Using the Roofline Model automation in Intel® Advisor to boost your application performance

Zakhar Matveev, PhD, Intel SSG, November’2018
Roofline Performance
Model Automation
Drawing the Roofline
Defining the speed of light

\[
\text{GFLOPS/s} = \min \left\{ \frac{\text{Platform PEAK}}{\text{Platform BW} \times \text{AI}} \right\}
\]

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s
Old approach – pen and paper

Run STREAM

Run DGEMM

Read the source, count FP ops, loads&stores

4 loads
27 muls
1 store
51 adds

“3D stencil performance evaluation and auto-tuning on multi and many-core computers”, C.Andreolli et.al.

Cumbersome – but people still did it!
Roofline Automation in Intel® Advisor 2017

Each Roof (slope) Gives peak CPU/Memory throughput of your PLATFORM (benchmarked)

Each Dot represents loop or function in YOUR APPLICATION (profiled)

Automatic and integrated – first class citizen in Intel® Advisor
Getting Roofline data in Intel® Advisor: two-pass approach

<table>
<thead>
<tr>
<th>Step 1: Survey (-collect survey)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Provide <strong>#Seconds</strong></td>
<td><strong>1x</strong></td>
</tr>
<tr>
<td>- <em>Root access not needed</em></td>
<td></td>
</tr>
<tr>
<td>- User mode sampling, non-intrusive.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2: FLOPS (-collect tripcounts –flops)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Provide <strong>FLOP</strong>, <strong>Bytes</strong>, AVX-512 Mask</td>
<td><strong>3-5x</strong></td>
</tr>
<tr>
<td>- <em>Root access not needed</em></td>
<td></td>
</tr>
<tr>
<td>- Precise, instrumentation based, count number of instructions</td>
<td></td>
</tr>
</tbody>
</table>

**Roofline:**

<table>
<thead>
<tr>
<th>Axis X: <strong>AI = #FLOP / #Bytes</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis Y: <strong>FLOP/S = #FLOP (mask aware) / #Seconds</strong></td>
<td></td>
</tr>
</tbody>
</table>
Why Mask Utilization Important?

```
for(i = 0; i <= MAX; i++)
    if (cond(i))
        c[i] = a[i] + b[i];
```

3 elements suppressed

SIMD Utilization = 5/8

62.5%
Roofline Performance Model in Intel Advisor: HOW-TO
Methods to get roofline profile in Intel Advisor

**Roofline:**

**Command Line**
advixe-cl. Full automation, works for MPI. Loops mark-up not easy.

2 pass:
advixe-cl -collect survey
advixe-cl -collect tripcounts -flop

**GUI.**
“all in one”. No automation. Doesn’t work for multi node MPI. Easy to mark-up loops.

“Run Roofline”

Step by step:
1. Survey, 2. TripCounts/FLOPS
   (“Batch Mode” is another option)
Get roofline data using **command line**. Example:

> source advixe-vars.sh

1\(^{st}\) method (available starting 2018 Update 1 only):

> advixe-cl **-collect roofline** -project-dir ./your_project -- <your-executable-with-parameters>

2nd method (more flexible):

> advixe-cl **--collect survey** -no-auto-finalize --project-dir ./your_project -- <your-executable-with-parameters>

> advixe-cl **--collect tripcounts -flop** -no-auto-finalize --project-dir ./your_project -- <your-executable-with-parameters>

> advixe-gui ./your_project
Observe slower Survey analysis or “finalization”? (1.5x analysis slow-down or more)

Change default program tree processing mode (*especially for Fortran*)

```
advixe-cl -collect survey -stackwalk-mode=online -no-stack-stitching
```

Disable system modules and non-interesting modules processing:

```
advixe-cl -collect survey -module-filter-mode=include -module-filter=foo.so
```

*Other names and brands may be claimed as the property of others.*
Advanced roofline profiling capabilities (2nd pass)

1. Enable “Roofline with call-stacks” feature (see next slides)

   \texttt{advixe-cl --collect tripcounts -flop -stacks}

2. Select loops to profile

   \texttt{advixe-cl --collect tripcounts -flop -mark-up-list=<id1>}
   \texttt{or advixe-cl --collect tripcounts -flop -loops=scalar,loop-height=0}

3. Avoid running tripcounts (save more time)

   \texttt{advixe-cl --collect tripcounts -flop -no-trip-counts}
MPI example (slurm)

1<sup>st</sup> step:

```bash
srun -n <num-of-ranks> -c <num_of_cores_per_rank> advixe-cl -v -collect survey -project-dir=<same_dir_name> -data-limit=0 <your_executable>
```

2<sup>nd</sup> step:

```bash
srun -n <num-of-ranks> -c <num_of_cores_per_rank> advixe-cl -v -collect tripcounts -flops-and-masks -project-dir=<same_dir_name> -data-limit=0 <your_executable>
```
Get Roofline data using GUI
Configure your roofline chart

1. Check your #cores (#threads) and ranks-per-node. Avoid apples (benchmarks) vs. oranges (application profile)
   • Use “Single threaded roofs” as needed
   • Use Manual roofs adjustment if no other options due to 1 < N-threads << n_cores and Save your configuration
2) If your code is SP-only (or DP-only) remove unneeded roofs
3) Use filters as needed
Use Survey and Roofline views together
Use “Code Analytics” in conjunction with Roofline
Intel Advisor Roofline landscape

Regular Advisor Release 2018 and some 2017
Available at Workshop.

NEW: Roofline with Call Stacks

Advisor 2018 Update 1
Available at Workshop

NEW: Hierarchical ROOFLINE
CARM (L1+NTS), L2, LLC, MCDRAM, DRAM,

CACHE-AWARE ROOFLINE (CARM)

Engineering version:
Available at Workshop

“Classic Roofline”
PMU Uncore events
(DRAM-only traffic)
Roofline Performance Model: Interpretation
Questions to answer with Roofline: for your loops / functions

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

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

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
3 CARM: What is my current main limit? Optimization tactics.

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

If under or near a memory roof…
• Try a MAP analysis. Make any appropriate cache optimizations.
• If cache optimization is impossible, try reworking the algorithm to have a higher AI.

If just above the Scalar Add Peak
Check vectorization efficiency in the Survey. Follow the recommendations to improve it if it’s low.

If under the Vector Add Peak
Check the Survey to see if FMAs are used. If not, try altering your code or compiler flags to induce FMA usage.

If just above the Scalar Add Peak
Check the Survey Report to see if the loop vectorized. If not, try to get it to vectorize if possible. This may involve running Dependencies to see if it’s safe to force it.

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
What is my current main limit? Optimization tactics.

DRAM Roofline:
Is DRAM my current limit?

LLC Roofline:
Is LLC my current limit?

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
What is my current main limit? Optimization tactics.

L2 Roofline:
Is L2 my main limit?

L1 Roofline:
Is L1 my main limit?

Arithmetic Intensity (Flop:Byte)

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
What is my current main limit? Optimization tactics.

Find MIN of 5 (L1, L2, LLC, DRAM, CPU)
Roofline Performance
Model:
advanced usages
WHAT’S NEW: Roofline with Callstacks
WHAT’S NEW: Roofline with Callstacks.
In production from 2018 Update 1 release

Before 2018 Update 1: (For 2018 and 2017 Update 4) : $ export ADVIXE_EXPERIMENTAL=roofline_ex

GUI:

Command Line:

1st method:  $ advixe-cl -collect roofline -stacks -project-dir ...
2nd method:  $ advixe-cl -collect survey -project-dir ...
             $ advixe-cl -collect tripcounts -flops-and-masks -stacks -project-dir ...
Roofline with Callstacks: Motivation

• Motivation 1: **aggregated** “coarser-grain” Roofline for outer loopnests/functions
  - “Total FLOP/S” as opposed to “Self FLOP/S”
  - Create your own granularity Roofline – by “collapsing” loops on the chart

• Motivation 2: de-couple instances of the same function
  - Imagine “operator+” invoked by CPU and memory bound functions
  - Need these cases to be characterized separately + maybe aggregation up to callers
Roofline with Callstacks: loopnest example

- Computational Chemistry code ("DL_MESO"), CFD, LBE equation, Shan-Chen Lattice Boltzmann method. STFC Daresbury Laboratory (UK)

“Normal” Roofline mode (no call-stacks):

- Self (exclusive) FLOPS/S metrics-based
- Only can analyze separately:
  - Innermost loop alone (SIMD, CPU-bound)
  - Computations exclusively belonging to outer loop, but not to innermost loop (Scalar, cache-bound)
- No way to characterize function “as a whole”
- No way to characterize outer loop “as a whole”
Roofline with Callstacks: loopnest example

“Normal” Roofline mode (no call-stacks):

Roofline with Call Stacks:

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Density, Intensity, Machine balance

Arithmetic Intensity = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred}}

Arithmetic Operational Intensity = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred between DRAM (MCDRAM) and LLC}}

AI Intensity = \frac{\text{Total Intops+Flops computed}}{\text{Total Bytes transferred between CPU and “memory”}}

Future plans

WIP

Implemented in 2017 Update 1

Special Engineering (tech preview) version

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
WHAT’S NEW: “Original” Roofline, “Integrated” Roofline (tech preview, not in official product)

- **Original (DRAM <-> LLC traffic-based AI) Roofline**

- **Integrated Roofline**: all memory levels traffic breakdown. Identify bottlenecks precisely.

- Not in production at the moment

- Interested to try? drop email to vector_advisor@intel.com
Cache configuration – string description

Count omitted – same as previous by default
(for intermediate levels)
Explicit count required for KNL

4 : 8w : 32k / 4w : 256k / 16w : 6m

• Groups for each cache level (separated by ‘/’)
• Properties separated by ‘:’
• Suffix determines number meaning
  • W – associativity
  • K/M/G – size
  • No suffix – number of caches
• Some numbers may be omitted

Single string specifies entire hierarchy
• Easy to specify in command line
• Easy to pass between components (e.g. HW config is reported as context value)
Currently same string in GUI, may add easier interface in the product
Multiple memory levels – processing

Database – keep all numbers in one string

Filter determines
- Memory level (L1, L2 ... DRAM)
- Traffic type (loads, stores, everything)

GUI – use selected level/type in survey and roofline

“Hack” for command-line reports – environment variables (set before running advixe-cl -report):
- CACHE_SIM_MEMORY_LEVEL – level to report (0 – L1, 1 – L2 etc.)
- CACHE_SIM_ONLY_LOADS=1 report only load traffic
- CACHE_SIM_ONLY_STORES=1 report only load traffic (specify only 1 of these, or nothing for total traffic)
Back-up
How cache simulator works?

Address trace

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x321ea226</td>
<td>Read</td>
</tr>
<tr>
<td>0x321ea228</td>
<td>Read</td>
</tr>
<tr>
<td>0x6bc12380</td>
<td>Write</td>
</tr>
<tr>
<td>0x6bc12388</td>
<td>Read</td>
</tr>
<tr>
<td>0x6bc12390</td>
<td>Write</td>
</tr>
<tr>
<td>0x321ea200</td>
<td>Read</td>
</tr>
</tbody>
</table>

Hash function

Cache set

<table>
<thead>
<tr>
<th>Tags for N ways</th>
<th>Original address</th>
<th>Valid?</th>
<th>Dirty?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most recently used</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Least recently used</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next cache /DRAM

Required for random sampling

Cache hit and miss

- Make line most recently used
- Shift other lines in set
- Mark line as valid
- Mark line as dirty on write

Additional actions on cache miss

- Evict least recently used line
  - Write to next level if dirty
  - Read line from next level
How to make it faster?

<table>
<thead>
<tr>
<th>Cache sets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Set N</td>
<td></td>
</tr>
</tbody>
</table>

Select some sets randomly and simulate

Ignore others!

Scale data to get final result

Total misses = Simulated misses * Set count / Sampled set count

Random sampling – cache sets are similar, no need to model the whole cache
How to handle multithreading?

Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6

Cache manager

Find available cache
Block thread if all caches are busy

Correspond to core count
Work in parallel

Different sets work in parallel
Serialize on same set access

L1 → L2
L1 → L2
L1 → L2

LLC
Set 1
...
Set N

Scales well with core count!
Memory Access (Latency focused) deep dive analysis

```bash
$ export ADVIXE_EXPERIMENTAL = cachesim
```
Latency/SIMD (AoS/SoA) optimization & analysis with Advisor “Memory Access Pattern”
Platform PEAK FlopS

How many floating point operations per second

\[
\text{Gflop/s} = \min \left\{ \frac{\text{Platform PEAK}}{\text{Platform BW} \times AI} \right\}
\]

Theoretical value can be computed by specification

Example with 2 sockets Intel® Xeon® Processor E5-2697 v2

PEAK FLOP = 2 x 2.7 x 12 x 8 x 2 = 1036.8 Gflop/s

Number of sockets \quad Number of cores \quad 1 port for addition, 1 for multiplication

Core Frequency \quad Number of single precision element in a SIMD register

More realistic value can be obtained by running Linpack

≃ 930 Gflop/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2
Platform PEAK bandwidth

How many bytes can be transferred per second

\[
\text{Gflop/s} = \min \left\{ \frac{\text{Platform PEAK}}{\text{Platform BW} \cdot AI} \right\}
\]

Theoretical value can be computed by specification

Example with 2 sockets Intel® Xeon® Processor E5-2697 v2

PEAK BW = \(2 \times 1.866 \times 8 \times 4 = 119 \text{ GB/s}\)

More realistic value can be obtained by running Stream

\(\approx 100 \text{ GB/s} \) on a 2 sockets Intel® Xeon® Processor E5-2697 v2