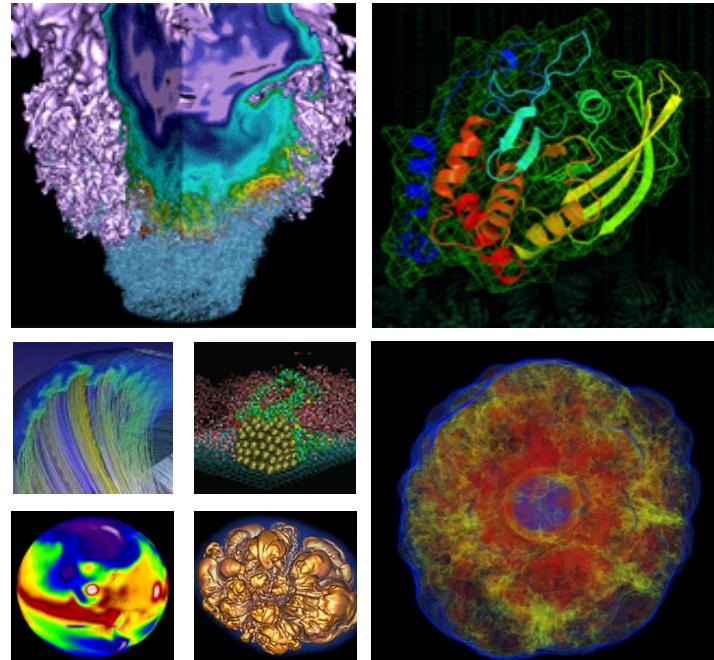


Quantitatively Assessing Performance Portability with Roofline

IDEAS Jan 23 2019



Jack Deslippe, John Pennycook, Charlene Yang
jrdeslippe@lbl.gov, john.pennycook@intel.com, cjyang@lbl.gov

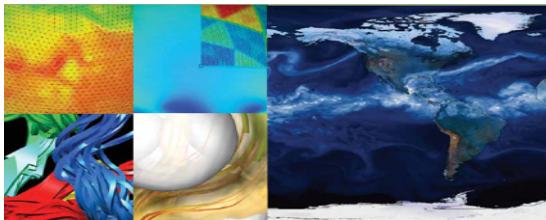
NERSC: Mission HPC for DOE Office of Science



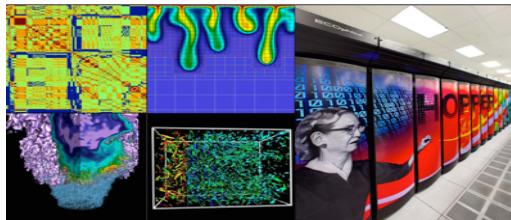
U.S. DEPARTMENT OF
ENERGY

Office of
Science

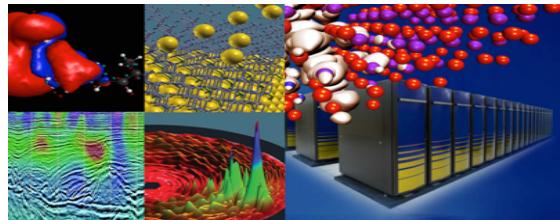
Largest funder of physical
science research in U.S.



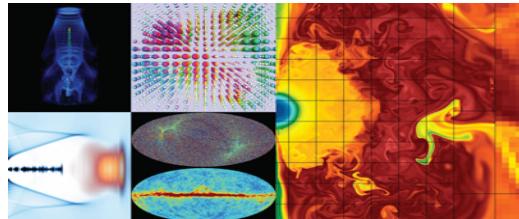
Bio Energy, Environment



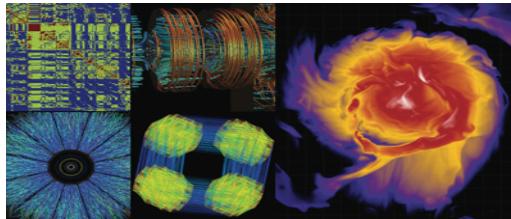
Computing



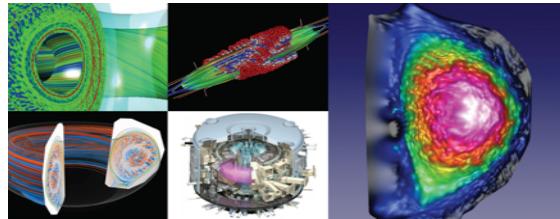
Materials, Chemistry, Geophysics



Particle Physics, Astrophysics



Nuclear Physics



Fusion Energy, Plasma Physics

7,000 users, 750 projects, 700 codes, 48 states, 40 countries, universities & national labs



How to Enable NERSC's diverse community
of 7,000 users, 750 projects, and 700 codes
to run on advanced architectures like Cori
(KNL), Perlmutter (GPUs) and Beyond

What was different about Cori?



Edison (“Ivy Bridge”):

- 24 physical cores per node
- 2.4 - 3.2 GHz
- 8 double precision ops/cycle
- 64 GB of DDR3 memory (2.5 GB per physical core)
- ~100 GB/s Memory Bandwidth
- L1/L2/L3 Caches

Cori (“Knights Landing”):

- 68 physical cores per node
- 1.4 - 1.6 GHz
- 32 double precision ops/cycle
- 16 GB of fast memory
96GB of DDR4 memory
- Fast memory has 400 - 500 GB/s
- L1/L2 Cache, No L3 Cache

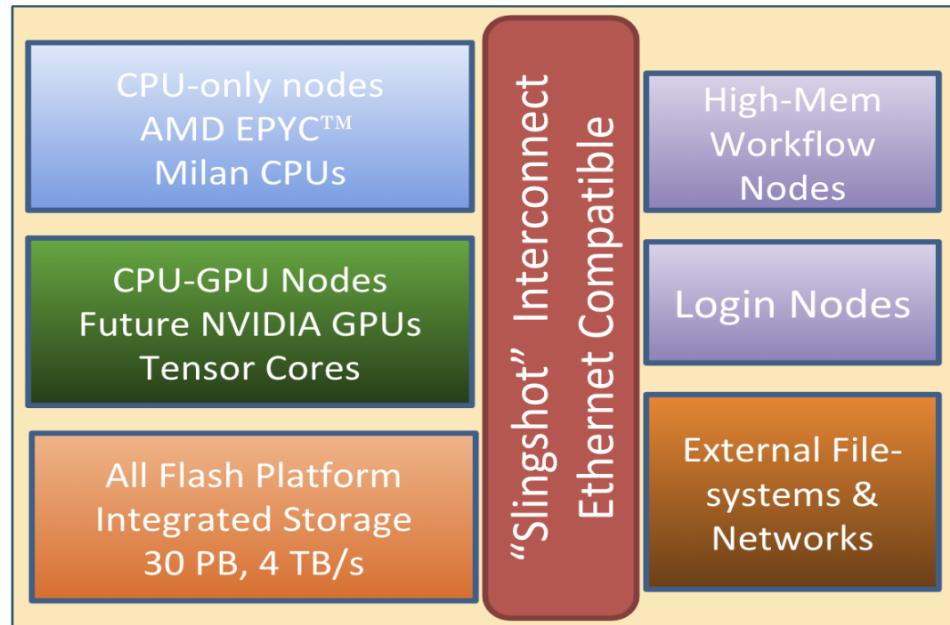


Perlmutter: A System Optimized for Science



GPU-accelerated and CPU-only nodes meet the needs of large scale simulation and data analysis from experimental facilities

NERSC's Goal is to provide a transition path from Cori to Perlmutter to NERSC-10



Science teams need a simple way to wrap their heads around performance and (performance portability) when main focus is scientific productivity:

1. 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 advertised
- How Do I know when to stop?

2. Many potential optimization directions:

- How do I know which to apply?
- What is the limiting factor in my app's performance?
- Again, how do I know when to stop?

3. How improve performance portably?

- Users are scientists. Have accounts on many system. Don't want yearly rewrite

Framing the Optimization Conversation



Energy-Efficient Processors Have Multiple Hardware Features to Optimize Against:

- Many (Heterogeneous) Cores
- Big WARPS/Vectors
- New ISA
- Multiple Memory Tiers

It is easy for users to get bogged down in the weeds:

- How do you know what KNL hardware feature to target?
- How do you know how your code performs in an absolute sense and when to stop?

Optimizing Code for Cori/Perlmutter is Like:

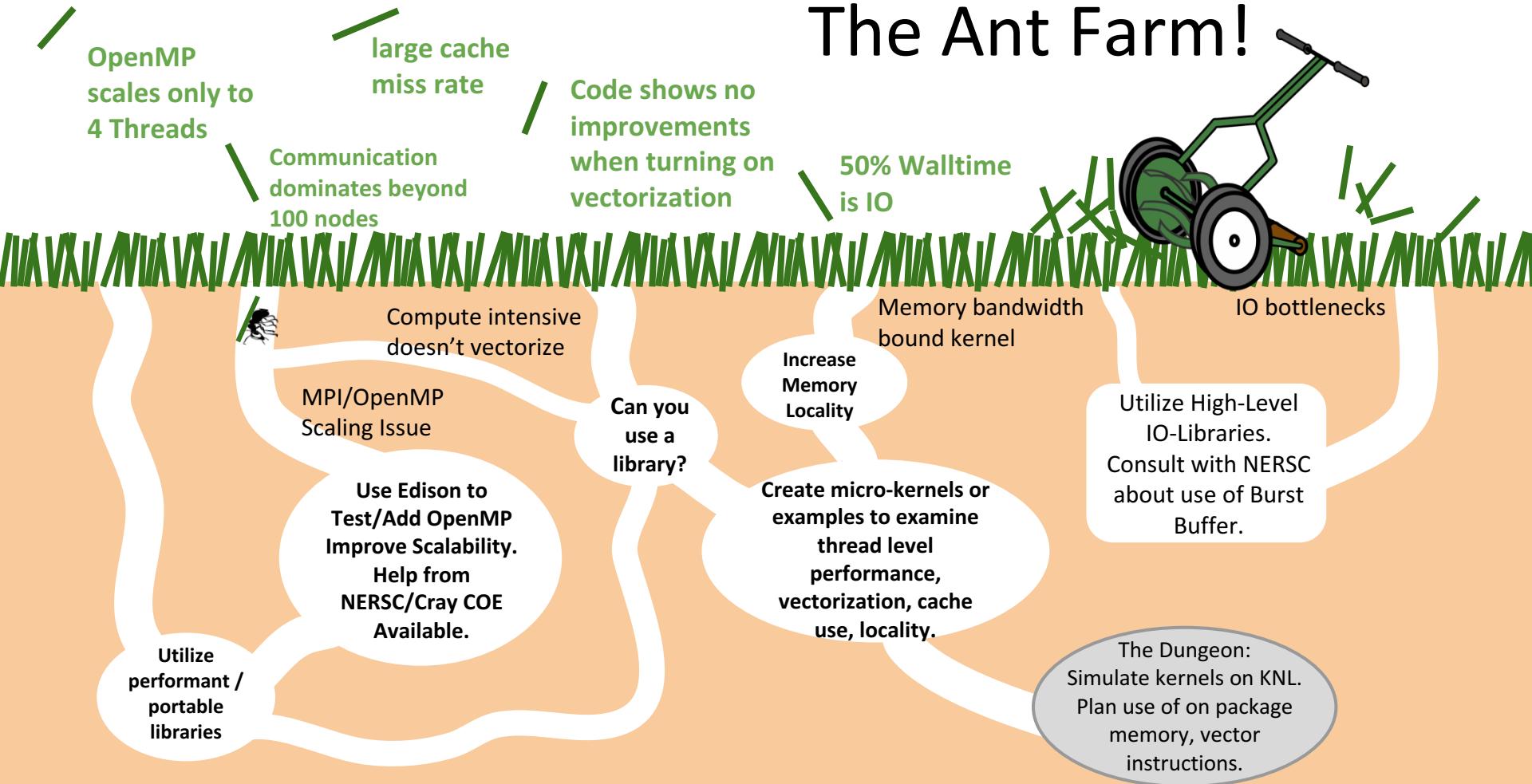
A Staircase ?

B Labyrinth ?

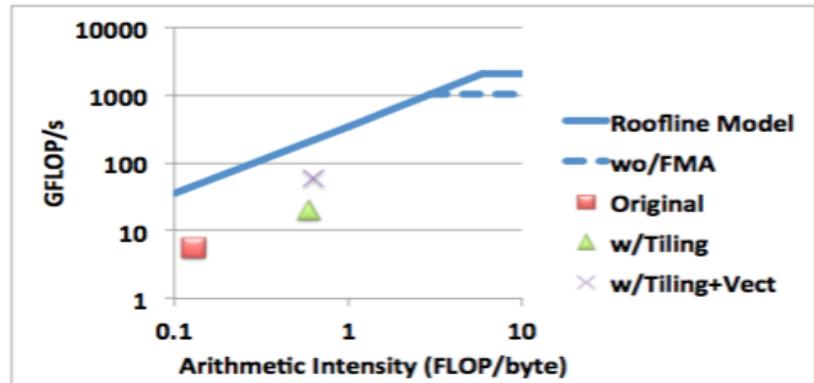
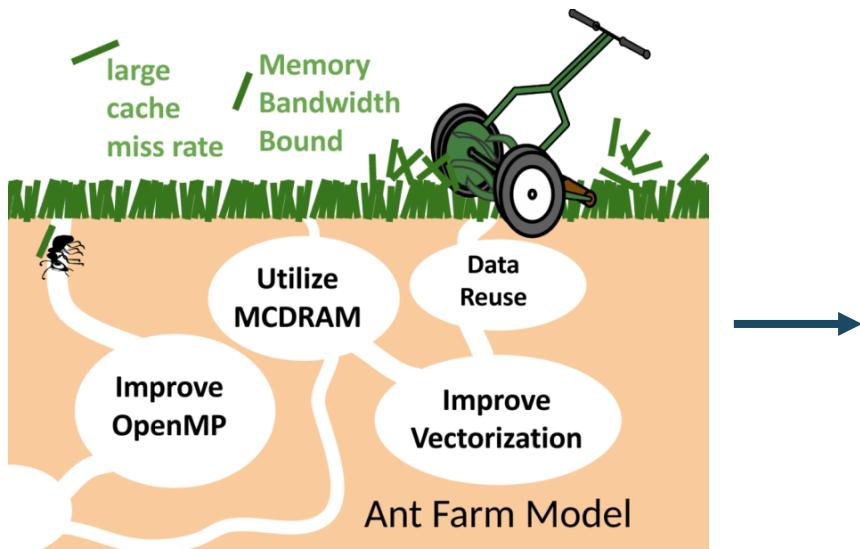
C Space Elevator?



The Ant Farm!



Evolution of The Story



(b) KNL Roofline

Framing Performance Portability



Everyone knows “**roughly**” what performance portability is. But, in order to make progress, **it pays to be precise and quantifiable**

DOE SC Facility Definition

An application is performance portable if it achieves a consistent ratio of the actual time to solution to either the best-known or the theoretical best time to solution on each platform with minimal platform specific code required.

Measuring Performance Portability



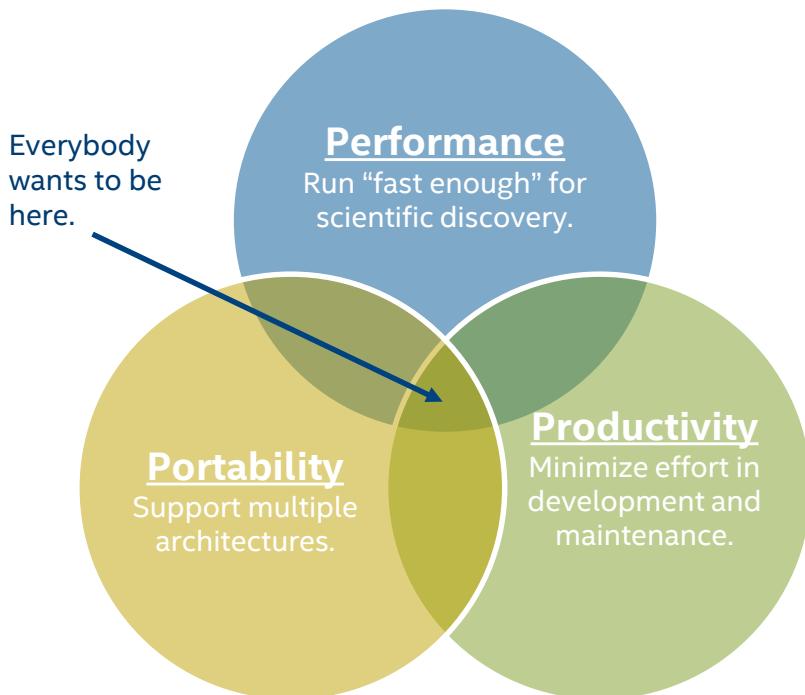
Bad Ways

1. Compare time-to-solution on one system vs another.
2. Compare ratio of actual app performance to peak system performance

Good Ways

1. Compare time-to-solution on each system against a well-known optimal implementation
2. Compare performance on each system against a relevant roofline-model ceiling on each system (We've included instructions for KNL and GPU)

The Three “P”s: An HPC Perspective



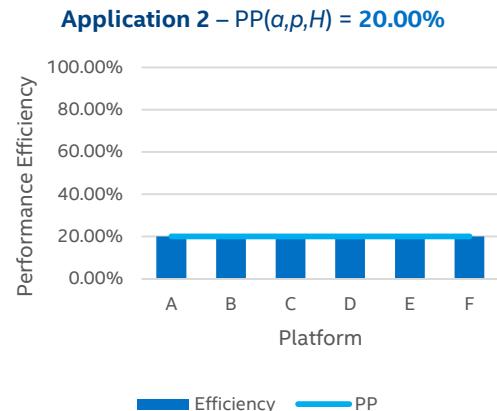
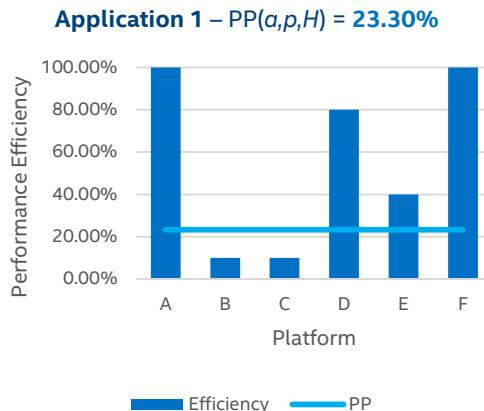
- **Performance / Productivity**
Enable domain scientists to write high-performance codes with minimal tuning
- **Performance / Portability**
Enable applications to run at different facilities, on different machine types
- **Portability / Productivity**
Enable developers to program in one shared language/programming model

Do We Need Quantitative Metrics?

- We can probably all agree that an ideal application:
 - Performs as well as possible
 - Runs on all platforms (even those that don't exist yet!)
 - Requires no effort to write or maintain
- How close are we? How do efforts compare? How close is close enough?
It's very hard to answer questions like these without a standard methodology.
- We're not looking to force our terminology and metrics on anybody, but we've found them useful in guiding discussion and refining development goals.

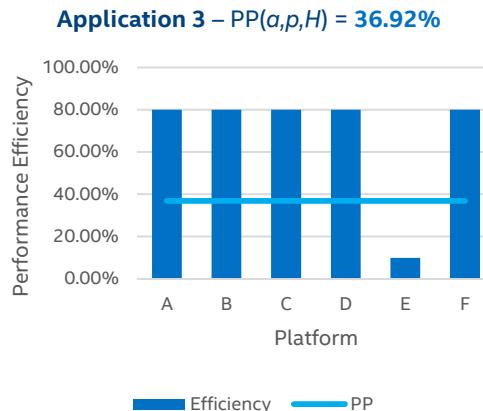
A Quantitative Metric for Performance Portability

$$\Phi(a, p, H) = \begin{cases} \frac{|H|}{\sum_{i \in H} \frac{1}{e_i(a, p)}} & \text{if } i \text{ is supported } \forall i \in H \\ 0 & \text{otherwise} \end{cases}$$



$e_i(a, p)$ = efficiency of application a for input problem p .

“The **harmonic mean** of an application’s performance efficiency on a set of platforms for a given problem.”



What are “Applications”? What are “Problems”?

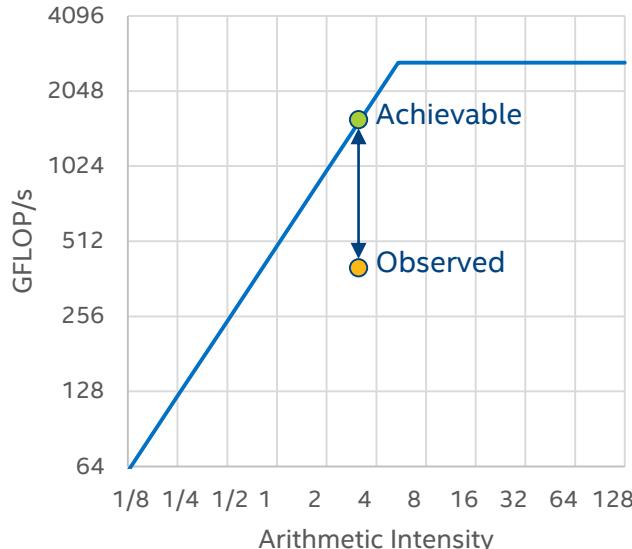
- An application is a suite of software that accepts a given problem as input and produces an output that can be validated to be correct
 - Two separate codes with a wrapper shell script can be considered one application
 - Different input problems can have very different PP scores for the same application
- Varying a and p allows us to answer questions like:
 - “What application has the highest PP for problem p ?”
 - “Is the PP of application a consistent across different input problems/sizes?”
 - “What is the impact of some transformation $a \rightarrow a'$ on PP?”

What are “Platforms”? What is H ?

- A platform is a particular execution environment
 - Includes hardware, operating system, compiler, runtime tools, etc
 - No restriction on what platforms can be used to compute PP
- Varying H allows us to answer questions like:
 - “What is the PP of application a between the platforms available in this cluster?”
 - “Is application a more PP across platforms of type X or type Y ? ”
 - “Are applications developed in f really PP between platforms of type X and type Y ? ”
 - “How PP is application a between different generations of architecture Z ? ”

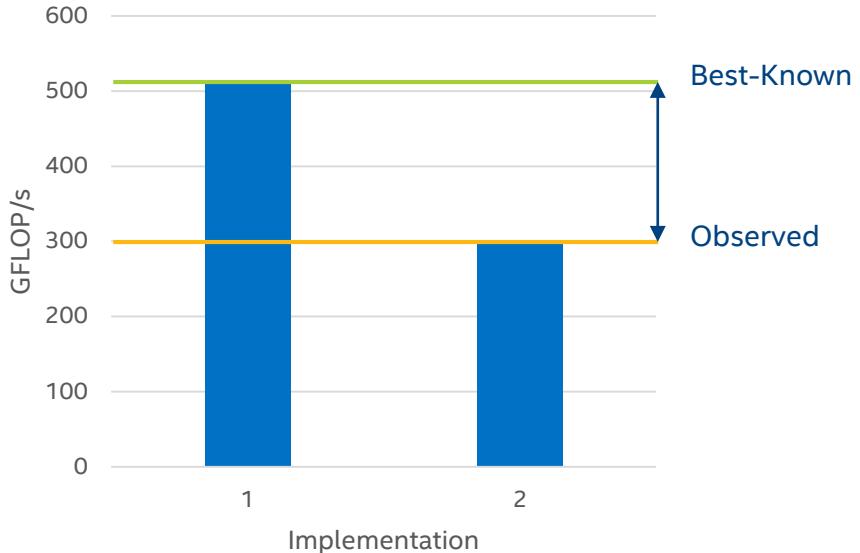
What is “Performance Efficiency”?

Architectural Efficiency



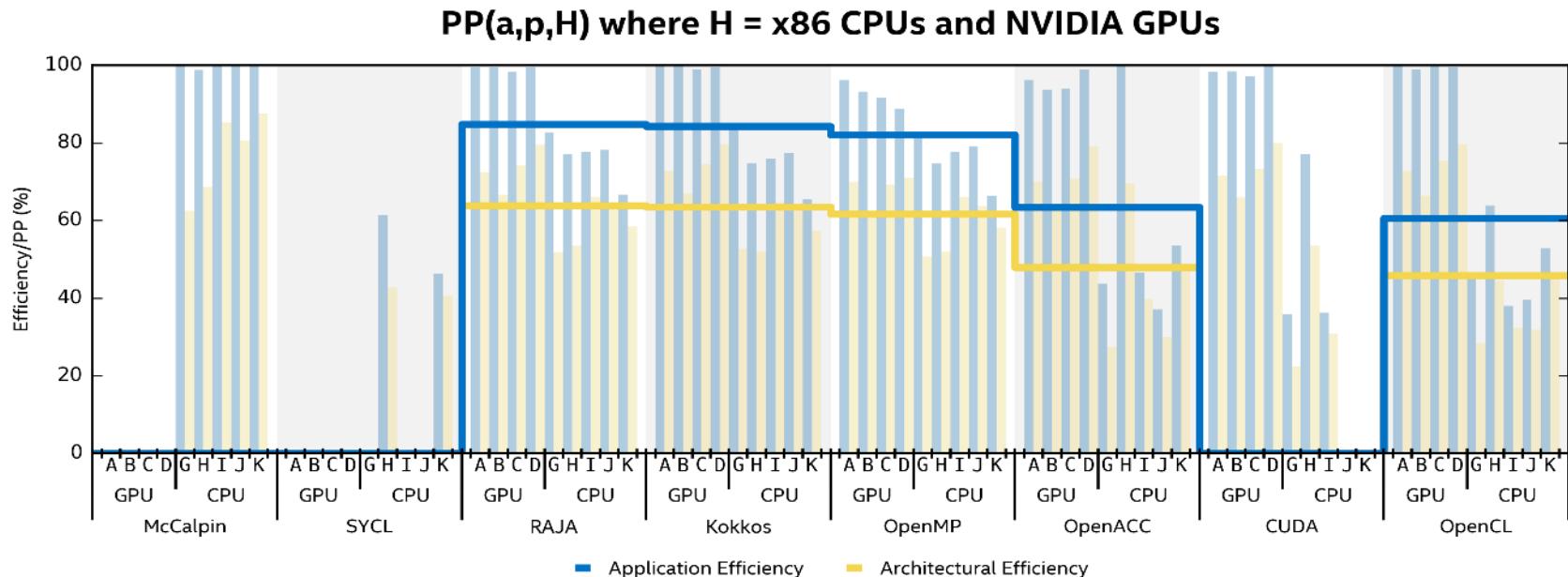
Represents how well an application utilizes each platform's resources

Application Efficiency



Represents whether an application uses appropriate algorithms on each platform

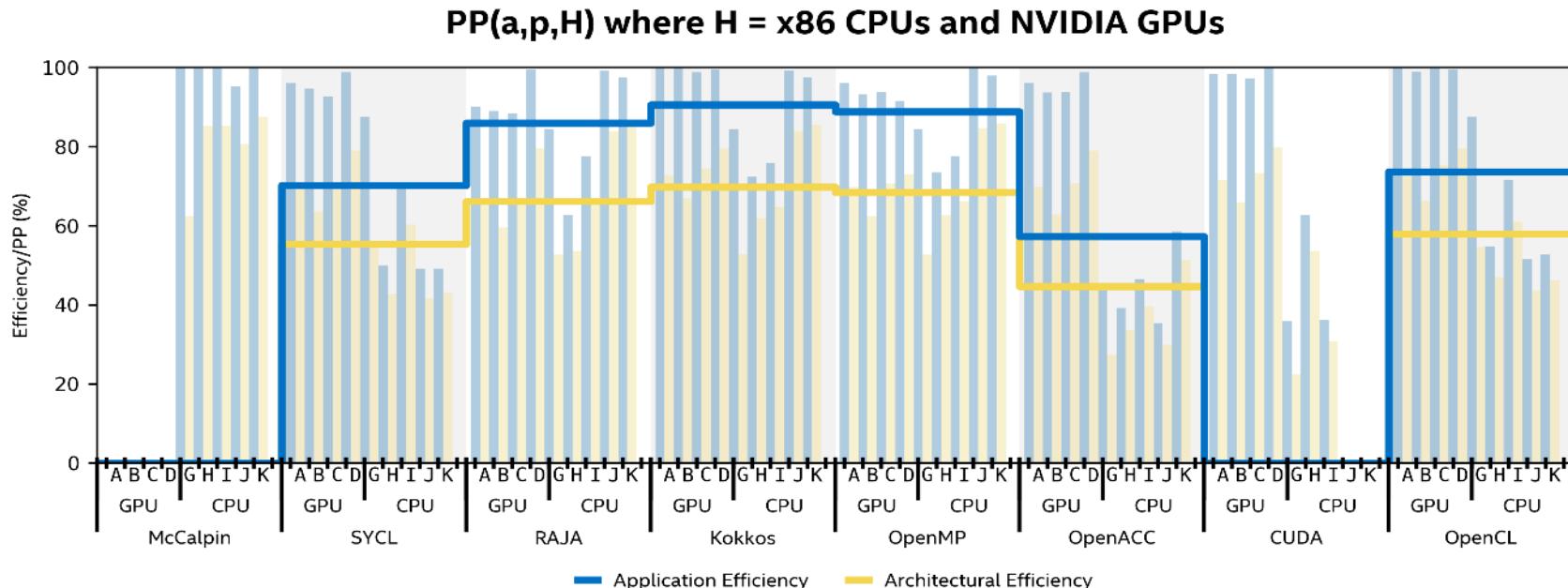
Case Study: BabelStream (2016)



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of November 2016 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. Intel does not control or audit third-party benchmark data or the other papers referenced in this document. You should visit the referenced documents and confirm whether referenced data are accurate. For configuration information, see Slide 10. Source: Intel Parallel Computing Center at University of Bristol

Case Study: BabelStream (2017)



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of August 2017 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. Intel does not control or audit third-party benchmark data or the other papers referenced in this document. You should visit the referenced documents and confirm whether referenced data are accurate. For configuration information, see Slide 10. Source: Intel Parallel Computing Center at University of Bristol

5 “Ways to Fool the Masses When...” Reporting PP

1. Use $PP(a,p,H)$ to compare platforms or frameworks.
 $PP(a,p,H)$ is a measurable property of an application.
2. Compute $PP(a,p,H)$ using a different H for each application.
Comparing portability across different sets of platforms is not useful.
3. Make a less-than-best effort to compute performance efficiency.
Picking a bad performance baseline skews the metric.
4. Bury the problem definition, or compare different problems.
Problem parameters (e.g. size, desired accuracy) impact performance.
5. Use a loose definition of “platform” to inflate claims.
“PP across 100 platforms” is not impressive if OS versions are “platforms”.

5 “Ways to Fool the Masses When...” Reporting PP

1. Use $PP(a,p,H)$ to compare platforms or frameworks.
 $PP(a,p,H)$ is a measurable property of an application.
2. Compute $PP(a,p,H)$ using a different H for each application.
Comparing portability across different sets of platforms is not useful.
3. Make a less-than-best effort to compute performance efficiency.
Picking a bad performance baseline skews the metric.
4. Bury the problem definition, or compare different problems.
Problem parameters (e.g. size, desired accuracy) impact performance.
5. Use a loose definition of “platform” to inflate claims.
“PP across 100 platforms” is not impressive if OS versions are “platforms”.

Legal Notices and Disclaimers

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of November 2016 and August 2017 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. Configurations: Testing was performed by the Intel Parallel Computing Center at the University of Bristol:

- T. Deakin, J. Price, M. Martineau and S. McIntosh-Smith, "GPU-STREAM v2.0: Benchmarking the Achievable Memory Bandwidth of Many-Core Processors Across Diverse Parallel Programming Models", in *Proceedings of the Workshop on Performance Portable Programming Models for Accelerators*, 2016 (Configuration: see Section 4)
- T. Deakin, J. Price, M. Martineau and S. McIntosh-Smith, "Evaluating Attainable Memory Bandwidth of Parallel Programming Models via BabelStream", *International Journal of Computational Science and Engineering*, 2017
- K. Raman, T. Deakin, J. Price and S. McIntosh-Smith, "Improving Achieved Memory Bandwidth from C++ Codes on Intel® Xeon Phi™ Processor (Knights Landing)", in *Proceedings of the IXPUG Annual Spring Conference*, 2017 (Configuration: see Slide 4)

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.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

Intel, the Intel logo, Look Inside, Xeon, and Intel Xeon Phi are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

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



Roofline Facilitates PP Analysis



Focus: Architectural Efficiency $e_i(a, p)$ and Roofline

F_i Peak GFLOP/s, B_i Peak Bandwidth, $I_i(a, p)$ Arithmetic Intensity (AI)

$$\Phi(a, p, H) = \begin{cases} \frac{|H|}{\sum_{i \in H} \frac{1}{e_i(a, p)}} & \text{if } i \text{ is supported, } \forall i \in H \\ 0 & \text{otherwise} \end{cases}$$

$$e_i(a, p) = \frac{P_i(a, p)}{\min(F_i, B_i \times I_i(a, p))}$$

Three Messages:

- Use empirical Roofline ceilings
- Appropriately account for divides in FLOPs
- Roofline can capture nuances of performance analysis such as changes in AI, instruction mix, instruction issue/exec bandwidth, memory access pattern, etc



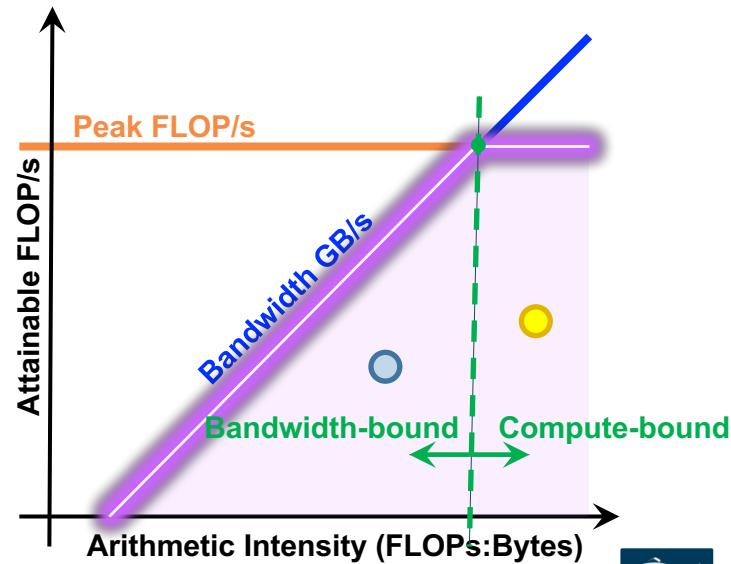
A Primer on Roofline



- An application's maximum attainable performance on a machine is:

$$P_{attainable} = \min(F, B \times I)$$

- F : peak FLOP/s
- B : peak bandwidth
- I : arithmetic intensity (AI) = FLOPs / Bytes
- Hierarchical Roofline
 - DRAM/HBM/L2/L1 bandwidths
 - vector/scalar/etc compute peaks
- Log-Log scale, easy to extrapolate



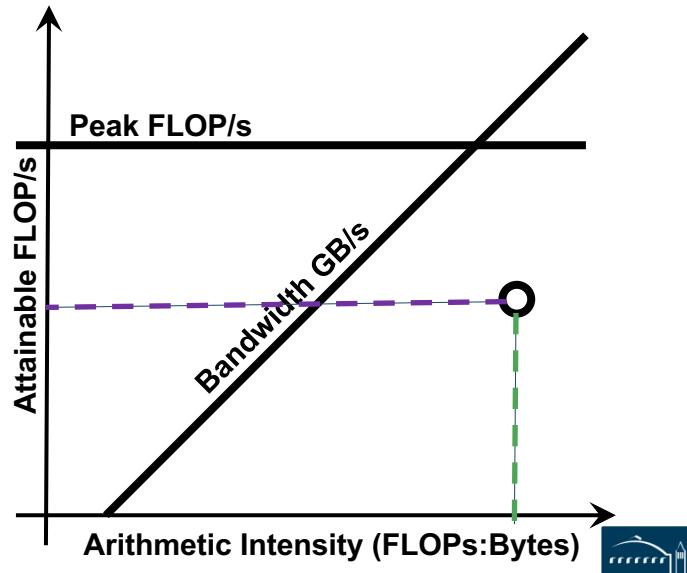
How to Collect Roofline Data



- Methodology to build a Roofline for an application
 - Measure empirical compute and bandwidth ceilings:
 - Empirical Roofline Toolkit (ERT)
 - <https://bitbucket.org/berkeleylab/cs-roofline-toolkit/>
 - Measure application performance:
 - SDE and LIKWID on KNL
 - NVPROF on V100

$$\text{Arithmetic Intensity} = \frac{\text{SDE or } nvprof \text{ FLOPs}}{\text{LIKWID or } nvprof \text{ Data Movement}} \quad (\text{X coordinate: FLOPs/Byte})$$

$$\text{Application Performance} = \frac{\text{SDE or } nvprof \text{ FLOPs}}{\text{Runtime}} \quad (\text{Y coordinate: GFLOP/s})$$



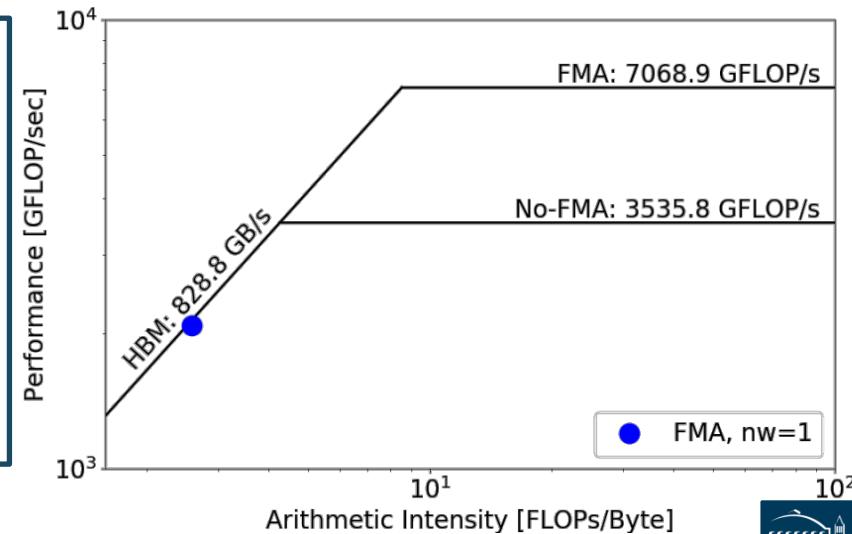
How to Plot Roofline Data



- Use Python, gnuplot, or other tools to plot Roofline
 - Example: `plot_roofline.py data.txt`
 - <https://github.com/cyanguwa/nersc-roofline/tree/master/Plotting>

```
data.txt
# all data is space delimited
memroofs 828.758
mem_roof_names 'HBM'
comproofs 7068.86 3535.79
comp_roof_names 'FMA' 'No-FMA'

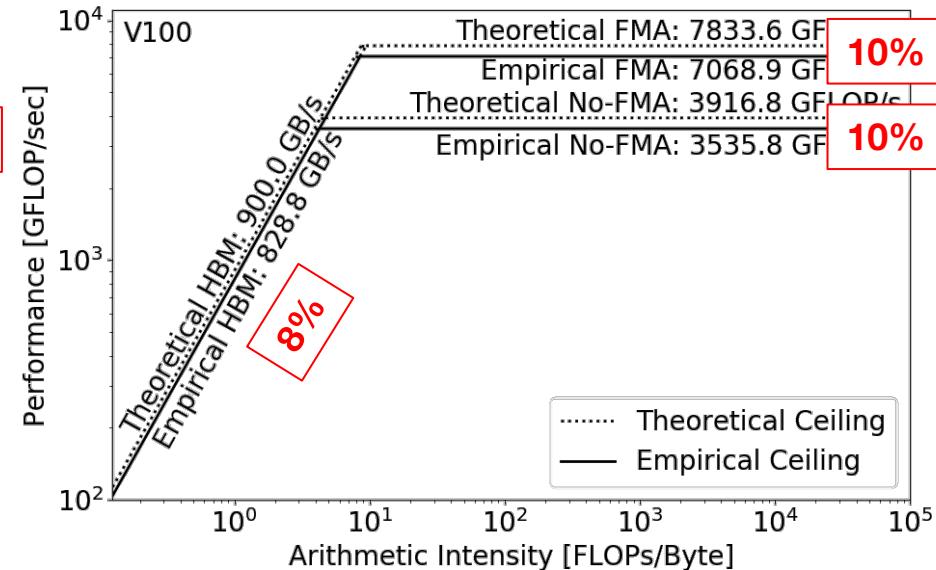
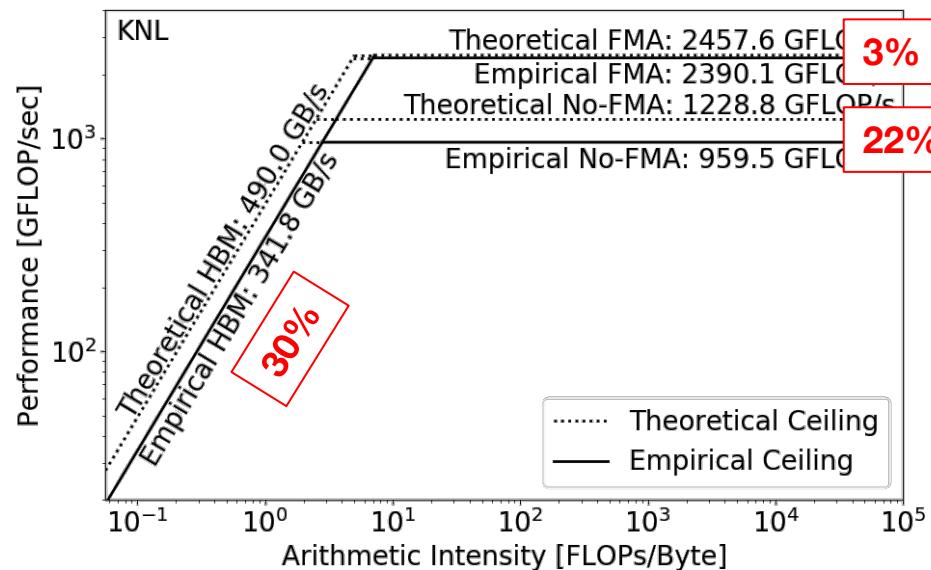
# omit the following if only plotting roofs
AI 2.584785579
GFLOPs 2085.756683
labels 'FMA, nw=1'
```



Message 1: Empirical vs. Theoretical



- Discrepancy between empirically measured peaks and arch specs
- You may be closer to the ‘realistic’ performance bounds than you think you are!



Message 2: Account for Divides



- Operations such as div, exp, log and trigonometric functions usually take more than one instructions
- Gap between canonical and empirical FLOPs:
 - Empirical: each divide counts as multiple FLOPs
 - Canonical: each counts as 1 FLOP

Message 2: Account for Divides



- Operations such as div, exp, log and trigonometric functions usually take more than one instructions
- GPP (General Plasmon Pole) kernel from BerkeleyGW (Material Science)
 - Tensor-contraction, abundant parallelism, large reductions
 - Low FMA counts, divides, complex double data type

```
do band = 1, nbands      #threadblocks
  do igr = 1, ngpown
    do ig = 1, ncouls      #threads
      do iw = 1, nw        #unrolled
        compute; reductions
```

Message 2: Account for Divides



Highly parameterizable:

- Varying **nw** from 1 to 6 to increase arithmetic intensity
 - increasing FLOPs, same HBM data movement

```
do band = 1, nbands      #threadblocks
  do  igr = 1, ngpown
    do ig = 1, ncouls      #threads
      do iw = 1, nw        #unrolled
        compute; reductions
```

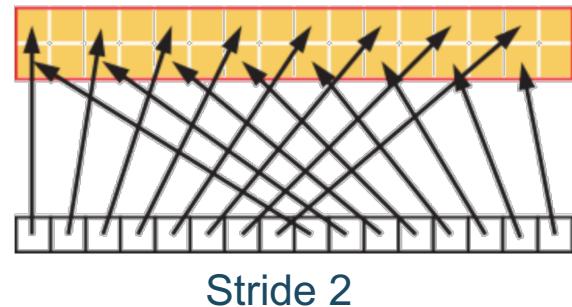
Message 2: Account for Divides



Highly parameterizable:

- Varying `nw` from 1 to 6 to increase arithmetic intensity
 - increasing FLOPs, same HBM data movement
- Striding `ig` loop to analyze impact of strided memory access
 - Split `ig` loop to two loops and place the 'blocking' loop outside

```
do band = 1, nbands          #threadblocks
    do igrp = 1, ngpown
        do igs = 0, stride - 1 #threads
            do ig = 1, ncouls/stride
                do iw = 1, nw      #unrolled
                    compute; reductions
```



Message 2: Account for Divides



- Gap between canonical and empirical FLOPs:
 - Empirical: each divide counts as multiple FLOPs
 - Canonical: each counts as 1 FLOP

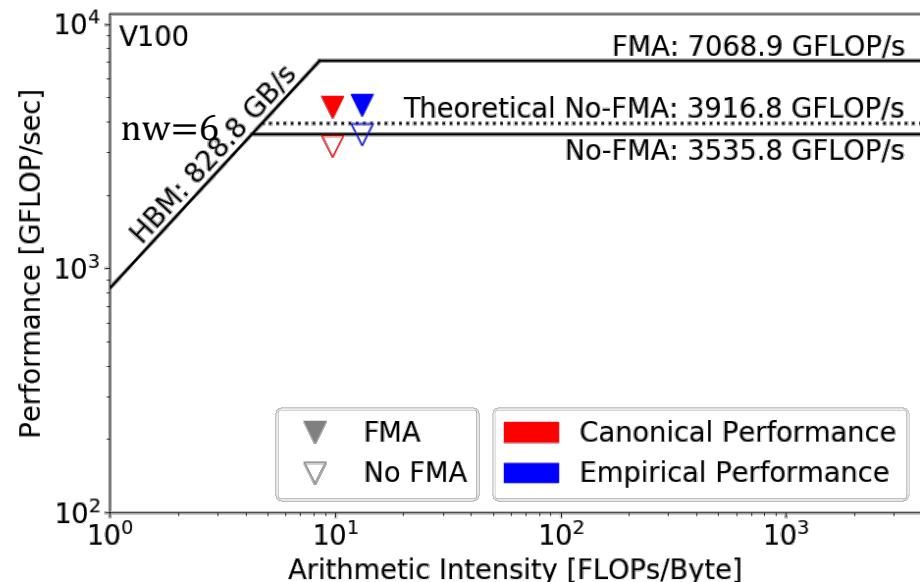
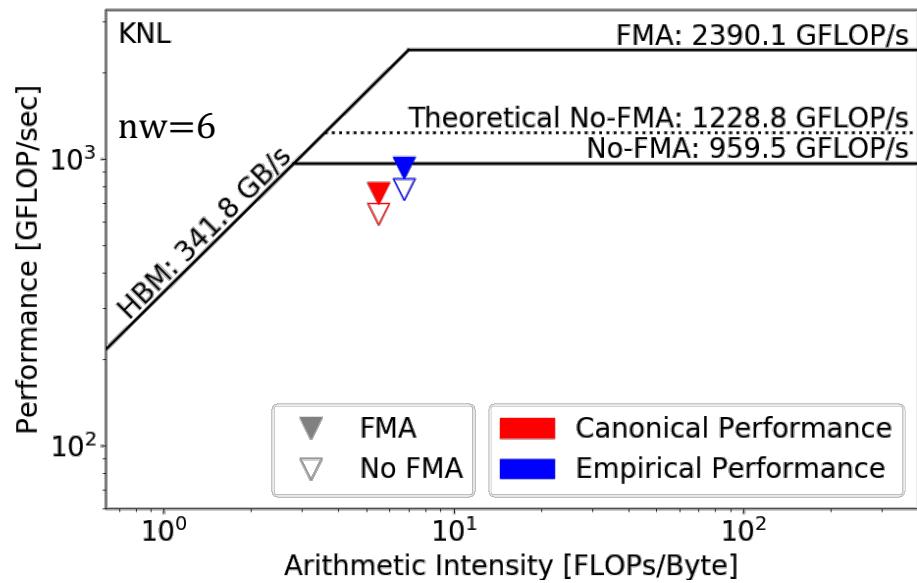
Count (GFLOPs)	KNL			V100		
	$nw = 1$	$nw = 3$	$nw = 6$	$nw = 1$	$nw = 3$	$nw = 6$
Canonical	921.4	2354.7	4504.6	895.8	2329.1	4350.9
Empirical	1055.8	2834.5	5502.7	1151.6	3096.8	5886.5
Difference	15%	20%	22%	29%	33%	35%

- Kernel performance will move diagonally up!
 - Increased GFLOP/s and arithmetic intensity (FLOPs/Byte)

Message 2: Account for Divides



- Your code may be in a different regime or closer to the ceiling than you realize!



Message 3: Roofline Capabilities



Again, test with different variants of the GPP kernel:

- Vary AI by varying `nw` from 1 to 6
- Enable/Disable FMA by compiling with `-fmad=true/false`
- Change memory access pattern by striding the `ig` loop

Platforms: Intel KNL and NVIDIA V100

Architectural Efficiency

→

Performance Portability Score

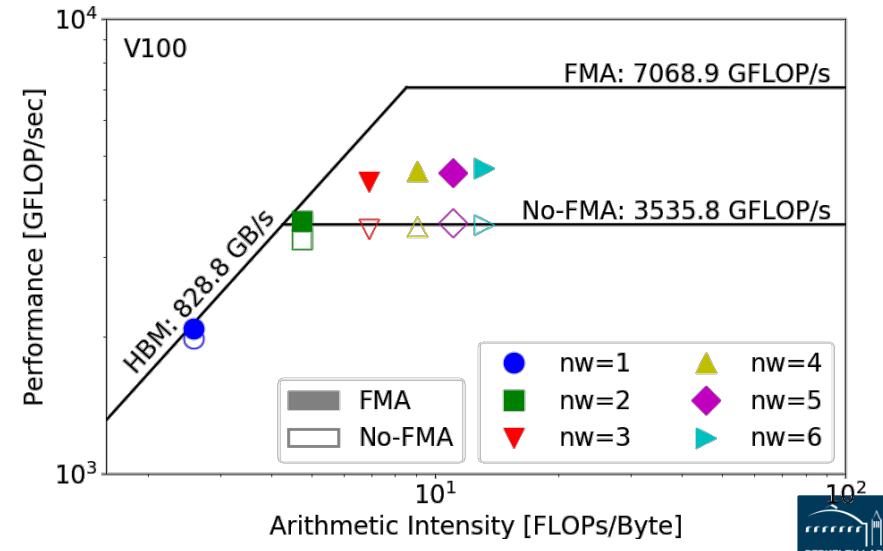
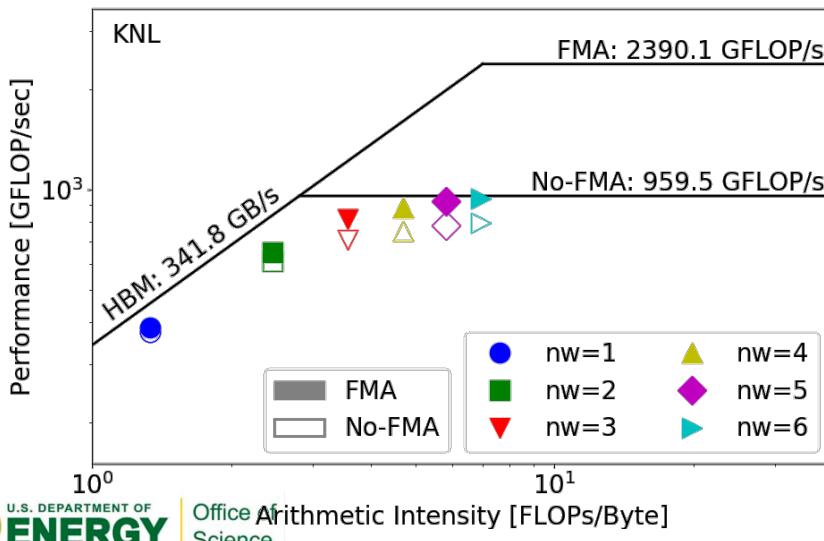
$$e_i(a, p) = \frac{P_i(a, p)}{\min(F_i, B_i \times I_i(a, p))}$$

$$\Phi(a, p, H) = \begin{cases} \frac{|H|}{\sum_{i \in H} \frac{1}{e_i(a, p)}} & \text{if } i \text{ is supported, } \forall i \in H \\ 0 & \text{otherwise} \end{cases}$$

Message 3: Roofline Capabilities



- Varying AI: bottleneck shifts at $nw = 2$ from KNL to V100
- Easier to achieve no-FMA ceiling on V100 than KNL
 - KNL issues 2 instr./cycle and executes 2 instr./cycle
 - V100 issues 4 warps/cycle and executes 1 warp/cycle (32 FP64 cores)



Message 3: Roofline Capabilities



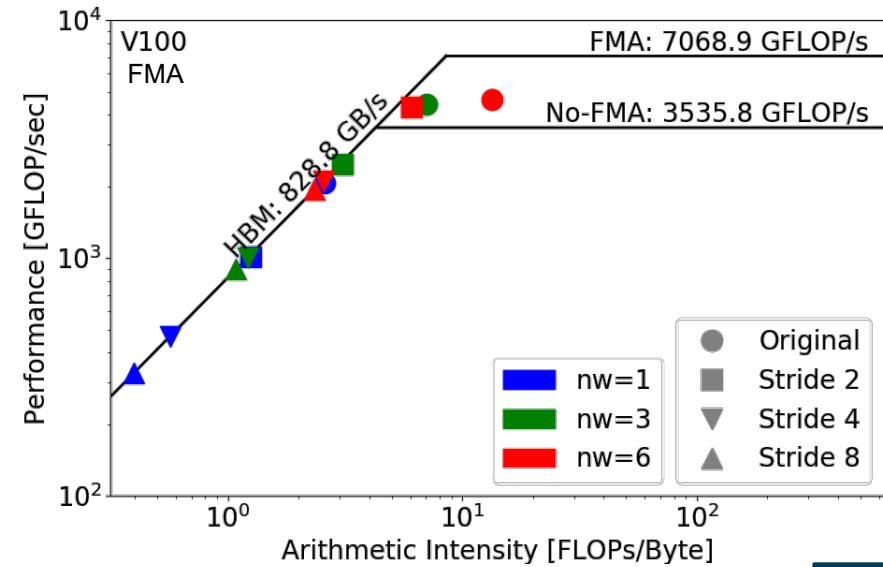
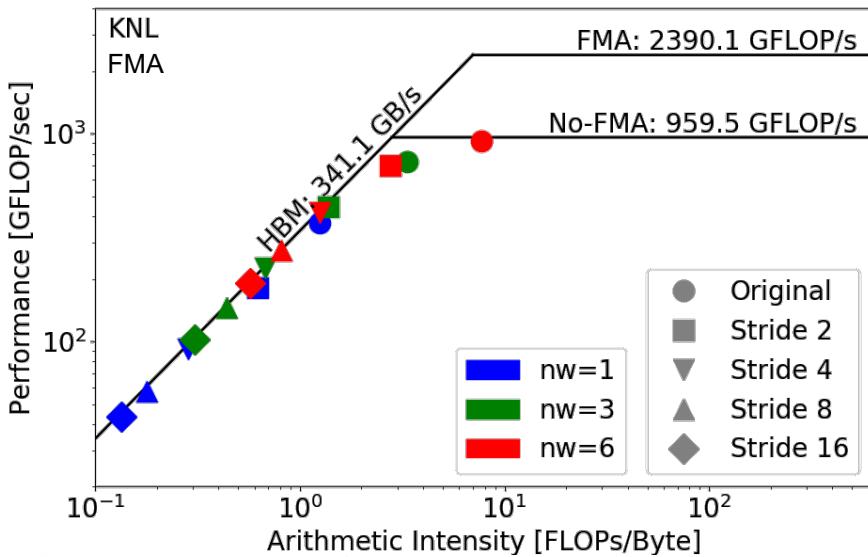
- With increasing nw (and AI):
 - No-FMA performance portability score is consistently $> 80\%$
 - FMA benefit is far less than 2x at high nw 's. Architectural efficiency suffers and so does performance portability.
- At high nw 's, increasing FMA instruction percentage is key on both platforms!

		Architectural Efficiency	$nw = 1$	$nw = 2$	$nw = 3$	$nw = 4$	$nw = 5$	$nw = 6$
No-FMA	KNL	82.06%	72.95%	73.74%	78.72%	81.28%	82.81%	
	V100	92.88%	92.88%	97.43%	98.91%	1	99.73%	
	Performance Portability	87.14%	81.72%	83.95%	87.67%	89.93%	90.49%	
FMA	KNL	84.98%	77.50%	66.77%	55.28%	46.56%	39.65%	
	V100	97.36%	91.50%	76.70%	65.44%	65.07%	66.38%	
	Performance Portability	90.76%	83.92%	71.39%	59.93%	54.28%	49.65%	

Message 3: Roofline Capabilities



- Strided memory access pattern
 - Transaction size: 64B on KNL vs. 32B on V100
 - Data: 16B per complex number



Message 3: Roofline Capabilities



- With increasing stride size
 - GPP becomes more and more bandwidth bound on both architectures, eventually all saturating HBM
- Even though performance in GFLOP/s drops, architecture efficiency grows and so does performance portability score.
- Stride- n performance is bound by a lower ceiling than stride-1 performance.

Architectural Efficiency	Original	Stride 2	Stride 4	Stride 8	Stride 16
KNL	38.40%	75.24%	98.39%	99.20%	98.00%
V100	65.64%	85.43%	98.81%	99.89%	-
Performance Portability	48.46%	80.01%	98.60%	99.55%	-

Summary and Conclusions



- Why performance portability is important and past attempts to define it and quantify it → PP Metric proposed by Pennycook *et al.*
- Methodology to collect Roofline data for performance port analysis
- Roofline is very powerful in capturing nuances of performance analysis such as changes in AI, instruction mix, instruction issue/exec bandwidth and memory access pattern.
- It is imperative to use empirical Roofline ceilings, account for complex instructions such as divides appropriately, and select relevant ceilings to compare performance with, in order to assess architectural efficiency more accurately and also perform performance portability analysis more accurately.

Reference



1. S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for multicore architectures,” Communications of the ACM, vol. 52, no. 4, pp. 65–76, 2009.
2. S. J. Pennycook, J. D. Sewall, and V. W. Lee, “A metric for performance portability,” arXiv:1611.07409, 2016.
3. S. J. Pennycook, J. D. Sewall, and V. W. Lee, “Implications of a metric for performance portability,” Future Generation Computer Systems, 2017.
4. C. Yang, R. Gayatri, T. Kurth, P. Basu, Z. Ronaghi, A. Adetokunbo, B. Friesen, B. Cook, D. Doerfler, L. Oliker, J. Deslippe, and S. Williams, “An Empirical Roofline Methodology for Quantitatively Assessing Performance Portability,” P3HPC workshop, 2018.
5. S. Williams, and T. Koskela, “Using the Roofline Model and Intel Advisor,” IDEAS webinar, Aug 16 2017.



U.S. DEPARTMENT OF
ENERGY

Office of
Science



Thank You!

