Roofline performance analysis and code optimization

8th LHCb Computing Workshop
16.11.2016

Omar Awile (omar.awile@cern.ch),

Performance Models

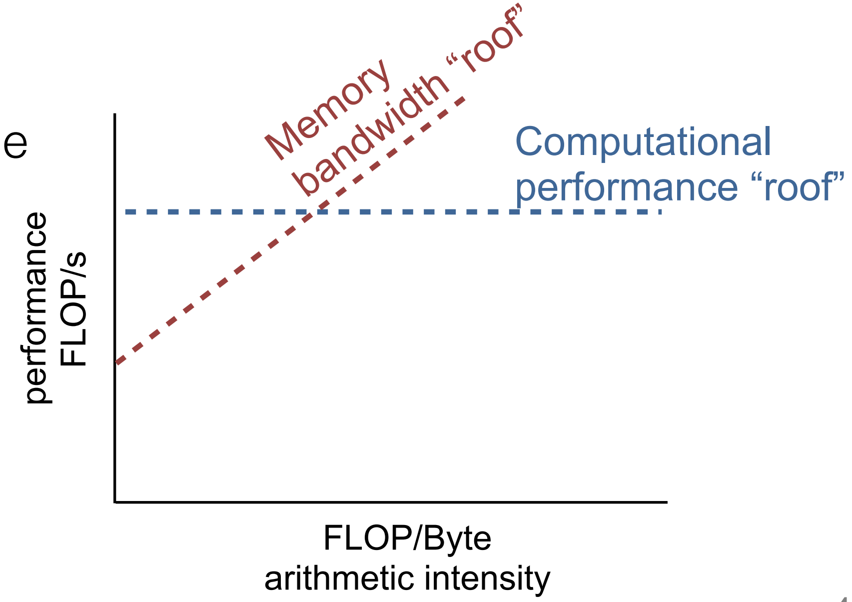
- Performance models help us better understand *why* our program is behaving in a certain way
  - With a simple model we abstract away a lot of technical details of the hardware
- We can better track performance through application development
- Guides performance optimization
  - Allows us to prioritize work
- We can make predictions for new code or new hardware

A visual performance model for floating-point applications

Disclaimer: This work was published by S. Williams et al in ACM Communications 52(4), 2009

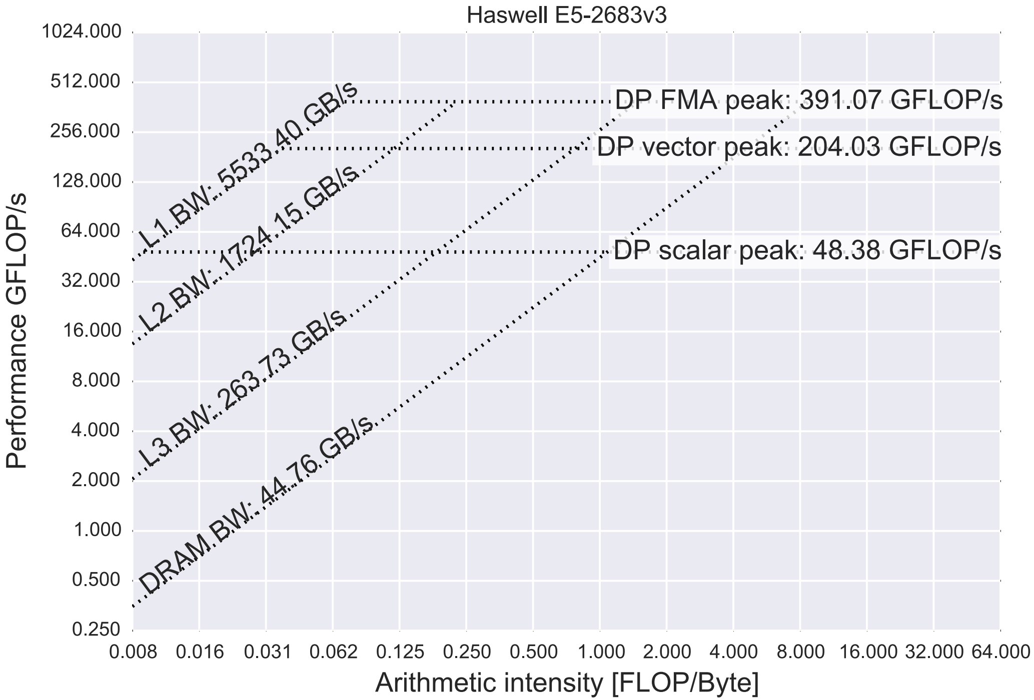
The Roofline Model

- Measure the floating point performance (FLOP/s) as a function of the arithmetic intensity (i.e. number of FLOPs per byte transferred from memory/cache).
- Performance is limited by
  - the peak performance available to the core
  - the memory bandwidth times the arithmetic intensity



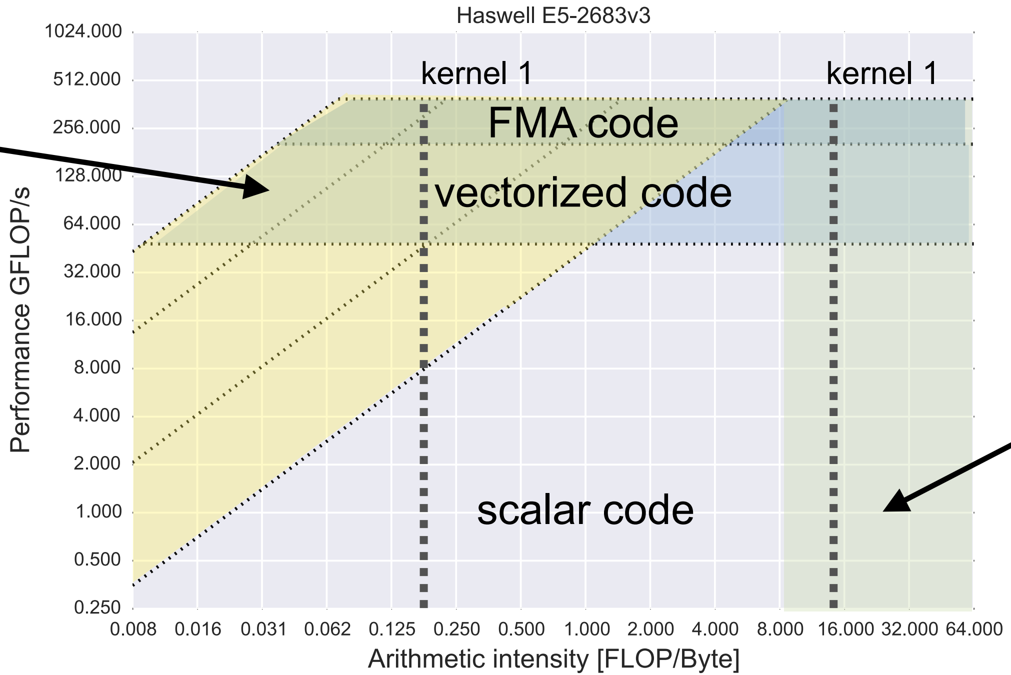
Roofline: Hardware limits

- AVX (vector instructions) takes 4 doubles: 4 x scalar perf
- FMA (fused multiply add) performs 1 multiply & 1 add at the same time: 2 x vector perf



Roofline: Software limits

Can be limited by memory/cache bandwidth
"break" through roofs by improve cache blocking & data/reuse



Only limited by comp. performance

Arithmetic Intensities

- Computational codes can be characterized by their arithmetic intensity:
  - floating point operations performed per bytes read and written

- A little example: Cholesky decomposition of 3x3 matrices

```
L[0] = sqrt(C[0]);
L_inv = 1.0 / L[0];
L[1]  = C[1]  * L_inv;
L[3]  = C[3]  * L_inv;
L[2] = sqrt((C[2] - L[1]*L[1]));
L_inv = 1.0 / L[2];
L[4]  = (C[4]  - L[3] *L[1]) * L_inv;
L[5] = sqrt((C[5] - L[3]*L[3] - L[4]*L[4]));
```

Arithmetic Intensities

- Computational codes can be characterized by their arithmetic intensity:
  - floating point operations performed per bytes read and written

- A little example: Cholesky decomposition of 3x3 matrices

```
L[0] = sqrt(C[0]);
L_inv = 1.0 / L[0];
L[1]  = C[1]  * L_inv;
L[3]  = C[3]  * L_inv;
L[2] = sqrt((C[2] - L[1]*L[1]));
L_inv = 1.0 / L[2];
L[4]  = (C[4]  - L[3] *L[1]) * L_inv;
L[5] = sqrt((C[5] - L[3]*L[3] - L[4]*L[4]));
```

16 FLOPs
48 Bytes read
48 Bytes written

0.16 FLOPs/Byte

Arithmetic Intensities

- Computational codes can be characterized by their arithmetic intensity:
  - floating point operations performed per bytes read and written

- A little example: Cholesky decomposition of 3x3 matrices

Depends on context: has C been used before? will L be used afterwards?

```
L[0] = sqrt(C[0]);
L_inv = 1.0 / L[0];
L[1]  = C[1]  * L_inv;
L[3]  = C[3]  * L_inv;
L[2] = sqrt((C[2] - L[1]*L[1]));
L_inv = 1.0 / L[2];
L[4]  = (C[4]  - L[3] *L[1]) * L_inv;
L[5] = sqrt((C[5] - L[3]*L[3] - L[4]*L[4]));
```

16 FLOPs
48 Bytes read
48 Bytes written

0.16 FLOPs/Byte

Arithmetic Intensities

- Computational codes can be characterized by their arithmetic intensity:
  - floating point operations performed per bytes read and written

- A little example: Cholesky decomposition of 3x3 matrices

```
L[0] = sqrt(C[0]);
L_inv = 1.0 / L[0];
L[1]  = C[1]  * L_inv;
L[3]  = C[3]  * L_inv;
L[2] = sqrt((C[2] - L[1]*L[1]));
L_inv = 1.0 / L[2];
L[4]  = (C[4]  - L[3] *L[1]) * L_inv;
L[5] = sqrt((C[5] - L[3]*L[3] - L[4]*L[4]));
```

16 FLOPs
48 Bytes read
48 Bytes written

0.16 FLOPs/Byte

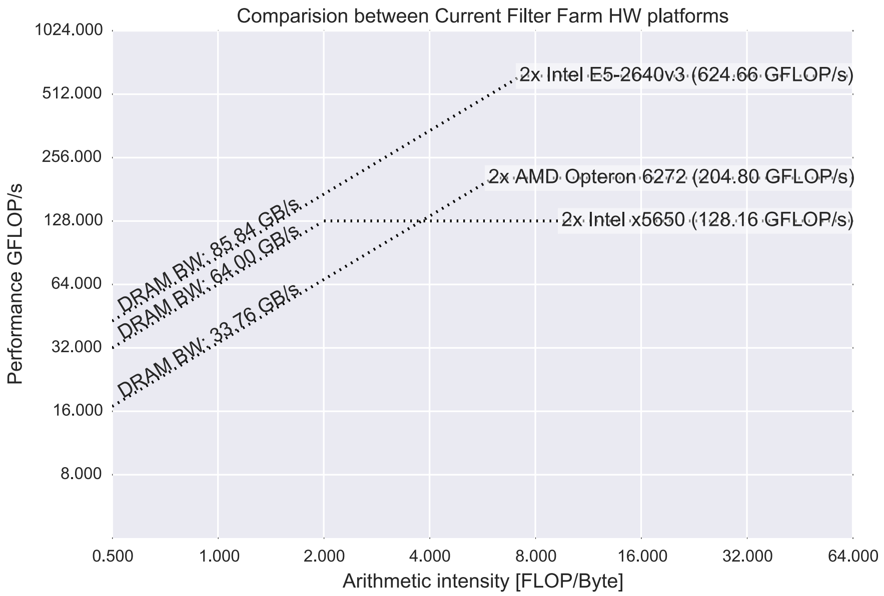
By the way… How many FMAs can we have here?
How does this change the arithmetic intensity?

Some hardware rooflines

Hardware limits

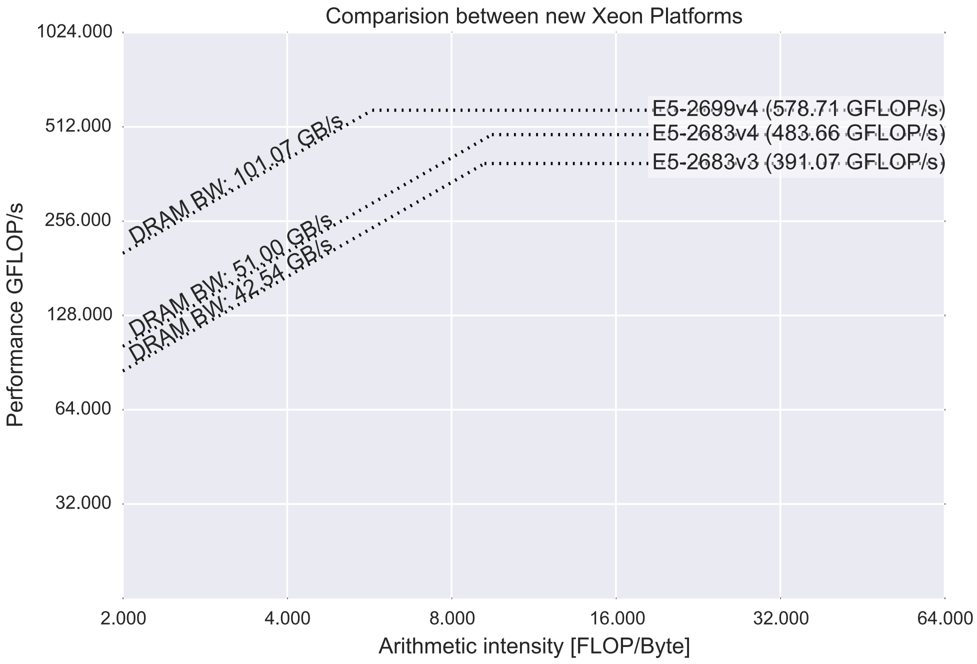
- By analyzing the specific rooflines for different hardware architectures we can see what is the maximum performance we can achieve with a particular code

- Intel x5650 has lowest peak perf **but** it is very well balanced.
    - peak perf can be achieved at arithmetic intensity < 2.0!



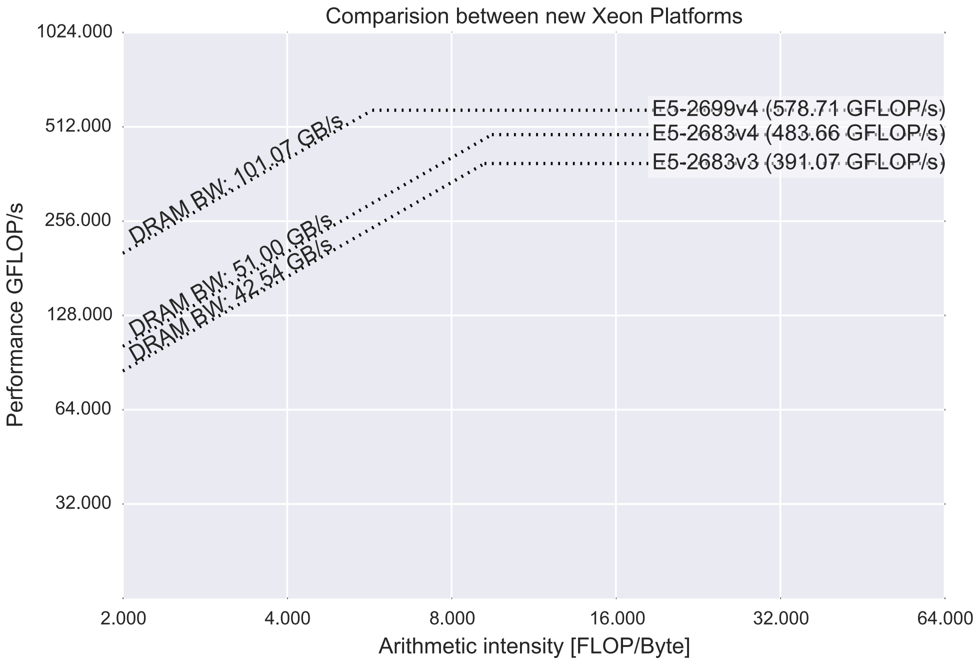
Hardware limits - looking forward

- E5-2699v4 shows impressive performance.
    - Great BW means lower “sweat spot” (5 FLOPs/Byte)



Hardware limits - looking forward

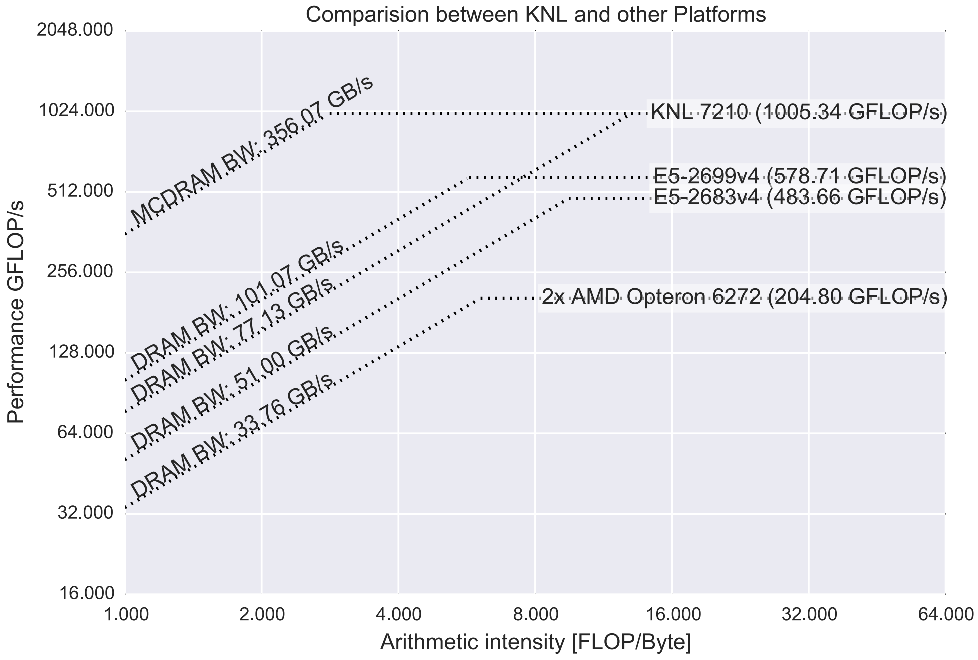
- E5-2699v4 shows impressive performance.
    - Great BW means lower “sweet spot” (5 FLOPs/Byte)



What about the KNL?

Hardware limits - KNL

- Careful: We are comparing single CPUs here!
    - A dual-socket E5-2699v4 will still beat the KNL (but $ x2 !)
- We have to learn how to properly use MCDRAM



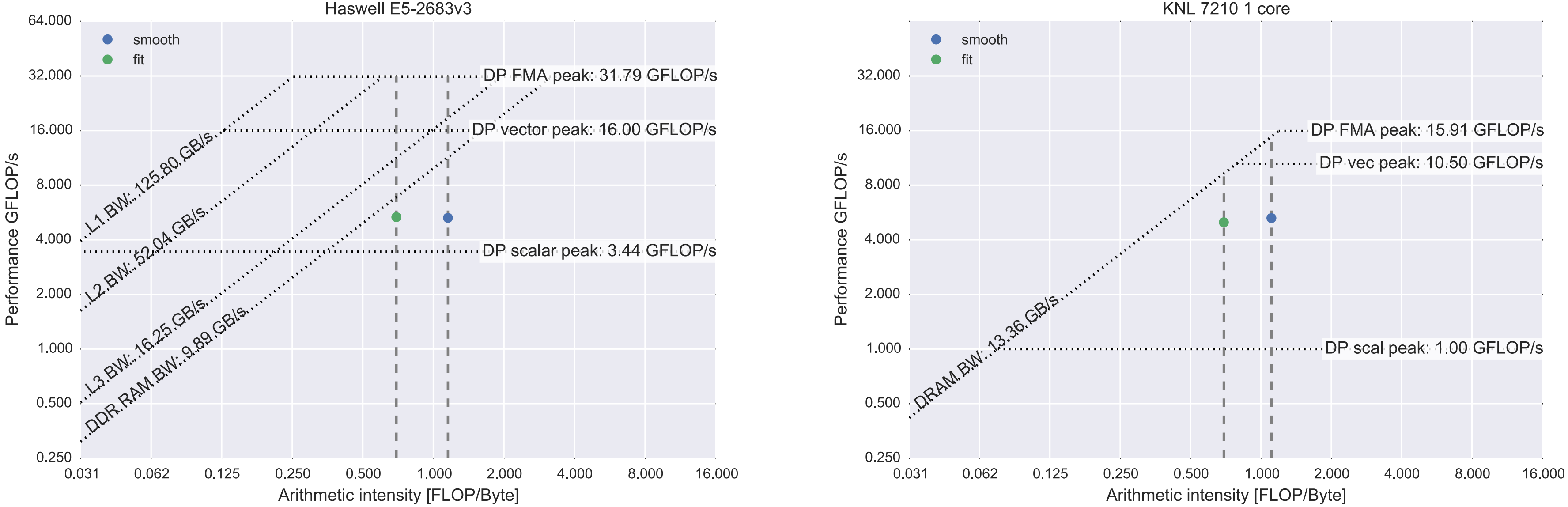
How to get your own Rooflines

The manual way

1. Get the roofs for the hardware architecture you are running on
    - Using theoretical limits from specification
    - Using micro benchmarks: https://bitbucket.org/berkeleylab/cs-roofline-toolkit
2. Get the number of FLOPs the code is incurring
    - By analyzing the code
    - By using Intel SDE: https://software.intel.com/en-us/articles/intel-software-development-emulator
3. Get the number of bytes read/written
    - By analyzing the code
    - By using vtune and hardware counters for memory read/write events

Kalman Filter

- These are single-threaded benchmarks.
- Especially *smooth* could probably be further improved
- KNL code uses AVX512 & FMA

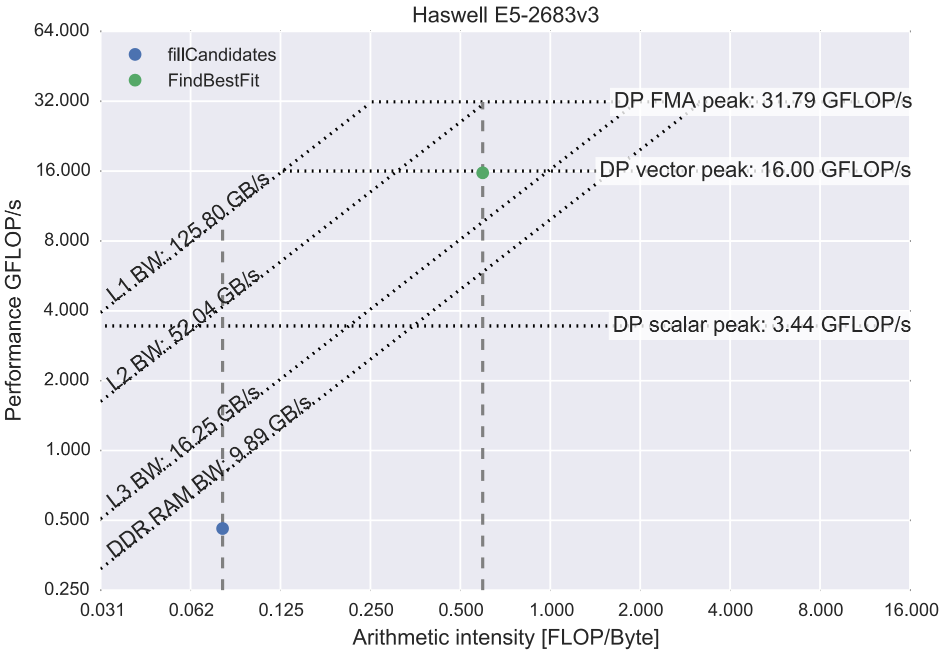


The automatic way

- Intel Analyzer 2018 will have a tool for roofline analysis.
  - Currently still in alpha/beta stage, but available at CERN openlab.
  - Contact me if you are interested

Velopixel track reconstruction

- Only one kernel of this algorithm has an arithmetic intensity that can take advantage to typical optimizations (here we show top 2)
- Overall arithmetic intensity very low —> A completely different approach might be worth it



In Summary

- The roofline model can be useful in three ways:
    1. It helps tracking hardware performance and allows easily comparing different platforms
    2. It can be used as a tool during code development or optimization to see how close (or rather how far) are we are to the optimum
    3. It gives guidance as to which is the next optimization to attack
- Caveats:
    - Bytes read&written is difficult to assess, depends on operations around kernel
    - The model works well for small computational kernels —> There is no point in making a Gaudi roofline!
    - Integer operations and memory latency sensitive operations are not exposed in this model

Thank you!

Who are we:

CERN openlab High Throughput Computing Collaboration
Olof Bärring, Niko Neufeld
Luca Atzori, Omar Awile, Paolo Durante, Christian Färber, Placido Fernandez, Karel Hà, Jon Machen (Intel), Rainer Schwemmer, Sébastien Valat, Balázs Vőneki