Intel Advisor and Roofline Automation
5 Steps to Efficient Vectorization - Vector Advisor

1. Compiler diagnostics + Performance Data + SIMD efficiency information

2. Guidance: detect problem and recommend how to fix it

3. “Accurate” Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads

4. Loop-Carried Dependency Analysis

5. Memory Access Patterns Analysis

- 2 -
Vectorization Analysis Workflow

1. Run Survey

2. Check Trip Counts

(Mark-up Loops)

3. Check Dependencies

Use the same target binary within every cycle

4. Check Memory Access Patterns

Start

Take Snapshot

Edit & Compile

Deeper-dive analysis
Quickly Characterize the Vectorization Efficiency of Your Code: Advisor Summary

Use the summary view to quickly characterize your program.

Time in Scalar vs. Vector loops. SIMD Efficiency.

Focus on Hottest kernels.

### Summary of predicted parallel behavior

[Image of summary screen]

#### Program metrics
- Elapsed Time: 9.66s
- Vectorized: Yes
- Not Vectorized: No
- Number of CPU Threads: 1

<table>
<thead>
<tr>
<th>Metric</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total CPU time</td>
<td>9.27s</td>
</tr>
<tr>
<td>Time in 2 vectorized loops</td>
<td>8.11s</td>
</tr>
<tr>
<td>Time in scalar code</td>
<td>1.15s</td>
</tr>
</tbody>
</table>

#### Vectorization Gain/Efficiency
- Vectorized Loops Gain/Efficiency: 2.64x
- Program Theoretical Gain: 2.43x

#### Top time-consuming loops
- matvec Multiply.c:72
  - Self Time: 5.6256s
  - Total Time: 5.6256s

#### Refinement analysis data
- [loop in ... at _]
  - No information available
  - Strides Distribution: 67% / 33% / 0%
- [loop in ... at _]
  - No information available
  - No strides found
- [loop in ... at _]
  - RAW:1 WAW:1
  - No information available

[Images and logos]
Advisor Survey: Focus and Characterize

What prevents vectorization?

hotspots

HOW to improve performance

ISA

Efficiency

All the data you need for effective vectorization
Advisor Memory Access Pattern

Memories Access Patterns Report

- Stride: 3
- Type: Mixed strides
- Source: fPropogationSwap

1246 #endif 
1247 for (int m=1; m<=half; m++)
1248  nextr = fCppMod(i + lbv[3*m])
1249  nexty = fCppMod(j + lbv[3*m+]
1250  nextz = fCppMod(k + lbv[3*m+

- Stride: 0; 1
- Source: SWAP_OLAP

1251 #ifndef SWAP_OLAP
1252 ilnext = (nextr * Ymax + nex
1253 fSwapPair (lbv[11*lbsitelen + l*lbsy].

16%: percentage of memory instructions with unit stride or stride 0 accesses
Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration
Stride 0 = Instruction accesses the same memory from iteration to iteration

84%: percentage of memory instructions with fixed or constant non-unit stride accesses
Constant stride (stride N) = Instruction accesses memory by N elements from iteration to iteration
Example: For the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4*sizeof(double)) with each iteration

0%: percentage of memory instructions with irregular (variable or random) stride accesses
Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration
Typically observed for indirect indexed array accesses, for example, A[index[i]]
- gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture
Roofline Performance Model:
Am I bound by VPU/CPU or Memory

Roofline reflects an absolute performance bound (Gflops/s) of the system as a function of Arithmetic Intensity (flops/byte) of the application.

Arithmetic Intensity = \[
\frac{\text{Total Flops computed}}{\text{Total Bytes transferred}}
\]
Why Do We Need the Roofline Model?

• Need a sense of absolute performance when optimizing applications
  – How do I know if my performance is good?
  – Why am I not getting peak performance of the platform?

• Many potential optimization directions
  – How do I know which one to apply?
  – What is the limiting factor in my app’s performance?
  – How do I know when to stop?
Is My Application Bound by a Memory Bandwidth Peak or a Compute Peak?

Often it’s a combination of the two:
- Applications in area 1 are purely memory bandwidth bound
- Applications in area 3 are purely compute bound
- In area 2 we need more information
Place your application on the roofline, then ask yourself “Why am I Here?”

Usually, the analysis is not straightforward... You won’t be on any ceiling. Or if you are, it is a coincidence. BUT - asking the questions “why am I not on a higher ceiling?” and “what should I do to reach it?” is always productive.
Roofline Automation in Advisor 2017

Each roof gives measured peak CPU/Memory throughput on the platform.

Each dot represents a loop or a function in the profiled application.
Advisor Roofline Under the Hood

Roofline application profile:

Axis Y: \( \text{FLOP/S} = \frac{\#\text{FLOP (mask aware)}}{\#\text{Seconds}} \)

Axis X: \( \text{AI} = \frac{\#\text{FLOP}}{\#\text{Bytes}} \)

- **Seconds**
  - User-mode sampling
  - Root access not needed

- **FLOP**
  - Binary Instrumentation
  - Does not rely on CPU counters

- **AI**
  - Flop/byte

- **Bytes**
  - Binary Instrumentation
  - Counts operands size (not cachelines)

- **Performance**
  - Flops/seconds

- **Roofs**
  - Microbenchmarks
  - Actual peak for the current configuration
What to Do With This Information?

• **Compute bound applications**
  – Make sure you have good OpenMP scalability. Look to see thread activity for major OpenMP regions.
  – Make sure your code is vectorizing. Look at Cycles per Instruction (CPI) and VPU utilization.

• **Memory bandwidth bound applications**
  – Try to improve memory locality, cache reuse
  – Identify the key arrays leading to high memory bandwidth usage and make sure they are/will-be allocated in HBM on KNL
    ➔ Profit by getting ~ 5x more bandwidth GB/s.
Advisor Roofline access and how-to command line example

> module load advisor
> advixe-cl --collect survey
   --project-dir ./your_project
   -- <your-executable-with-parameters>
> advixe-cl --collect tripcounts
   -flops-and-masks --project-dir
   ./your_project
   -- <your-executable-with-parameters>
> advixe-gui ./your_project
> advixe-cl --report survey
   --project-dir <your-project-dir>
   --show-all-columns --format=csv
   --report-output <output-file.csv>

1st step
Obtain “Seconds”
1.1x overhead

2nd step
Obtain #FLOP count:
3x-5x overhead

Launch GUI

Write report into csv file
Command line MPI example (slurm)

1\textsuperscript{st} step:

\begin{verbatim}
> 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>
\end{verbatim}

2\textsuperscript{nd} step:

\begin{verbatim}
> 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>
\end{verbatim}
Further Reading and Viewing

- [https://www.youtube.com/watch?v=h2QEM1HpFgg](https://www.youtube.com/watch?v=h2QEM1HpFgg)
- [https://www.codeproject.com/Articles/1169323/Intel-Advisor-Roofline-Analysis](https://www.codeproject.com/Articles/1169323/Intel-Advisor-Roofline-Analysis)
- [https://software.intel.com/sites/default/files/managed/1e/19/roofing-a-house.pdf](https://software.intel.com/sites/default/files/managed/1e/19/roofing-a-house.pdf)
- [https://github.com/tkoskela/pyAdvisor](https://github.com/tkoskela/pyAdvisor)