Code that Outperforms

Intel® oneAPI Analyzers

Intel VTune Profiler and Intel Advisor
Agenda

1. Intel® oneAPI Overview
   Introduction to the Intel oneAPI Base and HPC Toolkits

2. Intel VTune Profiler and Intel Advisor Overview
   Overview of the oneAPI analyzers

3. MandelbrotOMP Sample Configuration
   Configure the sample used in the exercises

4. CPU Profiling Exercises
   Running the sample on midway3 with Intel Advisor and Intel VTune Profiler.

5. 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
oneAPI

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

Visit oneapi.com for more details
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

The open source and Intel DPC++/C++ compiler supports Intel CPUs, GPUs, and FPGAs. Codeplay announced a DPC++ compiler that targets Nvidia GPUs.
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 XPU

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: software.intel.com/oneapi for more details

Some capabilities may differ per architecture and custom-tuning will still be required. Other accelerators to be supported in the future.
# 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

<table>
<thead>
<tr>
<th>Direct Programming</th>
<th>API-Based Programming</th>
<th>Analysis &amp; debug Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® oneAPI DPC++/C++ Compiler</td>
<td>Intel® oneAPI DPC++ Library oneDPL</td>
<td>Intel® VTune™ Profiler</td>
</tr>
<tr>
<td>Intel® DPC++ Compatibility Tool</td>
<td>Intel® oneAPI Math Kernel Library - oneMKL</td>
<td>Intel® Advisor</td>
</tr>
<tr>
<td>Intel® Distribution for Python</td>
<td>Intel® oneAPI Data Analytics Library - oneDAL</td>
<td>Intel® Distribution for GDB</td>
</tr>
<tr>
<td>Intel® FPGA Add-on for oneAPI Base Toolkit</td>
<td>Intel® oneAPI Threading Building Blocks - oneTBB</td>
<td></td>
</tr>
</tbody>
</table>

Learn More: [intel.com/oneAPI-BaseKit](intel.com/oneAPI-BaseKit)
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

Learn More: intel.com/oneAPI-HPKit
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 re- compilation
- 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
Intel® VTune™ Profiler

Algorithm Optimization
✓ Hotspots
✓ Anomaly Detection
✓ Memory Consumption

Microarch.&Memory Bottlenecks
✓ Microarchitecture Exploration
✓ Memory Access

Accelerators / xPU
✓ GPU Offload
✓ GPU Compute / Media Hotspots
✓ CPU/FPGA Interaction

Parallelism
✓ Threading
✓ HPC Performance Characterization

Platform & I/O
✓ Input and Output
✓ System Overview
✓ Platform Profiler

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)

Filter by Process & Other Controls
Tuning Opportunities Shown in Pink. Hover for Tips
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
1) 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.

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
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

![Image of Microarchitecture Exploration](image-url)
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 device-agnostic hardware event-based metrics for DRAM, PMEM, Intel UPI, PCIe, Intel DDIO, MMIO traffic consumption

- **API and OS I/O**: DPDK, SPDK, kernel I/O

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

✓ Start from the functions at the bottom and work your way up
✓ Pay attention to wide rectangles for hot/sync functions

*Adapted based on Brendan Gregg’s Flame Graphs
Flame Graph

Each rectangle represents a stack frame and function total CPU time – Top-Down (Aggregated data NOT overtime)

The horizontal-axis shows the stack profile population, sorted alphabetically

The vertical-axis shows stack depth, counting from zero at the bottom

Click rectangle to zoom

User/System/Threading runtime overhead/ Sync coloring to comprehend the App structure

Find the Hottest code-path(s) and function(s)
Start optimization from the bottom functions to top
Pay attention to Hot/Wide Sync function(s) too

Filter by Process/Thread/Module/FunctionType/Time via Filter bar or/and Timeline
Choose your next analysis:

Characterize high-level aspects:

- **Elapsed Time**: 7.906s
- **Logical Core Utilization**: 12.4% (0.990 out of 8)
- **Microarchitecture Usage**: 27.2% of Pipeline Slots
  - Retiring: 27.2% of Pipeline Slots
  - Front-End Bound: 5.5% of Pipeline Slots
  - Bad Speculation: 3.7% of Pipeline Slots
  - Back-End Bound: 63.8% of Pipeline Slots
  - Memory Bound: 35.2% of Pipeline Slots
  - Core Bound: 28.4% of Pipeline Slots
- **Memory Bound**: 35.2% of Pipeline Slots
- **Vectorization**: 0.0% of Packed FP Operations
Intel® VTune™ Profiler
Application Performance Snapshot for MPI

- Outlier analysis for MPI applications at scale
  - Explore on source of imbalance
  - Choose nodes/ranks for detailed 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

Learn more: intel.com/advisor
“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 Intel® Advisor
  ▪ 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 Intel® Advisor.
  ▪ Consider data layout changes Intel® SIMD Data Layout Templates can help

Arrays of structures are great for intuitively organizing data, but are much less efficient than structures of arrays. Use the Intel® SIMD Data Layout Templates (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

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

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.
Log into DevCloud via ssh

Start interactive gpu node:

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

Create MandelbrotOMP sample:

https://github.com/oneapi-src/oneAPI-samples

Start Intel VTune Profiler Server in second ssh terminal

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:

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 to estimate performance on Gen9 GT2 GPU
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/oneAPI-samples/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

```makefile
CXX := icpx
SRCDIR := src
BUILDDIR := release
CFLAGS := -O3 -ipo -qopenmp -std=c++11
   -g -D__INTEL_COMPILER
EXTRA_CFLAGS :=
LIBFLAGS := -qopenmp
ifdef perf_num
   ____EXTRA_CFLAGS += -D PERF_NUM
endif
TARGET := $(BUILDDIR)/Mandelbrot
icpx: $(TARGET)
```
MandelbrotOMP main.cpp

Increase workload size

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

```cpp
// 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
Collect CPU Hotspots

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

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

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
Collect CPU Hotspots

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

![Effective CPU Utilization Histogram](image-url)
Collect CPU Hotspots

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
Collect HPC Performance Characterization

Generate VTune Command Line

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

$ 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
Intel® Advisor Exercise

MandelbrotOMP Sample
Collect CPU Roofline

Run from GUI

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

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
Collect CPU Roofline

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
Collect CPU Roofline

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.
Collect CPU Roofline

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 LLVM-based compiler - which is why some vectorization details are missing. This is being worked on.
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 --target-device=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!
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_parallel_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
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/uxxxxx/MandelbrotOMP/release/Mandelbrot 5`
- `advisor -report=roofline -gpu -project-dir=./offload_mandel --report-output=./gpu_roofline.html`

Create a snapshot for download to the local GUI:

- `advisor --snapshot --project-dir=./offload_mandel --pack --cache-sources --cache-binaries -- ./offload_mandel_snapshot`
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.
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.
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
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/uxxxxx/MandelbrotOMP/release/Mandelbrot and set the application parameter to 5.
- Run the GPU Compute/Media Hotspots analysis type
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 sub-tab, there is an OpenMP task called `zeModuleCreate` that runs for about 3.5s. That explains the high CPU utilization 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](#)
  - Presentation slides – 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 – Cookbooks, User Guide, Processor Tuning Guides
- 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:

• Standalone component:
  • https://software.intel.com/content/www/us/en/develop/articles/oneapi-standalone-components.html#vtune

• Linux:
  • Package managers:
  • Containers:
    • https://github.com/intel/oneapi-containers