INTEL® ADVISOR FOR OPENMP ON CPU AND INTEGRATED GPU

Kirill Rogozhin, November 2019
Intel Advisor

Roofline: Characterize and optimize on **CPU**

Vectorization

Memory

Threading

Offload Advisor: identify best offload candidates

Offload Advisor: identify best offload candidates

Roofline: Characterize and optimize on **GPU**

**omp target**

**omp teams**

**omp map...**

**omp simd**

**omp declare simd ...**

**omp parallel_for**

**omp task ...**
Advisor Roofline

Characterize and optimize CPU code
What is the Roofline Model?

Characterization of your application performance in the context of the hardware

It uses two simple metrics
- Flop count
- Bytes transferred

\[ a_i = b_i + c_i \times d_i \]

1W+3R = 4*4 bytes = 16 bytes

The maximum FLOPS as a product of ops/byte (AI) and maximum bytes supplied per second is a **diagonal** line.

The CPU’s maximum FLOPS can be plotted as a **horizontal** line.

FLOPS

Arithmetic Intensity

FLOPS/Byte
The Roofline Chart in Intel® Advisor

Focus on big dots with headroom.

- Intel® Advisor sizes and color-codes dots by relative time taken.
- Space between dots and their uppermost limits is room to improve.

Roofs above a dot indicate potential sources of bottlenecks.
## Getting Roofline data in Intel® Advisor

### Step 1: Survey
- **Collect survey**
- Provide \#Seconds
- **Root access not needed**
- User mode sampling, non-intrusive.

### Step 2: FLOPS
- **Collect tripcounts –flops**
- Provide \#FLOP, \#Bytes, AVX-512 Mask
- **Root access not needed**
- Precise, instrumentation based, count number of instructions

### Roofline:

<table>
<thead>
<tr>
<th>Axis X: ( AI = \frac{#FLOP}{#Bytes} )</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis Y: ( \frac{FLOP/S}{#Seconds} = \frac{#FLOP (mask aware)}{#Seconds} )</td>
<td>1x</td>
</tr>
</tbody>
</table>

**Overhead:**
- 1x
- 5-10x
Questions to answer with Roofline

1. Am I doing well? How far am I from the pick?
   (do I utilize hardware well or not?)

2. Final Bottleneck?
   (where will be my limit after I done all optimizations?)
   Long-term ROI, optimization strategy

- Memory-bound, invest into cache blocking etc
- Compute bound: invest into SIMD,..
Advisor Roofline

Characterize and optimize CPU code: Threading
Roofline: iso3dfd example

Adjust roofs to #cores, ranks, NUMA in your OpenMP application:

- `OMP_NUM_THREADS`
- `KMP_AFFINITY`
- `KMP_HW_SUBSET`

Hotspot OpenMP loop

```
[loop in iso_3dfd_it at iso-3dfd_parallel.cc:43]
Scalar: processes Float32 data type(s)
Performance: 0.326 GFLOPS
CARM (L1 + NTS) Arithmetic Intensity: 0.201 FLOP/Byte
Self Time: 343.630 s
Self Elapsed Time: 343.630 s
Total Time: 343.630 s
Total Elapsed Time: 343.630 s
Self Memory Traffic: 557.826 GB
Total Memory Traffic: 657.826 GB
```
Roofline: iso3dfd example

Top max GFLOPS limit increases with number of threads

Compare with previous version

Visualized performance difference after adding #pragma omp parallel for
Advisor Roofline

Characterize and optimize CPU code: Memory
CARM Roofline Guidance: either DRAM or LLC is the bottleneck
Memory access pattern analysis
How should I access data?

Unit stride access are faster

```
for (i=0; i<N; i++)
    A[i] = B[i]*d
```

Constant stride are usually worse

```
for (i=0; i<N; i+=2)
    A[i] = B[i]*d
```

Non predictable access are usually bad

```
for (i=0; i<N; i++)
    A[i] = B[C[i]]*d
```
# Memory Access Patterns

## Strided Access in OpenMP Region

**Memory Object**

**Strided access in OpenMP region**

<table>
<thead>
<tr>
<th>ID</th>
<th>Stride</th>
<th>Type</th>
<th>Nested Function</th>
<th>Variable References</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>65536</td>
<td>Constant stride</td>
<td>iso-3dfd_parallel.cc47</td>
<td>block 0x7fd89ffe010 allocated at iso-3dfd_main.cc:184, block 0x7</td>
</tr>
<tr>
<td>P2</td>
<td>65536</td>
<td>Constant stride</td>
<td>iso-3dfd_parallel.cc49</td>
<td>block 0x7fd89ffe010 allocated at iso-3dfd_main.cc:184, block 0x7</td>
</tr>
<tr>
<td>P3</td>
<td>65536</td>
<td>Constant stride</td>
<td>iso-3dfd_parallel.cc50</td>
<td>block 0x7fd89ffe010 allocated at iso-3dfd_main.cc:184, block 0x7</td>
</tr>
<tr>
<td>P4</td>
<td>65536</td>
<td>Constant stride</td>
<td>iso-3dfd_parallel.cc51</td>
<td>block 0x7fd89ffe010 allocated at iso-3dfd_main.cc:184, block 0x7</td>
</tr>
</tbody>
</table>

```c
47    value += ptr_prev[offset]*coeff[0];
48    for(int ir=1; ir<=HALF_LENGTH; ir++) {
49    value += coeff[ir] * (ptr_prev[offset + ir] + ptr_prev[offset - ir]); // horizontal
50    value += coeff[ir] * (ptr_prev[offset + ir*1] + ptr_prev[offset - ir*1]); // vertical
51    value += coeff[ir] * (ptr_prev[offset + ir*dim1n2] + ptr_prev[offset - ir*dim1n2]);
```
Advisor Roofline

Characterize and optimize CPU code: Vectorization
Advisor Vectorization analysis

Filter by which loops are vectorized!

Trip Counts

What prevents vectorization?

Focus on hot loops

What vectorization issues do I have?

Which Vector instructions are being used?

How efficient is the code?
Enabling Vectorization

1. Check dependencies

2. Use `#pragma omp simd`

3. Table showing vector issues, self time, total time, type, and vectorized loops with AVX512, efficiency, and gain.
Offload Advisor

Define regions to be offloaded
# Offload Advisor Summary

## Intel® Advisor Beta

### Offload Advisor

#### Summary

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Up for Accelerated Code</td>
<td>4.2x</td>
</tr>
<tr>
<td>Number of Offloads</td>
<td>2</td>
</tr>
<tr>
<td>Fraction of Accelerated Code</td>
<td>95%</td>
</tr>
</tbody>
</table>

#### Program metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>7.04s</td>
</tr>
<tr>
<td>Accelerated</td>
<td>1.92s</td>
</tr>
<tr>
<td>Target Platform</td>
<td>Gen11 GT2</td>
</tr>
<tr>
<td>Number of Offloads</td>
<td>2</td>
</tr>
<tr>
<td>Speed Up for Accelerated Code</td>
<td>4.2x</td>
</tr>
<tr>
<td>Amdahl's Law Speed Up</td>
<td>3.6x</td>
</tr>
<tr>
<td>Fraction of Accelerated Code</td>
<td>95%</td>
</tr>
<tr>
<td>Time on Host</td>
<td>0.30s</td>
</tr>
<tr>
<td>Time on Accelerator</td>
<td>1.61s</td>
</tr>
<tr>
<td>Data Transfer Tax</td>
<td>0s</td>
</tr>
<tr>
<td>Invocation Tax</td>
<td>&lt;0.01s</td>
</tr>
<tr>
<td>Code Transfer Tax</td>
<td>&lt;0.01s</td>
</tr>
</tbody>
</table>
## Explore offload candidates

### Top offloaded

<table>
<thead>
<tr>
<th>Location</th>
<th>Speed Up</th>
<th>Bounded By</th>
<th>Data Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>[loop in iso_3dfd_it at iso3dfd.cpp:233]</td>
<td>4.27x</td>
<td>CPU 6.88s</td>
<td>DRAM_BW 465.83MB</td>
</tr>
<tr>
<td>[loop in initializeFT$omp$parallel@51 at iso3dfd.cpp:59]</td>
<td>1.10x</td>
<td>CPU 0.06s</td>
<td>DRAM_BW 357.97MB</td>
</tr>
</tbody>
</table>

- CPU: 6.88s
- GPU: 1.56s
- CPU: 0.06s
- GPU: 0.05s

**Performance bounding factors**

- Total Execution Time by Compute: <0.01s
- Total Execution Time by L3 BW (s): 0.010
- Total Execution Time by LLC BW (s): 0.039
- Total Execution Time by Memory BW Time (s): 0.055
- Data Transfer Tax(s): 0
- Invocation Tax(s): <0.00001
- Kernel Code Transfer Tax(s): <0.00001
Offloaded regions details and source view

The region can be **4.27x** faster on Gen11

Add **“#pragma omp target”** here
## Data transfer and memory mapping

<table>
<thead>
<tr>
<th>Data Transfer (MB)</th>
<th>Total Data Transferred from CPU to GPU (MB)</th>
<th>Total Data Transferred from GPU to CPU (MB)</th>
<th>Memory Mapped To Device (MB)</th>
<th>Memory Mapped From Device (MB)</th>
<th>Memory Mapped ToFrom Device (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>465.83MB</td>
<td>232.91</td>
<td>232.91</td>
<td>0</td>
<td>0</td>
<td>232.91494</td>
</tr>
<tr>
<td>357.97MB</td>
<td>116.47</td>
<td>241.50</td>
<td>0</td>
<td>125.02630</td>
<td>116.46976</td>
</tr>
</tbody>
</table>

*Total estimated data traffic*
GPU Roofline

Characterize and optimize GPU code
Roofline for GPU

- Preview feature, supports Gen9 GPU
- Command line data collection
- HTML export as UI

Memory traffic across all memory levels on GPU
GPU Roofline

Tiled version: higher FLOPS, higher FLOP/byte

Basic version
Intel Advisor

Roofline: Characterize and optimize on CPU

Offload Advisor: identify best offload candidates

Roofline: Characterize and optimize on GPU

Vectorization

Memory

Threading

#pragma omp simd
#pragma omp declare simd
...

#pragma omp target
omp teams
...

#pragma omp target
omp teams
...