Modeling Heterogeneous Computing Performance with Offload Advisor

Vladimir Tsymbal (Intel) - presenter
Cédric Andreolli (Intel)
Zakhar Matveev (Intel)
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

IWOC '20, April 27–29, 2020, Munich, Germany
© 2020 Copyright is held by the owner/author(s).
https://doi.org/10.1145/3388333.3388665
Agenda

- Introduction to Offload Advisor
- Command line tips
- Understanding the performance modelization
- GPU Roofline Analysis
Introduction to Offload Advisor
Intel Offload Advisor (Beta)

- Starting from a baseline binary (running on CPU):
  - Helps defining which sections of the code should run on a given accelerator
  - Provides performance projection on accelerators (currently gen9 and gen11)
Modeling Flows supported: NOW

Baseline HW (Programming model)

1. CPU measured (C,C++,Fortran, Py)

1.a CPU measured (DPC++, OCL, OMP, “target=host”)

Target HW

CPU measured

GPU estimated

CPU measured

GPU estimated
Modeling Flows supported: NOW + Coming Soon

**Baseline HW** (Programming model)

1. **CPU** measured
   - (C, C++, Fortran, Py)
2. **CPU+iGPU** measured
   - (C, C++, Fortran, DPC++, OCL, OMP)

**Target HW**

- **CPU** measured + **GPU** estimated
- **CPU** measured + **GPU** estimated
- **CPU** measured + **GPU** estimated

*Other names and brands may be claimed as the property of others.*
From Your CPU Application, you wonder:

- How your code might perform on an accelerator?

![Program metrics](image)

- What might be limiting your performance on the accelerator?

![Offloads bounded by](image)

- What should you offload?

![Top offloaded](image)

- What are the bad candidates for offload and Why?

![Top non offloaded](image)
Top Offloaded in depth

- Provides a detailed description of each loop interesting for offload
  - Timings (total time, time on the accelerator, speedup)
  - Offload metrics (offload taxe, data transfers)
  - Memory traffic (DRAM, L3, L2, L1), trip count
  - Highlight which part of the code should run on the accelerator

This is where you will use DPCPP or OMP target for offload
Non Offloaded in depth

- Explains why Advisor doesn’t recommend a given loop for offload
  - Dependency issues
  - Not profitable
  - Total time is too small
Program Tree

- The program tree offers another view of the proportion of code that can be offloaded to the accelerator.
Command Line Tips
Before you start to use Offload Advisor

- The only strict requirement for compilation and linking is full debug information:
  - `-g:` Requests full debug information (compiler and linker)

- Offload Advisor supports any optimization level, but the following settings are considered the optimal requirements:
  - `-O2:` Requests moderate optimization
  - `-no-ipo:` Disables inter-procedural optimizations that may inhibit Offload Advisor to collect performance data (Intel® C++ & Fortran Compiler specific)
To set up the Intel® Advisor Beta environment, run one of the shell script:

source <ONEAPI_INSTALL_DIR>/setvars.sh

or

source <ADV_INSTALL_DIR>/env/vars.sh

This script sets all required Intel Advisor environment variables, including APM, which points to <ADV_INSTALL_DIR>/perfmodels

This is the location of the Offload Advisor scripts in the Intel® Advisor Beta installation directory

The performance modeling functionality is available on Linux* OS only
How does it work?

- Easy to collect data and generate output with batch mode:
  
  ```bash
  advixe-python <ADV_INSTALL_DIR>/perfmodels/run_oa.py <path_to_result_dir> --config gen9 --out-dir <path_to_result_dir> [--options] -- <app>
  ```

- By default, `run_oa.py` marks up all regions and only selects the most profitable ones for analysis.

- To generate the report.html, uses the following command:
  
  ```bash
  advixe-python $APM/analyse.py <project_dir> --config gen9 [--options] -- <app_binary> [app_options]
  ```

```
ls
accelerators analyze.py collect.py debug.so environ.py oa_wrapper.so shared.so toml analyze_impl.so collect_impl.so compute_stats.py dot_graph.so helpers run_oa.py template tree.so
```
Run_oa.py: What is running behind?

- Run_oa.py
- Python collect.py --a gen OR use directly advixe-cl
- Python analyze.py --a gen

**Application**
- Application (-G -02)

**Advisor Profiling Tool**
- Survey

**Mark-Up Policy Applied**

**Advisor Profiling Tool**
- Dynamic Instrumentation
- FLOPS/Trip Counts
- Traffic Simulation
- Offload data transfer analysis
- Binary static analysis
- *Dependencies

**Advisor: Intel GPU Performance Models**

**Model Outputs**
Offload advisor Output Overview

- **report.html**: Main report in HTML format
- **report.csv** and **whole_app_metric.csv**: Comma-separated CSV files
- **program_tree.dot**: A graphical representation of the call tree showing the offloadable and accelerated regions
- **program_tree.pdf**: A graphical representation of the call tree
  Generated if the DOT(GraphViz*) utility is installed
  1:1 conversion from the **program_tree.dot** file
- **JSON** and **LOG** files that contain data used to generate the HTML report and logs, primarily used for debugging and reporting bugs and issues
Want to avoid dependency checking?

- Dependency adds a lot of time to the collection and you might want to remove it.
- Add the option `–c basic` for the collection:

  ```bash
  advixe-python <ADV_INSTALL_DIR>/perfmodels/run_oa.py <path_to_result_dir> -config gen9 -c basic --out-dir <path_to_result_dir> [--options] -- <app>
  ```

- Add the option `--assume-parallel` for the analysis:

  ```bash
  advixe-python $APM/analyse.py <project_dir> --assume-parallel --config gen9 [--options] -- <app_binary> [app_options]
  ```
Understanding the performance modelization
The mechanisms behind 1/2

First order analytical modeling pillars:
- Compute throughput model
- Memory sub-system model
- Offload data transfer modeling

**Execution time on baseline platform (CPU)**
- Execution time on accelerator. Estimate assuming bound exclusively by Compute
- Execution time on accelerator. Estimate assuming bound exclusively by caches/memory
- Offload Tax estimate (data transfer + invoke)

Final estimated time on target platform (eg GPU)

\[ t_{\text{region}} = \max(t_{\text{compute}}, t_{\text{memory subsystem}}) + t_{\text{data transfer tax}} + t_{\text{kernel launch}} \]

Region X
- Profitable to accelerate, \( t(X) > t(X') \)

Region Y
- Too much overhead, not accelerable, \( t(Y) < t(Y') \)
We minimize the total time spent in this loop hierarchy by varying offload strategies $U$ (offload/non-offload, #threads for each component $loop_i$ of loopnest).

Objective function: $T_{all} = \min_{U=\{u_1, u_2, \ldots\}} (\sum_i T_i + t_{data\ transfer} + t_{invoke} + T_{cpu})$

$$T_i = \max \left\{ \begin{array}{l}
T^\text{Comp\_only}(\ ) \\
T^\text{M\_only}(M^k_i) = \frac{M^k_i}{BW_k}
\end{array} \right.$$  

Under algorithmic constraints (Dependencies and TripCount/Granularity)
GPU Roofline Analysis
Intel® Gen9 Memory Hierarchy

- Intel® Graphics Compute Architecture uses the same DRAM with the CPU
- Level-3 (L3) data cache: slice-shared asset
- Shared Local Memory (SLM): a dedicated structure within the L3 that supports the work-group local memory address space
- Graphics Technology Interface (GTI): a dedicated interface unit connects the entire architecture interfaces to the rest of the SoC components
- The rest of SoC memory hierarchy includes the large Last-Level Cache (LLC, which is shared between CPU and GPU), possibly embedded DRAM and finally the system DRAM

A view of the SoC chip level memory hierarchy and its theoretical peak bandwidths for the compute architecture of Intel processor graphics gen9
Find Effective Optimization Strategies

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
How to run?

The Roofline model on GPU is a technical preview feature and is not available by default.

To enable it:

```bash
export ADVIXE_EXPERIMENTAL=gpu-profiling
```

To run the GPU Roofline analysis in the Intel® Advisor CLI:

Run the Survey analysis with the `--enable-gpu-profiling` option:

```bash
advixe-cl --collect=survey --enable-gpu-profiling --project-dir=<my_project_directory> --search-dir src:r=<my_source_directory> -- ./myapp [app_parameters]
```

Run the Trip Counts and FLOP analysis with `--enable-gpu-profiling` option:

```bash
advixe-cl --collect=tripcounts --stacks --flop --enable-gpu-profiling --project-dir=<my_project_directory> --search-dir src:r=<my_source_directory> -- ./myapp [app_parameters]
```

Generate a GPU Roofline report:

```bash
advixe-cl --report=roofline --gpu --project-dir=<my_project_directory> --report-output=roofline.html
```

Open the generated roofline.html in a web browser to visualize GPU performance.
Roofline Analysis on Intel® GPU

Matrix2<float>
Self Performance: 8.02 GFLOPS
Self GTI Arithmetic Intensity: 4.56 FLOP/Byte
Self Elapsed Time: 0.268 s
Self Memory Traffic: 0.471 GB
Legal notices & disclaimers

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Statements in this document that refer to Intel’s plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel’s results and plans is included in Intel’s SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, the Intel logo, Pentium, Celeron, Atom, Core, Xeon, Movidius and others are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2018 Intel Corporation.