Agenda

Introduction
Overview of Analysis Tools

Intel® Advisor
GPU Offload Modeling, GPU Roofline Analysis

VTune™ Profiler
Work Scheduling Balancing, Data Transfer, GPU Occupancy

Recommended Workflow
When are Advisor and VTune most helpful?

Q&A
Introduction
Overview of Analysis Tools
Intel® Advisor

Offload Advisor
Design your code for efficient GPU offload, even before you have the hardware

Automated Roofline Analysis
See performance headroom against hardware limitations. Get insights for an effective optimization roadmap.

Vectorization Optimization
Enable more vector parallelism and improve its efficiency

Thread Prototyping
Model, tune and test multiple threading designs

Build Heterogeneous Algorithms
Create and analyze data flow and dependency computation graphs

Learn More: software.intel.com/advisor
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
Intel® Advisor
Offload Advisor

- Run on CPU or GPU – Predict for GPU
- Helps to define which section of the code should run on given accelerator
- Provides performance projection on accelerators

- Offloading is GTI BW Bound
- Recommended for offloading
- Not Profitable for offloading
In-Depth Analysis of Top Offload Regions

Intel® Advisor

Loop at 1_CPU_only.cpp:22 is recommended for offloading
- GTI BW Bound
- Estimated to run on GPU in 94.5ms

```c
// Copyright Intel Corporation
#include <chrono>
#include <string>
#include "util.h"

void inline isodd(Iteration(float* ptr_next_base, float* ptr_prev_base,
float* ptr1_base, float* coeff, const size_t n1, const size_t n2, const size_t n3) {
auto dimn1 = n1 * n2;

// Remove YLD from the end
auto n2_end = n3 - kMemStride;
auto n1_end = n3 - kMemStride;

for (auto iy = n1_end; iy < n_end; iy++) {
    for (auto ix = n1_end; ix < n_end; ix++) {
        // Calculate start pointers for the row over X dimension
        float* ptr_next = ptr_next_base + (iy * n1_end + ix); 
    }
}
```
Focus optimization effort where it makes the most difference

- Large, red loops have the most impact
- Loops far from the upper roofs have more room to improve

Additional roofs can be plotted for specific computation types or cache levels
GPU Analysis Flow

Intel® Advisor

Survey Analysis
#1 app run
- GPU Kernels list, timings
- MDAPI collection (hardware counters)

Characterization Analysis
#2 app run
- Measure GPU hardware maximum capabilities
- Binary instrumentation with GTPIN collection

Performance Modeling
No run
- Analyze performance metrics

GPU Roofline Perspective
- Performance Metrics
- Roofline
- Kernel Details
- Guidance
- Recommendations
GPU Roofline Chart
Intel® Advisor

Customizable GPU Roofline Chart

GPU performance of compute tasks

View GPU Details, Data Transfer, GPU Source and Assembly info

Customizable GPU Roofline Chart

GPU performance of compute tasks

View GPU Details, Data Transfer, GPU Source and Assembly info
Profiling/Optimizing GPU Offload Application using Intel® VTune™ Profiler

Rupak Roy– Technical Consulting Engineer
Xiao Zhu– Technical Consulting Engineer
Cory Levels – Technical Consulting Engineer
Agenda

• GPU Profiling Overview
• GPU Performance Issues
• OpenMP Offload Case Study
• Conclusion
Intel® VTune™ Profiler

HPC Performance Characterization

A starting point for performance optimization
- CPU/GPU usage, Memory efficiency, and Floating-point utilization
GPU Performance Problems

Addressing performance issues with dynamic analysis tools

• Work Distribution
• Data transfer
• GPU occupancy
• Memory access
• Kernel inefficiencies
• Non-scaling implementations
• ...

![Image of dynamic analysis tools and performance metrics]
Work Distribution

Work distribution among computing resources

- CPU or GPU bound?
- GPU Utilization for OpenMP regions
- EU/XVEs efficiency (Active, Stalled, Idle)
- Offload Time characterization
  - Compute
  - Data Transfer
  - Overhead
Host and GPU Data Transferring

A commonly known problem of host-to-device transfer performance

- Data transfer time
- Amount of transferred data
- Transfer direction
- Execution time
### Graphics View of GPU Offload

**INTEL VTUNE PROFILER**

#### Grouping: (custom) Computing Task

<table>
<thead>
<tr>
<th>Computing Task</th>
<th>Total Time by Device Operation Type</th>
<th>Instance Count</th>
<th>Transfer Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Execution</td>
<td>Host-to-Device Transfer</td>
<td>Device-to-Host Transfer</td>
</tr>
<tr>
<td>matrixMultiply&lt;float, unsig&gt;</td>
<td>0.053s</td>
<td>44.528s</td>
<td>7.362s</td>
</tr>
<tr>
<td>zeCommandListAppend Barb</td>
<td>0s</td>
<td>0s</td>
<td>0s</td>
</tr>
<tr>
<td>[Outside any task]</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Graphics View**

- **Thread**
  - matrix_multiply (TID: 13204...)
  - matrix_multiply (TID: 13205...)
  - matrix_multiply (TID: 13205...)

- **GPU Execution Units**
  - GPU Execution Units
  - GPU Computing Threads Distribution
  - GPU Utilization
  - CPU Time
Achieving High XVE Threads Occupancy

Occupancy analysis helps identifying ground problems with work mapping

- Detecting workgroups by global and local sizes
- SIMD Width
- Barriers usage
- Tiny/huge kernels scheduling issues
Memory Access problems

- Global memory access penalty
- Cache memory resource limit

- Which code is responsible for latency?
- Per Basic Block and latencies per individual instructions
Source level in-kernel profiling

Basic Block Latency

Memory Latency
Kernel code optimizations

Advanced code optimizations on kernel level

• Are FPUs and EM pipelines fully utilized?
• How are the systolic instructions used in AI application?

• Instructions counting profiles
• FPU and XMX pipeline Utilization
Source level in-kernel profiling
Case Study (OpenMP* Offload on GPU)

- Application Name: iso3dfd OpenMP Offload
- Profile the baseline version
- Detect the bottlenecks
- Make code changes based on the findings
- Compare baseline vs optimized
HPC Characterization

CPU Stack Utilization:

- EU State:
  - Active: 28.8%
  - Stalled: 71.1%
  - Idle: 0.1%
- Occupancy: 99.2% of peak value
- Offload Time: 31.9% (18.907s) of elapsed time
  - Compute: 96.5% (18.250s) of offload time
  - Data Transfer: 1.5% (0.281s) of offload time
  - Overhead: 2.0% (0.376s) of offload time

Top OpenMP Offload Regions:

<table>
<thead>
<tr>
<th>OpenMP Offload Region</th>
<th>Offload Time</th>
<th>Percentage of Elapsed Time</th>
<th>Data Transfer</th>
<th>Overhead</th>
<th>EU Array Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>iso3dfdIteration$omp$target$region;dvc=0@/home/intel/rroy/oneAPI-samples/Dir...</td>
<td>18.252s</td>
<td>30.8%</td>
<td>0s</td>
<td>0.001s</td>
<td>28.8%</td>
</tr>
<tr>
<td>iso3dfd$omp$target$region;dvc=0@/home/intel/rroy/oneAPI-samples/Dir...</td>
<td>0.655s</td>
<td>1.1%</td>
<td>0.281s</td>
<td>0.374s</td>
<td>0.0%</td>
</tr>
<tr>
<td>[Outside any OpenMP Offload Region]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*IVA is applied to non-summable metrics.
## HPC Characterization Bottom-up View for Baseline Version

### HPC Performance Characterization

#### Analysis Configuration

**Grouping:** OpenMP Offload Region / Function / Call Stack

<table>
<thead>
<tr>
<th>OpenMP Offload Region / Function / Call Stack</th>
<th>OpenMP Offload Time</th>
<th>Instance Count</th>
<th>EU State</th>
<th>Occup...</th>
<th>CPU Time</th>
<th>Serial CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iso3dfclliteration$omp$target$region:dvc=0@/home/int</td>
<td>18.252s</td>
<td>100</td>
<td>28.8%</td>
<td>71.1%</td>
<td>0.1%</td>
<td>99.2%</td>
</tr>
<tr>
<td>Iso3dfcl$omp$target$region:dvc=0@/home/int</td>
<td>0.655s</td>
<td>2</td>
<td>0.0%</td>
<td>0.0%</td>
<td>100.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>[Outside any OpenMP Offload Region]</td>
<td></td>
<td></td>
<td>0.0%</td>
<td>0.1%</td>
<td>99.9%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Process / OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack</td>
<td>Elapsed Time</td>
<td>SP GFLOPS</td>
<td>Serial CPU Time</td>
<td>Memory Bound</td>
<td>Imbalance</td>
<td>Lock Contention</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>iso3dfd</td>
<td>59.330s</td>
<td>13.415</td>
<td>21.425s</td>
<td>28.9%</td>
<td>0.816s</td>
<td>0s</td>
</tr>
<tr>
<td>Z22Iso3dfdVerifyIterationPfs_S_S_iimm</td>
<td>37.481s</td>
<td>21.222</td>
<td>0s</td>
<td>30.3%</td>
<td>0.816s</td>
<td>0s</td>
</tr>
<tr>
<td>Iso3dfdVerifyIteration$omp$loop_barrier</td>
<td>37.479s</td>
<td>21.222</td>
<td>0s</td>
<td>30.3%</td>
<td>0.816s</td>
<td>0s</td>
</tr>
<tr>
<td>INTERNALbae80b52::_kmp_for_stat</td>
<td>0.000</td>
<td>0.000</td>
<td>0s</td>
<td>0.0%</td>
<td>0.000</td>
<td>0s</td>
</tr>
<tr>
<td>__kmp_get_global_thread_id</td>
<td>0.000</td>
<td>0.000</td>
<td>0s</td>
<td>100.0%</td>
<td>0.000</td>
<td>0s</td>
</tr>
<tr>
<td>[Loop@0x1677e in oa_buffer_check_un</td>
<td>0.000</td>
<td>0.000</td>
<td>0s</td>
<td>0.0%</td>
<td>0.000</td>
<td>0s</td>
</tr>
<tr>
<td>oa_buffer_check_unlocked</td>
<td>0.000</td>
<td>0.000</td>
<td>0s</td>
<td>0.0%</td>
<td>0.000</td>
<td>0s</td>
</tr>
</tbody>
</table>
GPU-Offload Platform View for Baseline Version
### GPU-Offload Graphics View for Baseline Version

#### Grouping: GPU Computing Task / Host Call Stack

<table>
<thead>
<tr>
<th>GPU Computing Task / Host Call Stack</th>
<th>Total Time by Device Operation Type</th>
<th>Transfer Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Allocation</strong></td>
<td><strong>Host-to-Device</strong></td>
</tr>
<tr>
<td>Iso3dfdoIteration$omp$offloading:500</td>
<td>18.584s</td>
<td>0 B</td>
</tr>
<tr>
<td>[Outside any task]</td>
<td>0.001s</td>
<td>5.1 KB</td>
</tr>
<tr>
<td>_tgt_target_data_begin_mapper</td>
<td>0.001s</td>
<td>5 KB</td>
</tr>
<tr>
<td>[Outside any task] ← [Outside any]</td>
<td>0.000s</td>
<td>0 B</td>
</tr>
<tr>
<td>[Unknown stack frame(s)] ← [tgt]</td>
<td>0.000s</td>
<td>64 B</td>
</tr>
</tbody>
</table>
GPU-Hotspots Graphics View for Baseline Version
Code Optimization

- Added ‘omp target teams distribute’ construct
- Distribute the work among different teams
- Store variables in registers for reuse
- Add OpenMP CPU Threads
GPU-Hotspots Summary View for Optimized Version

**Elapsed Time**: 53.246s

**EU Array Stalled/Idle**: 67.2% of Elapsed time with GPU busy

- **GPU L3 Bandwidth Bound**: 79.2% of peak value
- **HotTest GPU Computing Tasks Bound by GPU L3 Bandwidth**

**Occupancy**: 96.4% of peak value

**Sampler Busy**: 9.0% of peak value
GPU-Hotspots Graphics View for Optimized Version
Recommended Workflow
Using Intel® Analyzers to increase performance

Use Offload Advisor to find kernels to offload

Optimize your kernels With Advisor and VTune