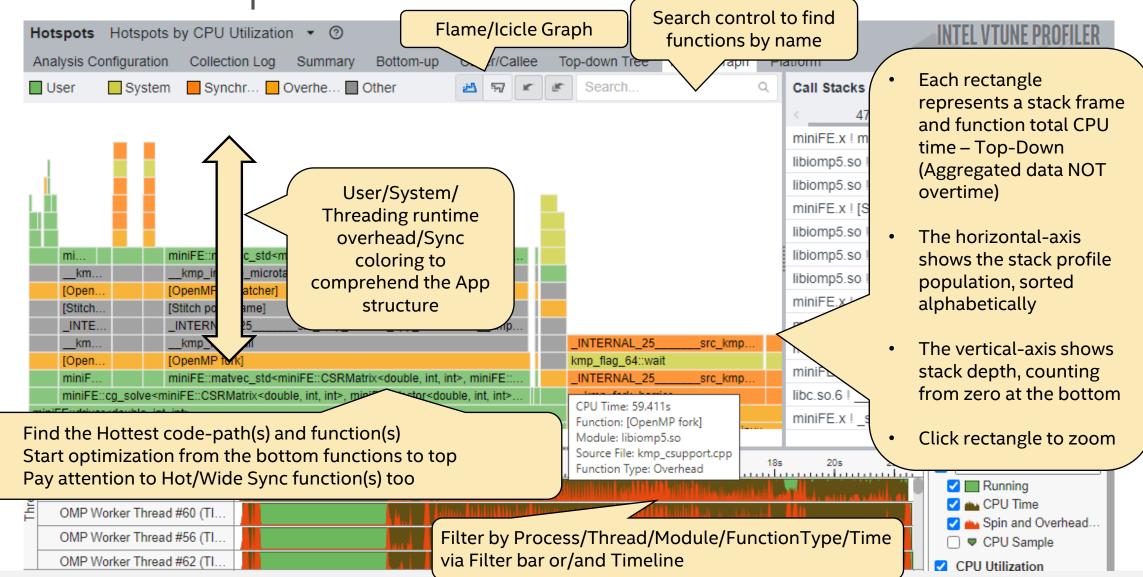


*Adapted based on Brendan Gregg's Flame Graphs

Intel® VTune™ Profiler Flame Graph

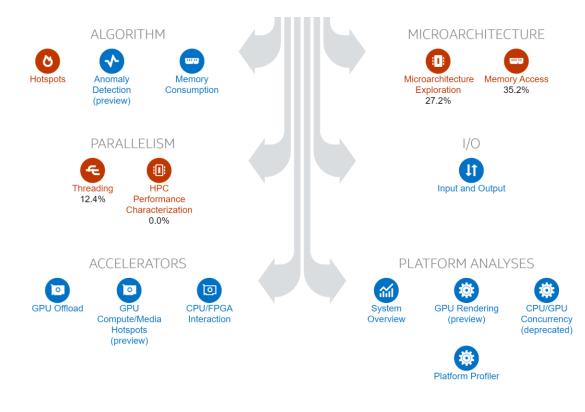
•

•

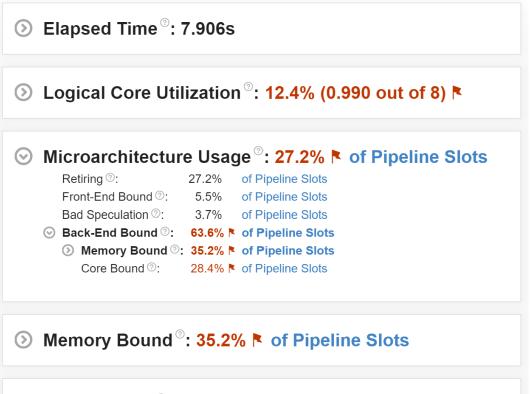


Intel® VTune™ Profiler Performance Snapshot Analysis

Choose your next analysis:



Characterize high-level aspects:

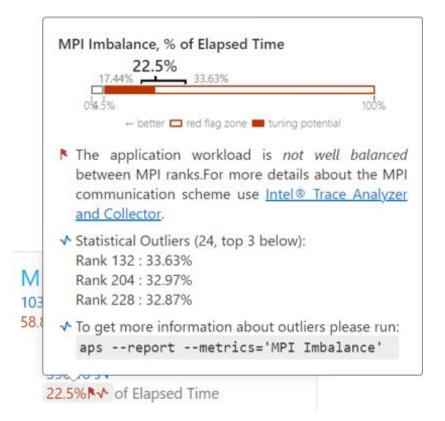


○ Vectorization[®]: 0.0% ► of Packed FP Operations

Roman Khatko

Intel® VTune[™] Profiler Application Performance Snapshot for MPI

- Outlier analysis for MPI applications at scale
 - Explore on source of imbalance
 - Choose nodes/ranks for <u>detailed</u> profiling with VTune



Intel[®] Advisor Overview

Intel[®] Advisor

Design code for modern hardware

Offload Modelling

 Efficiently offload your code to GPUs even before you have the hardware

Automated Roofline Analysis

- Optimize your GPU/CPU code for memory and compute
- Vectorization Optimization
- Enable more vector parallelism and improve its efficiency

Thread Prototyping

• Add effective threading to unthreaded applications

Flow Graph Analyzer

• Create, visualize and analyze task and dependency computation graphs

	ns > Logs		+	See Main Offload Mod	eling Viev
Top Metrics					~ ×
105.9x Speed Up for Accele	Image: Speed Up Image: Open control of the speed Up	58% Fraction of A	Image: Comparison of	2 Number of Offloads	?
Program Metrics					× >
Original Accelerated	44.86s 6.25s				
0	 Program Time on Host After . Non Accelerated Time Time in MPI calls Time on Target 	0.11s Os Nur 0.26s Spe Am	get Platform nber of Offloads eed Up for Accelera dahl's Law Speed ction of Accelerate	Dev 2 ated Code 105 Up 2.4:	
Q [⊕] ← → ×	▼ L3; GTI (Memory) - * Ci	uidance 🔻			
400			SP Vector F	-MA Peak: 437.83 (GFLOP
400 - GP OP S		8	SP Vector	Add Peak: 219.84 (GFLOP
100 70 47 44 GEL	OPS (5.9 x ³) + (5 -9)	8	SP Vector		GFLOP
100	OPS (5.9x) + a.	8	SP Vector	Add Peak: 219.84 (MA Peak: 108.62 (GFLOP
100 70 40 - 47.44 GFL	R CBISEC	c Intensity: 0.23 F e: 0.268 s	SP Vector DP Vector DP Vector	Add Peak: 219.84 (MA Peak: 108.62 (GFLOP

intel

"Automatic" Vectorization Often Not Enough

A good compiler can still benefit greatly from vectorization optimization

Compiler will not always vectorize

- Check for Loop Carried Dependencies using <u>Intel[®] Advisor</u>
- All clear? Force vectorization.
 C++ use: pragma simd, Fortran use: SIMD directive

Not all vectorization is efficient vectorization

- Stride of 1 is more cache efficient than stride of 2 and greater. Analyze with <u>Intel[®] Advisor</u>.
- Consider data layout changes
 <u>Intel[®] SIMD Data Layout Templates</u> can help

Arrays of structures are great for intuitively organizing data, but are much less efficient than structures of arrays. Use the <u>Intel® SIMD Data</u> <u>Layout Templates</u> (Intel® SDLT) to map data into a more efficient layout for vectorization.

Get Breakthrough Vectorization Performance

Intel® Advisor—Vectorization Advisor

Faster Vectorization Optimization

- Vectorize where it will pay off most
- Quickly ID what is blocking vectorization
- Tips for effective vectorization
- Safely force compiler vectorization
- Optimize memory stride

Data & Guidance You Need

- Compiler diagnostics + Performance Data + SIMD efficiency
- Detect problems & recommend fixes
- Loop-Carried Dependency Analysis
- Memory Access Patterns Analysis

E E Call Character I and			@ Performance	CPU Time 🗵		T	Miles Mar Marchael and	Vectorized Loops			
+ - Function Call Sites and Loops		Issues	Total Time	Self Time 🔻	Туре	Why No Vectorization?	Vector.	. Efficiency	Gain E.	. V	
Iloop in serial_mandelbrot at mandelbrot.cpp:70]	~		0.202s 35.39	0.202s 27.9%	Scalar	loop control variable was not identified. Explicitly compute the it					
Iloop in main\$omp\$parallel@164 at mandelbrot.cpp:181]	-		0.152s	0.152s	Scalar	loop control variable was not identified. Explicitly compute the it					
[loop in main\$omp\$parallel@219 at mandelbrot.cpp:237]	-		0.108s 🔲	0.108s	Inside vectorized						
Iloop in simd_mandelbrot at mandelbrot.cpp:126]	-		0.088s 🥅	0.088s	Inside vectorized						
Iloop in simd_mandelbrot at mandelbrot.cpp:114]	-	2 Possible ineffi	0.100s	0.012sl	Vectorized (Body)		AVX2	67%	2.69x	4	
Iloop in main\$omp\$parallel@164 at mandelbrot.cpp:169]	-	@ 1 Data type conv	0.162s 28.3%	0.010s	Scalar	outer loop was not auto-vectorized: consider using SIMD directive		1000			
Icop in serial_mandelbrot at mandelbrot.cpp:58]	-	@ 1 Data type conv	0.202s 35.39	0.000s1	Scalar	outer loop was not auto-vectorized: consider using SIMD directive					
Icoop in serial_mandelbrot at mandelbrot.cpp:57]	~	@ 1 Data type conv	0.202s 35.3%	0.000s1	Scalar	a outer loop was not auto-vectorized: consider using SIMD directive					
Solution Simd_mandelbrot at mandelbrot.cpp:112]	-	@ 1 Data type conv	0.100s	0.000s1	Scalar	inner loop was already vectorized					
Iloop in main\$omp\$parallel@164 at mandelbrot.cpp:164]	-	2 Assumed depe	0 1625 28 394	0.000s1	Threaded (OpenMP)	vector dependence prevents vectorization					

Optimize for Intel® AVX-512 with or without access to AVX-512 hardware

Intel.com/advisor

intel

Design your code for efficient offload Intel® Advisor - Offload Modeling

- Will your code benefit from GPU porting?
- How much performance acceleration will your code get from moving to the next-generation GPU?

Top Metrics						
3.3x Speed Up for Accelera	1 3.3X	⑦ Up	100% ⑦ 2 Fraction of Accelerated Code	⑦ f Offloads		
Program Metrics				^	Offload Bounded By	
Original	23.30s			_	Compute	w
Accelerated	7.00s				L3 Cache B	W 1
					Memory BW	
	Program Time on Host	0s	Target Platform	Gen9 GT2	Latencies	
	Non Accelerated Time	0s	Number of Offloads	2	Data Transfe	er
	Time in MPI calls	0s	Speed Up for Accelerated Code	3.3x	Launch Tax	
	Time on Target 7	7.00s	Amdahi's Law Speed Up	3.3x	Dependency	(
			Fraction of Accelerated Code	100%	Trip Count	
					Atomics	
					Unknown Non Offload	led
op Offloaded			^	Top Non-Offloaded	1	
Loop/Function ③	Speed-Up ③		Estimated Bound ⑦ Estimat ⑦		\wedge	
[loop in multiply1 at mu		23.2 6.99				
		28.5n			No data availat	ale

With Offload Modeling, you can:

- Pinpoint offload opportunities where it pays off the most.
- Project the performance on GPU.
- Identify bottlenecks and potential performance gains.
- Get guidance for optimizing a data transfer between host and target devices.

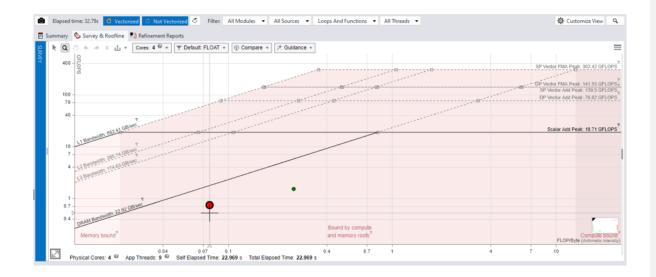
Results can be reviewed in HTML reports.

Find Effective Optimization Strategies

Intel® Advisor—Automated Roofline Analysis

Optimize for memory and compute

- Highlights poor performing loops
- Shows performance 'headroom' for each loop
 - Which can be improved
 - Which are worth improving
- Shows likely causes of bottlenecks
- Suggests next optimization steps



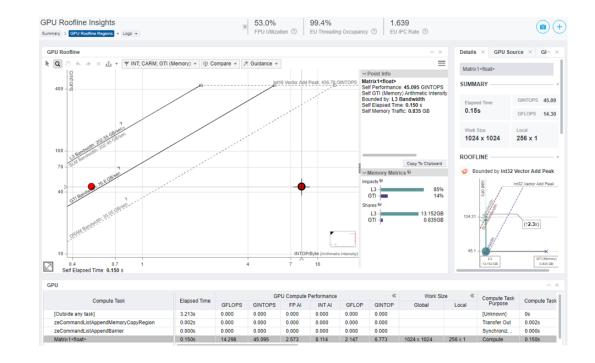
"I am enthusiastic about the new "integrated roofline" in Intel® Advisor. It is now possible to proceed with a step-by-step approach with the difficult question of memory transfers optimization & vectorization which is of major importance."

Nicolas Alferez, Software Architect Onera – The French Aerospace Lab

Find Effective Optimization Strategies Intel® Advisor - Roofline Analysis on GPU

GPU Roofline Performance Insights

- Highlights poor performing loops
- Shows performance 'headroom' for each loop
 - Which can be improved
 - Which are worth improving
- Shows likely causes of bottlenecks
 - Memory bound vs. compute bound
- Suggests next optimization steps



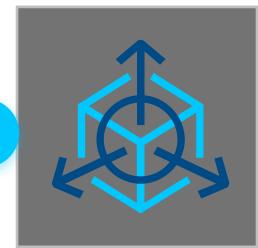
CPU Performance Profiling Exercises

MandelbrotOMP Sample

Workflow



Log into an Intel[®] DevCloud GPU node and configure the MandelbrotOMP sample Run Intel Advisor: Offload Advisor to estimate performance on Gen9 GT2 GPU Run Intel Advisor: **GPU Roofline** on offloaded implementation to visualize GPU performance



Run Intel VTune Profiler: **GPU Hotspots** for deeper insights into GPU kernels and device metrics

DevCloud Setup



Log into DevCloud via ssh



Start interactive gpu node:

\$ qsub -I -l nodes=1:gpu:ppn=2



Create MandelbrotOMP sample:

https://github.com/oneapisrc/oneAPI-samples



Start Intel VTune Profiler Server in second ssh terminal

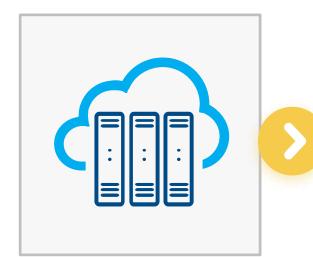
DevCloud Setup



Intel DevCloud provides a free environment for testing the latest Intel CPUs and GPUs. Intel oneAPI toolkits are already installed and set up for use.

To create a DevCloud account, follow these steps: https://www.intel.com/content/www/us /en/forms/idz/devcloudenrollment/oneapi-request.html

Workflow – CPU Profiling



Configure the MandelbrotOMP sample Run Intel VTune Profiler: Hotspots and HPC Performance Characterization to get a high level view of performance Run Intel Advisor: **CPU Roofline** to get a deeper understanding of vectorization performance

Run Intel Advisor:

Offload Advisor

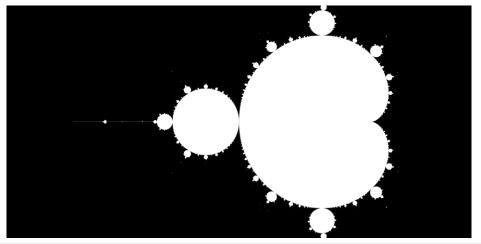
performance on

Gen9 GT2 GPU

to estimate

MandelbrotOMP

- This sample runs one or all of four algorithms for generating a Mandelbrot image. Each algorithm has an increasing level of optimization, from a serial implementation to using OpenMP for parallelization and simd vectorization.
- From oneapi-cli: oneAPI Direct Programming -> C++ -> Combinational Logic -> Mandelbrot OMP
- Github link: https://github.com/oneapi-src/oneAPIsamples/tree/master/DirectProgramming/C%2B%2B/CombinationalLogic/MandelbrotOMP



MandelbrotOMP Makefile

Change options to use LLVM-based Intel C++ compiler

- Change compiler from icpc to icpx
- Remove -ipo from CFLAGS and LIBFLAGS and add: -g D_INTEL_COMPILER
- Comment out ifdef defining PERFNUM

```
CXX := icpx
SRCDIR := src
BUILDDIR := release
CFLAGS := -03 - \frac{ipo}{-qopenmp} - std = c + +11
-g -D INTEL COMPILER
EXTRA CFLAGS :=
LIBFLAGS := -qopenmp
           -CFLAGS +=
                        <u>D DFDF</u>
TARGET := $ (BUILDDIR) / Mandelbrot
icpx: $(TARGET)
```

MandelbrotOMP main.cpp

Increase workload size

- src/main.cpp
 - Change the max_depth from 100 to 5000

//Modifiable parameters: int height = 1024; int width = 2048 //Width should be a multiple of 8 int max_depth = 5000;

All commands

- \$ cd MandelbrotOMP
- \$ vim Makefile
 - Make the changes from slide 25
- \$ vim src/main.cpp
 - Make the change from slide 25
- \$ make
- \$ make run

Intel[®] VTune[™] Profiler Exercise

MandelbrotOMP Sample

Generate VTune Command Line

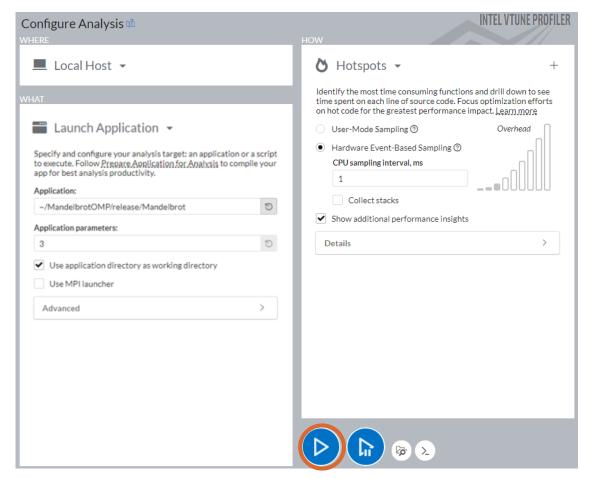
- 1. Open Intel VTune Profiler on local host
- 2. In "Where" select "Arbitrary Host"
 - This is just for generating command lines
- 3. In "What":
 - Application: ~/MandelbrotOMP/release/Mandelbrot
 - Application parameters: 3
- 4. In "How" select "Hotspots"
 - Hardware event-based Sampling
 - Show additional performance insights enabled
- 5. Press Command Line button

Configure Analysis 🛍	INTEL VTUNE PROFILER
 Arbitrary Host (not connected) This target system type is used to produce a command line analysis configuration for the selected microarchitecture. You cannot start this analysis from the host. To collect data on the remote system with no connection to the host, copy the generated command line and run it directly on the remote system. Hardware platform Intel(R) Processor code named Skylake Operating system GNU/Linux WHAT Example Application Specify and configure your analysis target: an application or a script to execute. Follow Prepare Application for Analysis to compile your app for best analysis productivity. Application: 	 Hotspots - + Identify the most time consuming functions and drill down to see time spent on each line of source code. Focus optimization efforts on to code for the greatest performance impact. Learn more: User-Mode Sampling O Hardware Event-Based Sampling O CPU sampling interval, ms Collect stacks Show additional performance insights Details
~/MandelbrotOMP/release/Mandelbrot 🔊	
Application parameters:	
3 ව	
 Use application directory as working directory 	
Use MPI launcher	
Advanced	

\$ vtune -collect hotspots -knob sampling-mode=hw --app-working-dir=~/MandelbrotOMP/release --~/MandelbrotOMP/release/Mandelbrot 3

Run from GUI

- 1. Run vtune-gui
- 2. In "Where" select "Local Host"
- 3. In "What":
 - Application: ~/MandelbrotOMP/release/Mandelbrot
 - Application parameters: 3
- 4. In "How" select "Hotspots"
 - Hardware event-based Sampling
 - Show additional performance insights enabled
- 5. Press Start button



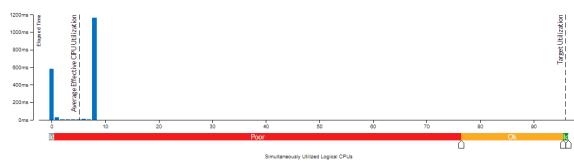
\$ vtune -collect hotspots -knob sampling-mode=hw --app-working-dir=~/MandelbrotOMP/release --~/MandelbrotOMP/release/Mandelbrot 3

intel. 42

View Results – Summary Tab

- Top hotspot is the parallel_mandelbrot function
- Effective CPU Utilization Histogram shows majority of time running on 8 logical CPUs
 - There are only 8 threads, so this is expected
- Insights highlights problems with the following:
 - Parallelism This number is low due to the number of threads defined in the application code. Increasing the number of threads should scale.
 - Microarchitecture Usage This number is low, indicating poor use of hardware resources.
 - Vectorization This number is very low, meaning that there are floating point operations in the code but none are vectorized. This is a good place to start.
- Next Step: Run the recommended HPC Performance Characterization Analysis for more vectorization insights

\odot	Elapsed Time [®] : 1.787s © CPU Time [®] : 9.640s Total Thread Count: 8 Paused Time [®] : 0s Top Hotspots This section lists the most active functions I functions typically results in improving over			; these hotspot	up view for the Flame G Explore Addit Parallelism Use C your ap Microarchit	nificant hotspots in t n-depth analysis per i raph view to track cri ional Insights © : 5.5% M Threading to explore i plication. ecture Usage © : 32	he Top Hotspots list, switch to the Bottom function. Otherwise, use the Caller/Callee tical paths for these hotspots. more opportunities to increase parallelism .8% operation to explore how efficiently your
	Function	Module		% of CPU Time ③	applica	tion runs on the used	
	Z19parallel_mandelbrotddddiii.extracted	Mandelbrot	9.350s	97.0%		n ③: 0.0% HPC Performance Ch	aracterization to learn more on vectorizat
	kmp_fork_barrier	libiomp5.so	0.240s	2.5%			. This code has floating point operations a
	stbi_write_png_to_mem	Mandelbrot	0.010s	0.1%			ither <u>recompiling the code with optimizat</u> or using <u>Intel Advisor</u> to vectorize the loop
	kmp_api_omp_set_num_threads	libiomp5.so	0.010s	0.1%	options	allow vectorization d	if using <u>inter Advisor</u> to vectorize the loop
	stbiw_zlib_countm	Mandelbrot	0.010s	0.1%			
		N/A*	0.020s	0.2%			



View Results – Bottom-up Tab

- The thread timeline view at the bottom shows that there are 8 OMP threads, and all of them are primarily colored in brown
 - Green indicates that the thread is running, but brown is where the thread was actively using the CPU
 - Red indicates spin and overhead time, and in this case the threads are waiting for all of them to finish.
- Overall, thread performance has good concurrency with no real problems. The only issue is that the application limits itself to 8 threads instead of taking advantage of more available CPUs.
- Next Step: Run the recommended HPC Performance Characterization Analysis for more vectorization insights

Analysis Configuration Collection Log	Summary Botto	om-up Caller	/Callee Top-down Tree Flan	ne Graph Platform
Grouping: Function / Call Stack			~	S Call Stacks
Function / Call Stack	CPU Time 🔻 🖉	Module	Function (Fu Mandelbrot ! _Z19parallel_man
_Z19parallel_mandelbrotddddiii.extracted	9.350s	Mandelbrot	_Z19parallel_mandelbrotddddii	i.e libiomp5.so ! [OpenMP dispatc
kmp_fork_barrier	0.240s	libiomp5.so	kmp_fork_barrier(int, int)	libiomp5.so !kmp_fork_call+0
stbi_write_png_to_mem	0.010s	Mandelbrot	stbi_write_png_to_mem	libiomp5.so ! [OpenMP fork]+0
kmp_api_omp_set_num_threads		libiomp5.so	kmp_api_omp_set_num_thread	ds Mandelbrot parallel mandelbr
stbiw_zlib_countm		Mandelbrot	stbiwzlib_countm(unsigned ch	Mandelbrot ! main+0x54d - ma
stbi_zlib_compress		Mandelbrot	stbi_zlib_compress	libc.so.6 ! _libc_start_main+0xf2
kmp_join_call	0.010s	libiomp5.so	kmp_join_call	Mandelbrot ! _start+0x2d
	25 0.45 0	ðs 0.85	1s 1.2s 1.4s 1.6	🖬 🚺 🚺 🚺
	2s 0.4s 0	.8s 0.8s	1s 1.2s 1.4s 1.8	
~. •	2s 0.4s 0	.0s 0.8s	1s 1.2s 1.4s 1.8	Running
	2s 0.4s 0	.0s 0.8s	1s 1.2s 1.4s 1.8	Running
OMP Primary Thread #0 (TID OMP Worker Thread #6 (TID	25 0.45 0	.0s 0.8s	1s 1.2s 1.4s 1.8	✓ Imead ✓ Image: Spin and Overhead Time
OMP Primary Thread #0 (TID OMP Worker Thread #6 (TID OMP Worker Thread #6 (TID	25 0.45 0	0s 0.8s	1s 1.2s 1.4s 1.8	Image: Second secon
OMP Primary Thread #0 (TID OMP Worker Thread #6 (TID OMP Worker Thread #6 (TID OMP Worker Thread #2 (TID	25 0.45 0	0s 0.8s	1s 1.2s 1.4s 1.8	Image: CPU Time Image: CPU Utilization Image: CPU Time
OMP Primary Thread #0 (TID OMP Worker Thread #6 (TID OMP Worker Thread #6 (TID OMP Worker Thread #2 (TID OMP Worker Thread #2 (TID	25 0.45 0	85 0.85 1	1s 1.2s 1.4s 1.5	Image: CPU Time Image: CPU Utilization Image: CPU Time
OMP Primary Thread #0 (TID OMP Worker Thread #6 (TID OMP Worker Thread #6 (TID OMP Worker Thread #2 (TID OMP Worker Thread #2 (TID OMP Worker Thread #4 (TID OMP Worker Thread #1 (TID	25 0.45 0	.85 0.85	1s 1.2s 1.4s 1.6;	Image: CPU Time Image: CPU Utilization Image: CPU Time
OMP Primary Thread #0 (TID OMP Worker Thread #6 (TID OMP Worker Thread #6 (TID OMP Worker Thread #2 (TID OMP Worker Thread #4 (TID OMP Worker Thread #1 (TID OMP Worker Thread #1 (TID	25 0.45 0	.85 0.85	15 1.25 1.45 1.8 	Image: Second secon

Collect HPC Performance Characterization

Generate VTune Command Line

- In "How" select "HPC Performance Characterization"
 - Use default options
- 2. Press Command Line button

Configure Analysis	INTEL VTUNE PROFILER
 WHERE Arbitrary Host (not connected) This target system type is used to produce a command line analysis configuration for the selected microarchitecture. You cannot start this analysis from the host. To collect data on the remote system with no connection to the host, copy the generated command line and run it directly on the remote system. Hardware platform Intel(R) Processor code named Skylake Operating system GNU/Linux March 2000 Content of the content o	HOW III: HPC Performance Characterization → + Analyze performance aspects of compute-intensive applications, including CPU and GPU utilization. Get information on OpenMP efficiency, memory access, and vectorization. <u>Learn more</u> CPU sampling interval, ms CPU sampling interval, ms Analyze memory bandwidth
WHAT Launch Application Specify and configure your analysis target: an application or a script to execute. Follow <u>Prepare Application for Analysis</u> to compile your app for best analysis productivity. Application:	 Evaluate max DRAM bandwidth Analyze OpenMP regions Collect affinity data Details >
 ~/MandelbrotOMP/release/Mandelbrot Application parameters: 3 ✓ Use application directory as working directory 	
Use MPI launcher Advanced	

\$ vtune -collect hpc-performance --app-working-dir=~/MandelbrotOMP/release --

~/MandelbrotOMP/release/Mandelbrot 3

Collect HPC Performance Characterization

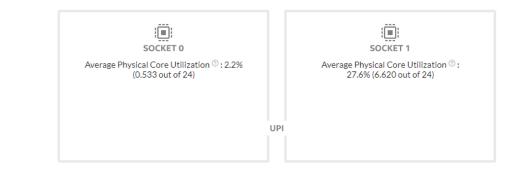
View Results – Summary Tab

- Platform Diagram shows majority of CPU utilization is on Socket 1, with no UPI or memory traffic
- Effective Physical Core Utilization is low due to the number of threads defined in the application
- There are no issues due to memory accesses
- Vectorization shows that 47% of uOps are floating point, but none are vectorized (packed). Adding vectorization could significantly improve performance.
- Next Step: Run CPU Roofline with Intel Advisor

HPC Performance Characterization ③ 🛱 Analysis Configuration Collection Log Summary Bottom-up

Elapsed Time[®]: 1.336s

O Platform Diagram



- ② Effective Physical Core Utilization[®]: 14.9% (7.158 out of 48) ▲
- Memory Bound[®]: 0.0% of Pipeline Slots
- Sectorization[®]: 0.0% ▶ of Packed FP Operations

\odot	Instruction Mix:									
	SP FLOPs [®] :	0.0%	of uOps							
	OP FLOPs [®] :	47.0%	of uOps							
	Packed [®] :	0.0%	from DP F	P						
	Scalar ⁽²⁾ :	100.0% 🎙	from DP F	P						
	x87 FLOPs [®] :	0.0%	of uOps							
	Non-FP [®] :	53.0%	of uOps							
	FP Arith/Mem Rd Instr. Ratio 💿:	128.568								
	FP Arith/Mem Wr Instr. Ratio [®] :	2,699.923								
\odot	Top Loops/Functions with FPU U	Jsage by CF	PUTime							
	This section provides informatio	n for the mo	ost time cor	nsun	ning loops/func	tions with f	oatin	g point operation	ns.	
	Function		CPU Time	(?)	% of FP ⑦ Ops	FP Ops: Packed	0	FP Ops: ③ Scalar	Vector Instruction ③ Set	Loop ⑦ Type
	[Loop at line 180 in _Z19parallel_ tddddiii.extracted]	mandelbro	9.31	2s	48.0%	().0%	100.0% 🏼		

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

Intel[®] Advisor Exercise

MandelbrotOMP Sample

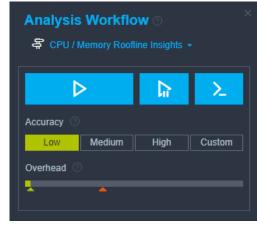
Run from GUI

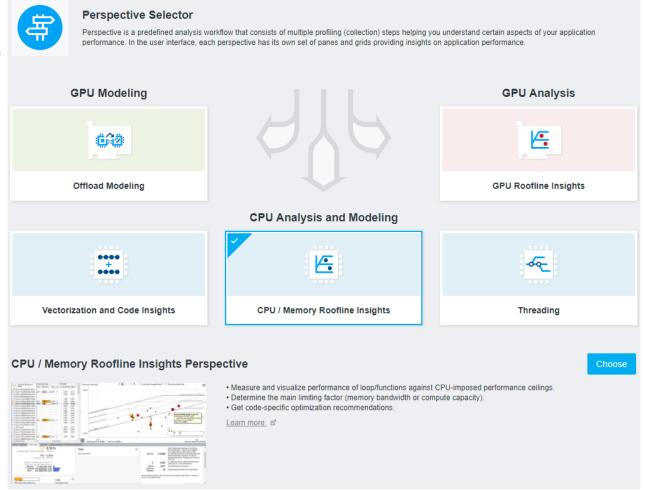
- 1. Run advixe-gui
- 2. Create a new project
 - Application: ~/MandelbrotOMP/release/Mandelbrot
 - Application parameters: 3
- 3. Press OK

Analysis Target Binary/Symbol Search	Source Search			
Survey Analysis Types Survey Hotspots Analysis Trip Counts and FLOP Ana Suitability Analysis Refinement Analysis Types		e application executable (target) to analyze. P application file "~/MandelbrotOMP/release/N		
Remory Access Patterns A Dependencies Analysis	Application:	~/MandelbrotOMP/release/Mandelbrot	~	Browse
Performance Modeling	Application parameters:	3	~	Modify
	Use application direct	ory as working directory		
	Working directory:		~	Browse
	User-defined environmer	nt variables:		Modify
	Modules:	 Include only the following module(Exclude the following module(s) 	(5)	
	Modules:		(s)	Modify
< >>	Modules: Managed code profiling Child application:	Exclude the following module(s)	(s)	Modify

Run from GUI - cont

- 1. From the Perspective Selector, select CPU/Memory Roofline Insights
- 2. Press Choose button
- 3. From the new Analysis Workflow panel, press the Run button





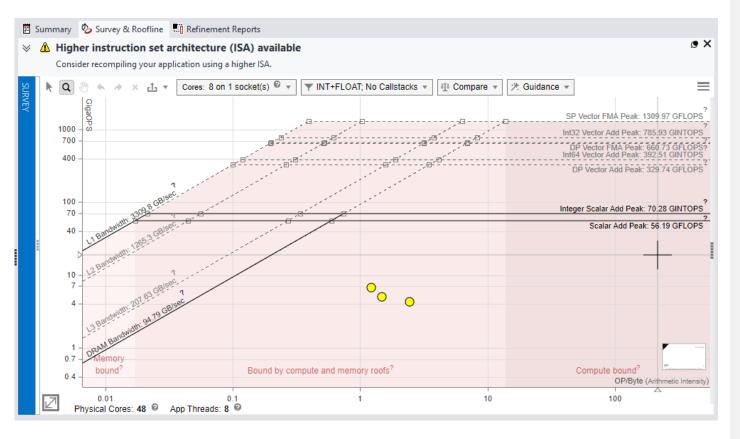
Run Advisor in the remote CLI

- 1. Run the roofline collection:
 - \$ advisor --collect=roofline --project-dir=./cpu_roofline
 -- ./release/Mandelbrot 3
- 2. Package results for viewing on the local host:
 - \$ advisor --snapshot --project-dir=./cpu_roofline --pack -cache-sources --cache-binaries -- ./cpu_roofline_snapshot

View Results – Roofline

- There are a few loops shown for INT data, but FLOAT is the one doing the compute.
- The FLOAT loop points to line 168, which is the inner for loop. It is compute bound.

Note that there is no large loop dot for the main while loop. This is likely due to the arithmetic intensity of the while loop getting applied to the earlier for loop. There are also issues with the compiler being unable to determine the number of iterations in the while loop due to the break statement.



View Results – Survey

- The top loop uses the while command, and the compiler won't vectorize it due to unknown iterations.
- The loop identified in the roofline view at line 168 is an inner for loop, which wasn't vectorized because of the added complexity of the while loop. The compiler determined it would hurt performance.

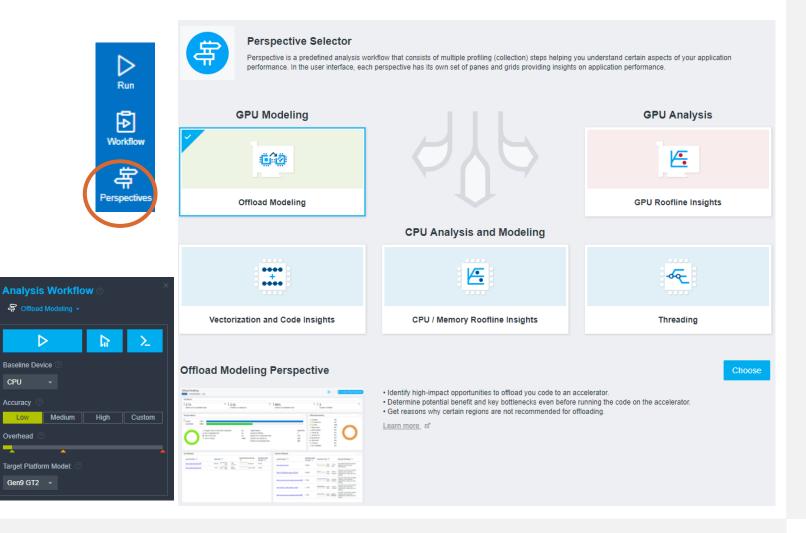
Note that Advisor currently lacks support for the vectorization report from icpx – the new LLVMbased compiler - which is why some vectorization details are missing. This is being worked on.

	Consider recompiling your application using a higher ISA.				-	1		т
ROOFLINE	🛨 🖃 Function Call Sites and Loops	Performance Issues	CPU Time Total Time		Туре	Why No Vec	torization?	1
Ë.	Standard Stand Standard Standard St Standard Standard Stand Standard Standard Stand	© 1 Potential	9.340s	9.340s	Scalar			
			0.010s l	0.010s l	Inlined Function			
	□ ^C [loop in stbiw_zlib_countm at stb_image_write.h:825]		0.010s l	0.010s l	Scalar			
	Solution State		0.020s l	0.010s l	Scalar			
	∑ f_start		9.400s 97	.1% 0.000s1	Function			
	_ □ f main		9.400s 97	.1% 0.000s1	Function			
	Image: State of the state o	-	9.370s 96	.8% 0.000s1	Function			
	Image: State S		9.340s 96	.5% 0.000s1	Function			TI.
	□ 0 [loop in Z19parallel mandelbrotddddiii.extracted at mandelbrot.cpp:168]	@ 1 Data type c	9.340s 96	=	Scalar			t
	□ 0 [loop in _Z19parallel_mandelbrotddddiii.extracted at mandelbrot.cpp:163]	@ 1 Data type c		=	Scalar			t
	☑ ^C [loop in _Z19parallel_mandelbrotddddiii.extracted at mandelbrot.cpp:163]		9.340s 96	_	Scalar			t
	Istbi_write_png		0.030s l	0.000s I	Inlined Function			t
	∑ f write image		0.000.1					
			0.030s1	0.000s	Inlined Function			
	(小作 Iloon in thi write nng to mem at th image write h:1002) <	<	0.03051	0.000s1	Inlined Function		2	
File	Top Down Code Analytics Assembly P Recommendations Why N : cache_0ef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_r	o Vectorization?	0.010-1	0.000-1	Cralar			
File Lir	Top Down Code Analytics Assembly P Recommendations Why N cache_0ef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_ine	lo Vectorization? nandelbrotddddiii	0.010-1	0.000+1		Time %) Traits	
File Lir 174	Top Down Code Analytics Assembly P Recommendations Why N : cache_0ef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_n me Source // depth should be an int, but the vectorizer will not	o Vectorization? mandelbrotddddiii vectorize,	• 0 010e I	0.000-1	Cralar	Time %		
File Lir 174 175	Top Down Code Analytics Assembly P Recommendations Why N cache_0ef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_ne Source // depth should be an int, but the vectorizer will not // complaining about mixed data types switching it to complain the source switch	lo Vectorization? mandelbrotddddiii vectorize, double is worth	• 0 010e I	0.000-1	Cralar	Time %		
File Lin 174 175 176	Top Down Code Analytics Assembly Recommendations Why N cache_0ef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_ ne Source // depth should be an int, but the vectorizer will not // complaining about mixed data types switching it to a // small cost in performance to let the vectorizer work	lo Vectorization? mandelbrotddddiii vectorize, double is worth	extracted	0.000-1	Cralar	Time %		
File Lin 174 175 176 177	<pre>Comparison Code Analytics Assembly Recommendations Why N cache_Oef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_t ne</pre>	<pre>vectorization? mandelbrotddddiii vectorize, double is wort} k</pre>	extracted	0.000-1	Cralar	Time %		
File Lin 174 175 176 177 178	<pre>Compose Code Analytics Assembly P Recommendations I Why N cache_Oef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_t ne</pre>	<pre>vectorization? mandelbrotddddiii vectorize, double is wort} k</pre>	extracted	0.000-1	Cralar	Time %		
File Lin 174 175 176 177 178 179	<pre>Top Down Code Analytics Assembly P Recommendations Why N cache_Oef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_ ne Source // depth should be an int, but the vectorizer will not // complaining about mixed data types switching it to o // small cost in performance to let the vectorizer word double depth = 0; // Figures out how many recurrences are required beform // max_depth</pre>	<pre>vectorization? mandelbrotddddiii vectorize, double is wort} k</pre>	extracted	Total Time %	Cralar Loop/Function			
File	<pre>cache_Oef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel, ne Source // depth should be an int, but the vectorizer will not // complaining about mixed data types switching it to o // small cost in performance to let the vectorizer worl double depth = 0; // Figures out how many recurrences are required before // max_depth while (depth < max_depth) {</pre>	<pre>vectorization? mandelbrotddddiii vectorize, double is wort! k e divergence, u</pre>	extracted h the up to	0.000-1	Cralar Loop/Function	Time %		
File Lin 174 175 176 177 178 179	<pre>cache_Oef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel, ne</pre>	<pre>vectorization? mandelbrotddddiii vectorize, double is wort! k e divergence, u</pre>	extracted h the up to	Total Time %	Cralar Loop/Function			
File Lin 174 175 176 177 178 179	<pre>Compose Code Analytics Assembly Recommendations Why N cache_Oef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel_i ne</pre>	<pre>vectorization? mandelbrotddddiii vectorize, double is wort! k e divergence, u</pre>	extracted h the up to	Total Time %	Cralar Loop/Function			
File Lin 174 175 176 177 178 179	<pre>cache_Oef0382e6feb3d36ecfcb7421231f7d5_mandelbrot.cpp:180_Z19parallel, ne</pre>	<pre>vectorization? mandelbrotddddiii vectorize, double is wort1 k e divergence, u ddelbrot.cpp:18</pre>	extracted h the up to	Total Time %	Cralar Loop/Function			

Collect Offload Advisor

Run from GUI - cont

- 1. Go back to the Perspective Selector and select Offload Modeling
- 2. Press Choose button
- 3. From the new Analysis Workflow panel:
 - 1. Select Low for Accuracy
 - 2. Select Gen9 GT2 from the Target Platform Model drop-down
 - 3. Press the Run button



Collect Offload Advisor

Run Advisor in the remote CLI

1. Run the offload collection:

\$ advisor --collect=offload --accuracy=low --targetdevice=gen9_gt2 --project-dir=./offload_advisor ./release/Mandelbrot 3

2. Package results for viewing on the local host:

\$ advisor --snapshot --project-dir=./offload_advisor -pack --cache-sources --cache-binaries --./offload_advisor_snapshot

Collect Offload Advisor

View Results – Summary

- Top Metrics shows that the speed-up for accelerated code and Amdahl's Law are very close, indicating that the offloaded code makes up most of the workload. If accelerated code speed up is high but the Amdahl's law speed up is close to 1.000x, then offloading likely isn't worth it.
- **Program Metrics** contains more details about the accelerated code and how much program time will remain on the host.
- Offload Bounded By shows the items that may impact performance of the code once it is offloaded. In this case the offloaded code will be compute bound.
- Modeling Parameters are the hardware characteristics of the target device. Advisor provides configurations for many Intel GPUs.
- Top Offloaded / Non-Offloaded these are loops or functions that have the potential to be offloaded. If the speed-up is significant enough, Advisor will recommend offloading. Some loops or functions will incur too much overhead to make offloading profitable. In this case, the main OMP compute loop is recommended for offloading.

Note that this report is from the non-vectorized implementation of the Mandelbrot function. With vectorization and parallelism, Offload Advisor will not recommend offloading as the overhead will be more than the performance gain.

Optimizing for CPU is still a good idea!

2.459x 2.446x 96% 1 Speed-up for Accelerated Code Amdahi's Law Speed Up Fraction of Accelerated Code 1 Program Metrics Image: Comparison of Accelerated Code Image: Comparison of Accele	neters o Remodel
Original 1.28s Accelerated 585.8ms Program Time on Host 110.0ms Time on Target 475.8ms Time in MPI calls 0s Non-Accelerable Time 0s Data Transfer Tax 0s Kernel Launch Tax 5.6us	
Accelerated 585.8ms Program Time on Host 110.0ms Time on Target 475.8ms Time in MPI calls 0s Non-Accelerable Time 0s Data Transfer Tax 0s Kernel Launch Tax 5.6us GTI BW O%	o Remodel
Accelerated 585.8ms Program Time on Host 110.0ms Time on Target 475.8ms Time in MPI calls 0s Non-Accelerable Time 0s Data Transfer Tax 0s Kernel Launch Tax 5.6us GTI BW O%	
Program Time on Host 110.0ms Time on Target 475.8ms Time in MPI calls 0s Non-Accelerable Time 0s Data Transfer Tax 0s Kernel Launch Tax 5.6us Gui P GT2 Reset 5et to H L3 Cache BW 0% GT RW 0%	
 Program Time on Host 110.0ms Time on Target 475.8ms Time in MPI calls 0s Non-Accelerable Time 0s Data Transfer Tax 0s Kernel Launch Tax 5.6us GTL RW 	
 Time in MPI calls Non-Accelerable Time Data Transfer Tax Kernel Launch Tax 5.6us GTL RW GTL RW 	
Non-Accelerable Time Os Data Transfer Tax Os Kernel Launch Tax 5.6us GTL RW 0%	rdware Default
Data Transfer Tax 0s Compute 96% L3 Cache BW 0% GTI BW 0%	arameters 💿
Kernel Launch Tax 5.6us GTI BW 0%	2
GTLBW 0%	24
	1.45.011
LLC BW 0%	1.15 GHz
Speed-up for Accelerated Code 2.459x Memory BW 0%	73.6 GB/s
Latencies 0%	
Fraction of Accelerated Code 96% Data Transfer 0%	220.8 GB/s
Launch Tax 0%	
	512 kE
Admics 0% LLC Bandwidth (73.6 GB/s
Non Offloaded 4%	
LLC Size (2)	8 ME
Top Offloaded ^ I Top Non-Offlo	ded

MandelbrotOMP with GPU Offload

To help demonstrate the capabilities of Intel Offload Advisor, we added a fifth function to use OpenMP offload to a GPU target:

- src/mandelbrot.cpp
 - Copy the omp_mandelbrot (..) function and rename to offload_mandelbrot (..)
 - Change #pragma omp parallel for schedule to:
 - #pragma omp target teams distribute \

```
parallel for simd collapse(2) \
map(from:output[0:width*height])
map(to:height,width,xstep,ystep,max_depth)
```

- src/mandelbrot.hpp
 - Copy the omp_mandelbrot (...) function and rename to offload_Mandelbrot (...)

MandelbrotOMP with GPU Offload

- Add a fifth option to enable the new offload_mandelbrot function
- src/main.cpp
 - Change the max_depth from 100 to 5000
 - Add variable offload_time to
 - double serial_time, omp_simd_time, omp_paraIlel_time, omp_both_time;
 - Add section for offload_mandelbrot under printf("\nRunning_all tests\n")
 - Add case 5 with offload_Mandelbrot to switch (option)
 - Not using PERF_NUM

MandelbrotOMP Makefile

- Change options to use OpenMP offload capability
 - Change compiler from icpc to icpx
 - Remove qopenmp from CFLAGS and LIBFLAGS and add: -fiopenmp fopenmp-targets=spir64
 - Add -g -D__INTEL_COMPILER to CFLAGS

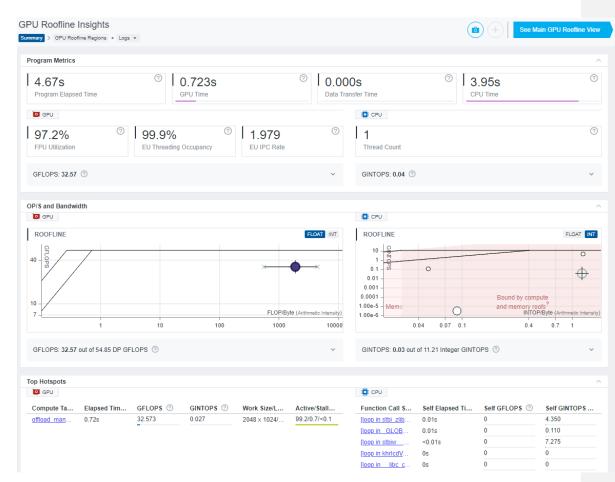
View how we are running compared to system max

- Use Intel[®] Advisor CLI to generate a GPU Roofline report on the offload implementation (option 5):
- advisor --collect=survey --project-dir=./offload_mandel --profile-gpu
 -- /home/uxxxxx/MandelbrotOMP/release/Mandelbrot 5
- advisor --collect=tripcounts --project-dir=./offload_mandel --flop -profile-gpu -- /home/uxxxx/MandelbrotOMP/release/Mandelbrot 5
- advisor -report=roofline -gpu -project-dir=./offload_mandel --reportoutput=./gpu_roofline.html
- Create a snapshot for download to the local GUI:
- advisor --snapshot --project-dir=./offload_mandel --pack --cachesources --cache-binaries -- ./offload_mandel_snapshot

GPU Roofline

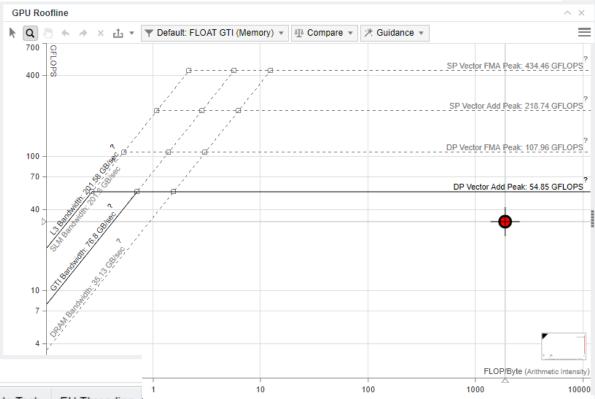
- The overall elapsed time of 4.67s is much higher in the offloaded version than the parallel CPU implementation (1.49s). But the compute task has a speed-up:
- From 1.03s in parallel_mandelbrot to 0.72s in offload_Mandelbrot. Not quite hitting the estimate of 538.2ms.
- Nearly 4s is spent on the CPU

Top Hotspots					
GPU					
Compute Ta	Elapsed Tim	GFLOPS 💿	GINTOPS ③	Work Size/L	Active/Stall
offload man	0.72s	32,573	0.027	2048 x 1024/	99.2/0.7/<0.1



GPU Roofline continued

- The offload task appears to be bounded by the DP Vector Add Peak. Otherwise, it appears to make good use of the GPU.
- EU Array is 99.2% active, and the threading occupancy is almost 100%
- There is an unknown task consuming 3.951s of CPU time with 100% idle GPU time.



_	_		
G	μ	U	
-		-	

Compute Task	Elapsed GPU Compute [≫]		EU Array 🛛 «			Compute Task	EU Threading	
Compute Task	Time	ne Performance		Stalled	Idle	Purpose	Occupancy	
[Outside any task]	3.951s	0.000	0.0%	0.0%	100.0%	[Unknown]	0.0%	
zeCommandListAppendMemoryCopy	0.000s	0.000	0.0%	0.0%	100.0%	Transfer In	0.0%	
zeCommandListAppendBarrier	0.000s	0.000	0.0%	0.0%	100.0%	Synchroniz	0.0%	
offload_mandelbrot\$omp\$offloading:266	0.723s	32.573	99.2%	0.7%	0.0%	Compute	99.9%	

Intel VTune Profiler: GPU Hotspots command-line

- Running gpu-hotspots on the command-line
- vtune –collect gpu-hotspots ./Mandelbrot 5
- Generating a report

Elapsed Time: 4.386s

- GPU Time: 0.682s
- EU Array Stalled/Idle: 0.8%
- GPU L3 Bandwidth Bound: 0.3%
- Hottest GPU Computing Tasks Bound by GPU L3 Bandwidth
- Computing Task Total Time
- -----
- Sampler Busy: 0.0%
- Hottest GPU Computing Tasks with High Sampler Usage
- Computing Task Total Time
- ------
- FPU Utilization: 96.3%

Copy result directory to local system

Hottest GPU Computing Tasks with High FPU Utilization

Intel VTune Profiler: GPU Hotspots

- Once the Intel VTune Profiler is running with the vtune-backend command, open the URL in the browser for the GUI.
- Set the application to /home/uxxxx/MandelbrotOMP/release/ Mandelbrot and set the application parameter to 5.
- Run the GPU Compute/Media Hotspots analysis type

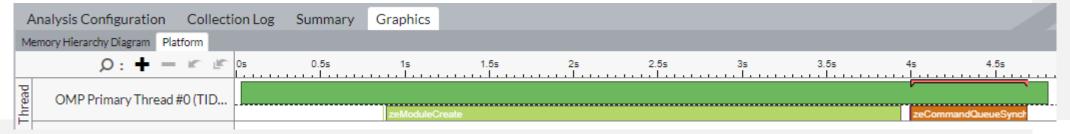
VT Configure Analysis 👔 WHERE	INTEL VTUNE PROFILER
VTune Profiler Server (127.0.0.1)	GPU Compute/Media Hotspots (preview)
WHAT C Launch Application Specify and configure your analysis target: an application or a script to execute. Application	Analyze the most time-consuming GPU kernels, characterize GPU utilization based on GPU hardware metrics, identify performance issues caused by memory latency or inefficient kernel algorithms, and analyze GPU instruction frequency per certain instruction types. <u>Learn more</u> A Measuring peak bandwidth is enabled by default. Accurate peak DRAM bandwidth is important for further analysis but it involves launching a small benchmark which leads to increased collection time. If you would like to omit it, disable this feauture in custom analysis.
Application: /home/u74346/MandelbrotOMP/release/Mandelbrot D Application parameters: 5	Access to /proc/kallsyms file is limited. Consider changing /proc/sys/kernel/kptr_restrict to 0 to enable resolution of OS kernel and kernel module symbols. Overhead Overview
Use application directory as working directory Advanced	GPU sampling interval, ms 1 Analyze memory bandwidth
	Trace GPU programming APIs Source Analysis ③
	Computing task of interest

intei

GPU Hotspots

- The Summary tab shows that although only a small percentage of the overall elapsed time is spent on the GPU, the offload task performs well on the GPU.
- The Graphics tab doesn't indicate any major problems. Under the Platform subtab, there is an OpenMP task called zeModuleCreate that runs for about 3.5s. That explains the high CPU utilization time.

Analysis Configuration	Collection Log	Summary	Graphics	
GPU Time ⁽²⁾ : 0				
🗵 EU Array Sta	alled/Idle [®] :	0.6%		
Section Sectio			ne floating point exe	cution units
-	omputing Tasks w is the most active of	-	Utilization sks that ran on the G	GPU heavily
C	Computing Task		Total Time 💿	



Related Content

GPU Offloading and Profiling Webinars

- Offload Excellence Designing for GPU Performance
- Is your code GPU offload ready?
 - Presentation slides
- Design and tune your applications for GPUs
 - <u>Presentation slides</u> this continues the MandelbrotOMP sample and adds GPU offloading
- High performance GPU acceleration
 - This uses the iso3dfd sample in a similar workflow to today's training

Additional Advisor Content

- Roofline: Optimize for Compute, Memory, or Both?
- Remove Memory Bottlenecks
 - Older, but includes Memory Access Pattern analysis
- Vectorization Advisor
 - Also older, but includes AoS->SoA

Additional VTune Content

Pump up your scaling

More Resources

Intel[®] VTune[™] Profiler – Performance Profiler

- Product page overview, features, FAQs...
- Training materials <u>Cookbooks</u>, <u>User Guide</u>, <u>Processor Tuning Guides</u>
- Support Forum
- Online Service Center Secure Priority Support
- What's New?

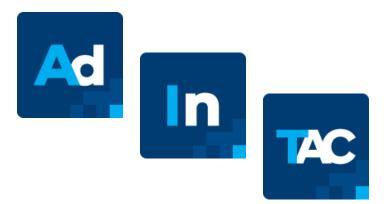
Additional Analysis Tools

- Intel[®] Advisor Design and optimize for efficient vectorization, threading, memory usage, and accelerator offload. Roofline and flow graph analysis.
- Intel[®] Inspector memory and thread checker/ debugger
- Intel[®] Trace Analyzer and Collector MPI Analyzer and Profiler

Additional Development Products

- Intel[®] oneAPI Toolkits
- Intel[®] Software Development Products





How to get

- As part of the oneAPI Base Toolkit:
 - <u>https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit/download.html</u>
- Standalone component:
 - <u>https://software.intel.com/content/www/us/en/develop/articles/oneapi-</u><u>standalone-components.html#vtune</u>
- Linux:
 - Package managers:
 - <u>https://software.intel.com/content/www/us/en/develop/articles/oneapi-</u> <u>repo-instructions.html</u>
 - Containers:
 - <u>https://github.com/intel/oneapi-containers</u>

#