Implications of a Metric for Performance Portability: Necessity of Specialization and Application-Specific Abstractions

S. J. Pennycook, J. D. Sewall and V. W. Lee
Intel Corporation
DOE COE Performance Portability Meeting 2017, Denver, CO

Acknowledgements:
Christian Trott (Sandia National Laboratories), Tom Deakin (University of Bristol), Roland Schulz (Intel Corporation)

Defining "Performance Portability" (1/2)

"Let us not get tied up in definitions. ***Performance portability means different things to different people and we need to accept that****. Both performance and portability are poorly defined and depend on the applications. Every app has different constraints and there is no way to get around it."*

- Unnamed Participant

DOE COE Meeting 2016

(Emphasis mine)

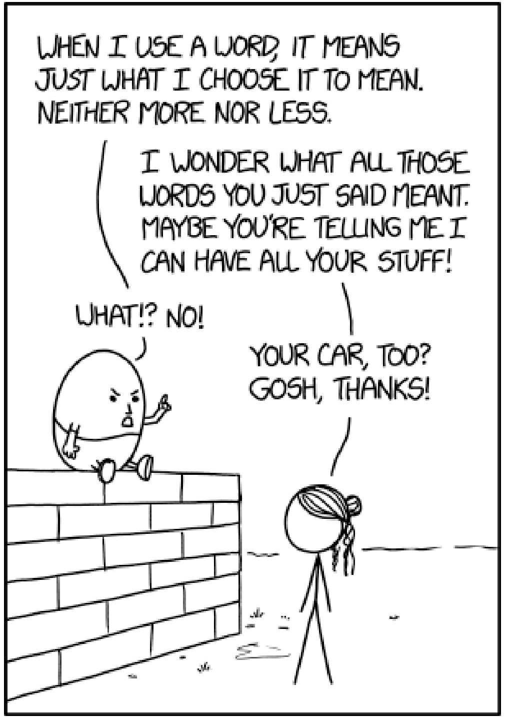


Image created by Randall Munroe and unmodified from https://xkcd.com/1860/, licensed under public license at https://xkcd.com/license.html (CC BY-NC 2.5), including all disclaimers and warranties.

Defining "Performance Portability" (2/2)

*"An approach to application development, in which developers **focus on providing portability** between platforms **without sacrificing performance**."*

*"The ability of the **same source code** to **run productively** on a variety of different architectures."*

*"The ability of an application to achieve a **similar** high **fraction of peak performance** across target devices."*

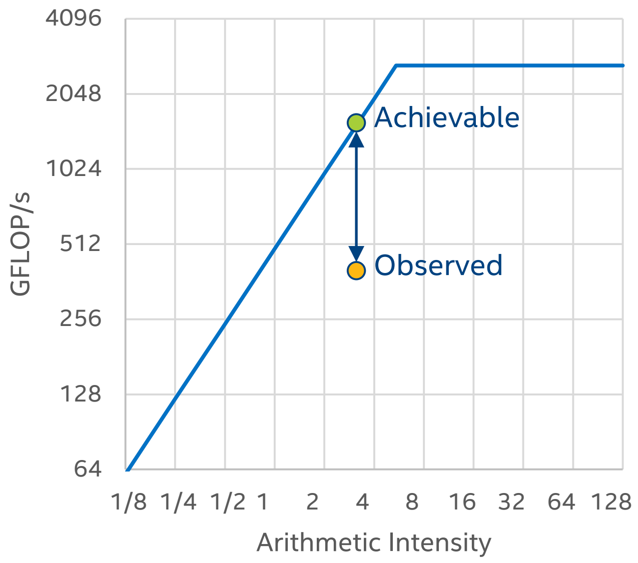
*"The ability of an application to obtain the same (or **nearly the same**) **performance as a variant** of the code that is written specifically for that device."*

*"The ability of an application to execute with a **performance difference of less than 2x** on two different systems, **without significant software changes**."*

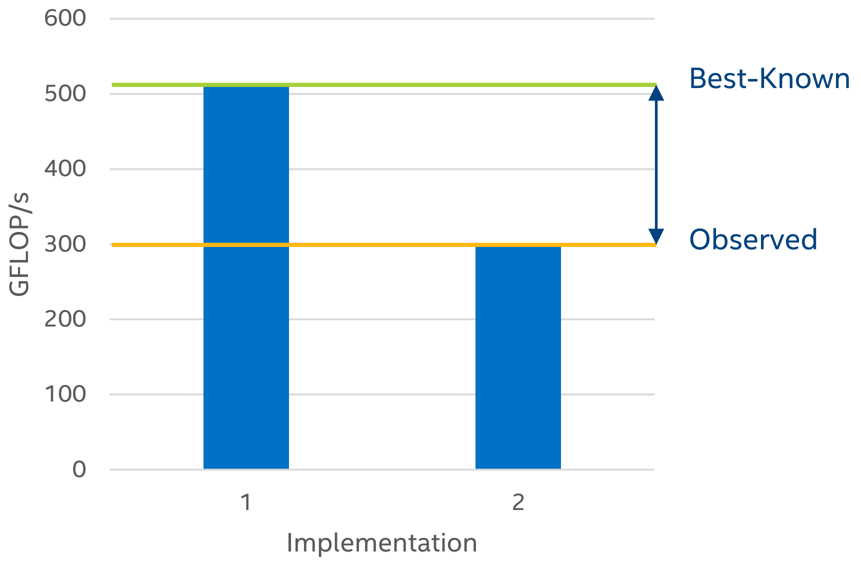
Existing definitions are **subjective** and may not reflect **application performance**.

Our Proposed Definition

Performance Portability
"A **measurement** of an application's **performance efficiency** for a given problem that can be executed correctly on all platforms in a given set."



Architectural Efficiency = observed : achievable



Application Efficiency = observed : best-known

Performance results are for illustration purposes only and not intended to express or imply real world results.

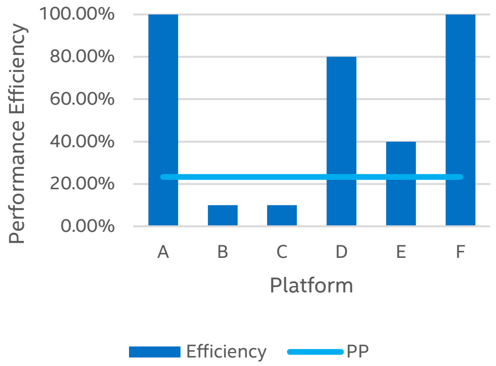
Our Proposed Metric

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

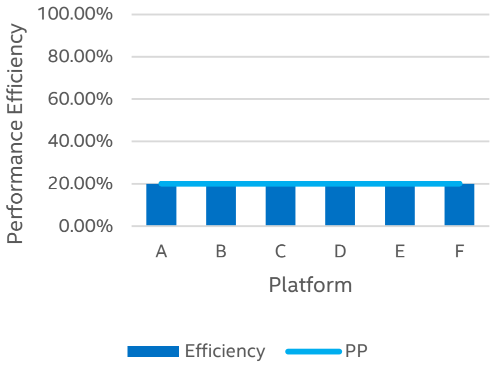
$e_i(a,p)$ = efficiency of application a for input problem p.

"The **harmonic mean** of an application's performance efficiency on a set of platforms for a given problem."

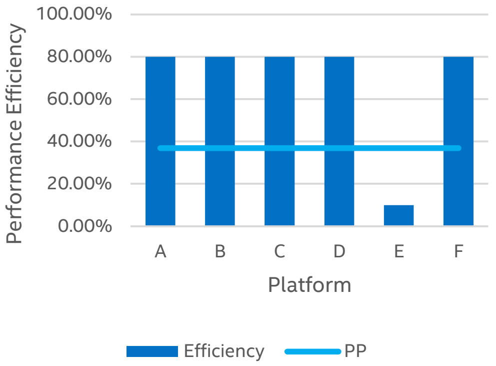
Application 1 – PP(*a*,*p*,*H*) = **23.30%**



Application 2 – PP(*a*,*p*,*H*) = **20.00%**



Application 3 – PP(*a*,*p*,*H*) = **36.92%**



Performance results are for illustration purposes only and not intended to express or imply real world results.

Wait! What About "Productivity"?

- Our definition is orthogonal to productivity, **not** incompatible with it:
  - "How many source code changes are required to achieve PP of y?"

- Productivity is even more subjective than PP!
  - Developers have different skill levels.
  - Codes differ in size and complexity.
  - Libraries and frameworks hide development costs.

- Attend our **Breakout Session:**
  "Performance, Portability and Productivity: Definitions & Metrics"

PP(a,p,H) Case Study: The BabelStream Benchmark

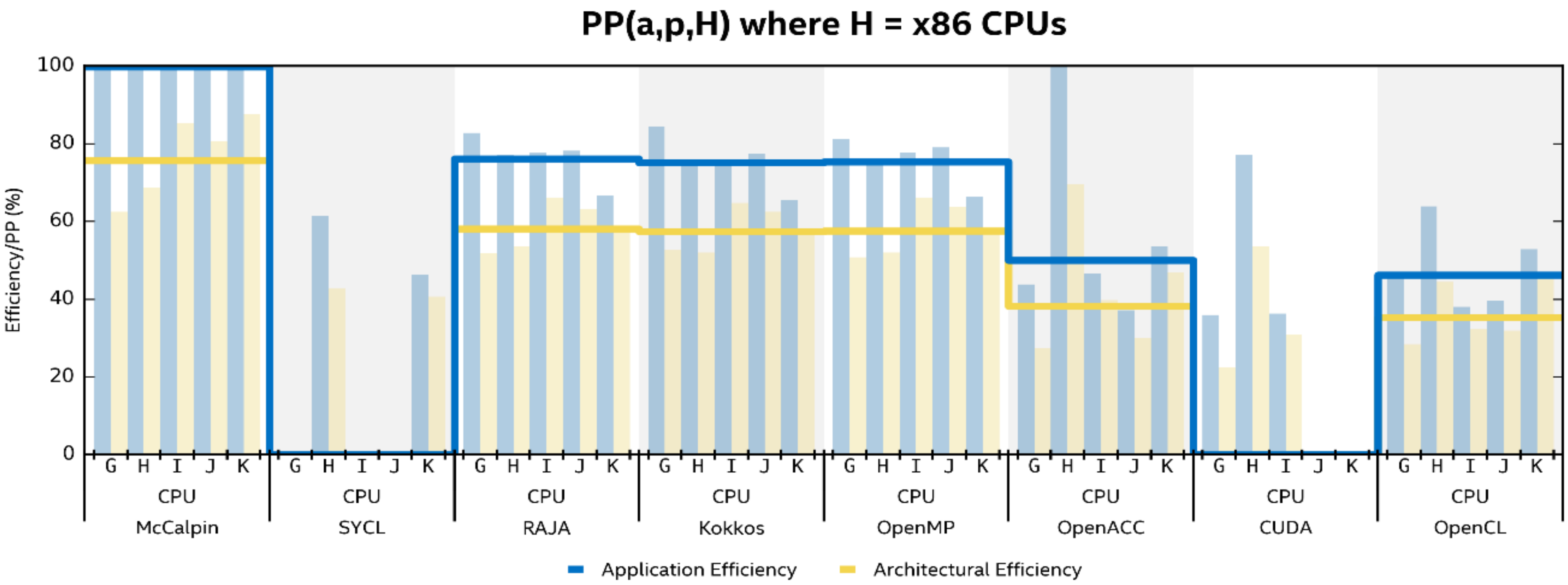
- Developed at University of Bristol; implements STREAM Triad in 7 programming languages/models:

SYCL	C++ wrappers for OpenCL
RAJA	Loop abstractions from Lawrence Livermore
Kokkos	Device/memory abstractions from Sandia
OpenMP* (with C++)	Standard pragmas for parallel programming
OpenACC*	Standard pragmas for accelerator programming
CUDA*	Proprietary language for stream programming
OpenCL*	Standard language for stream programming

- Results published for 12 platforms (incl. CPUs and GPUs) [1], of which we focus on 9.
  Ongoing investigations into performance portability improvements [2,3].

[1] T. Deakin, J. Price, M. Martineau and S. N. 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*, 2017
[2] 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 (to appear)
[3] 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

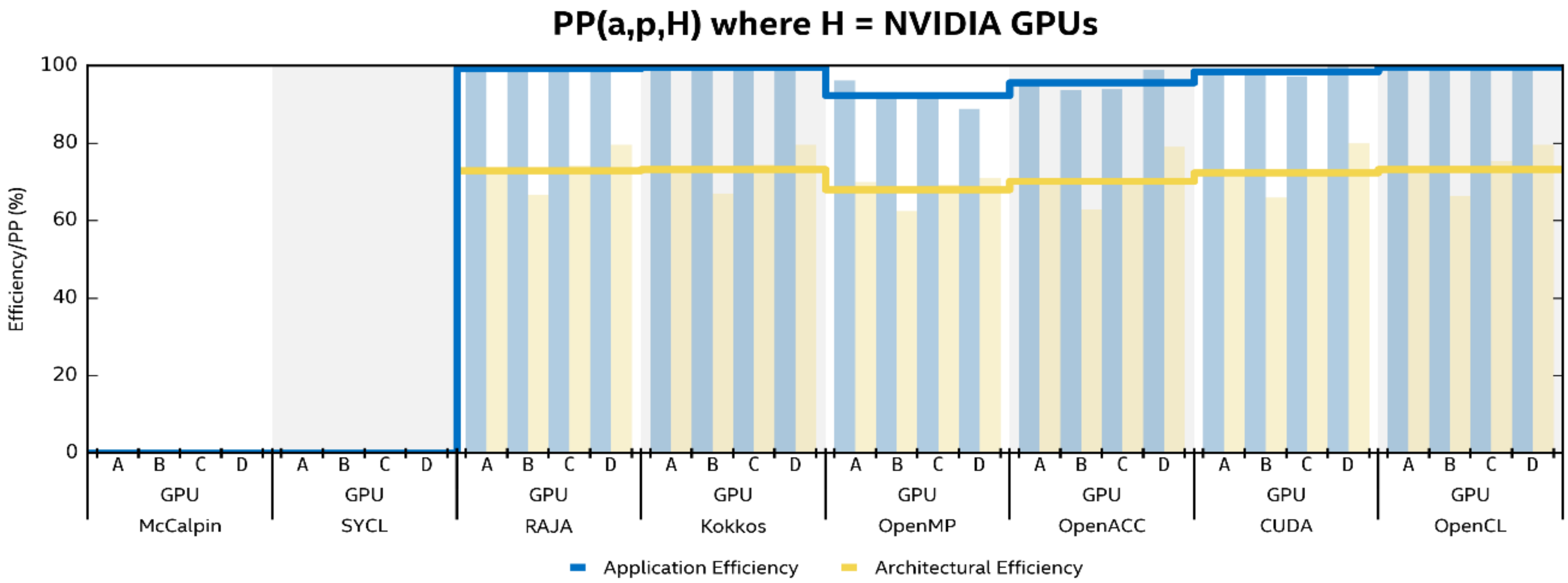
Performance Portability of BabelStream (2016)



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.

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 23.

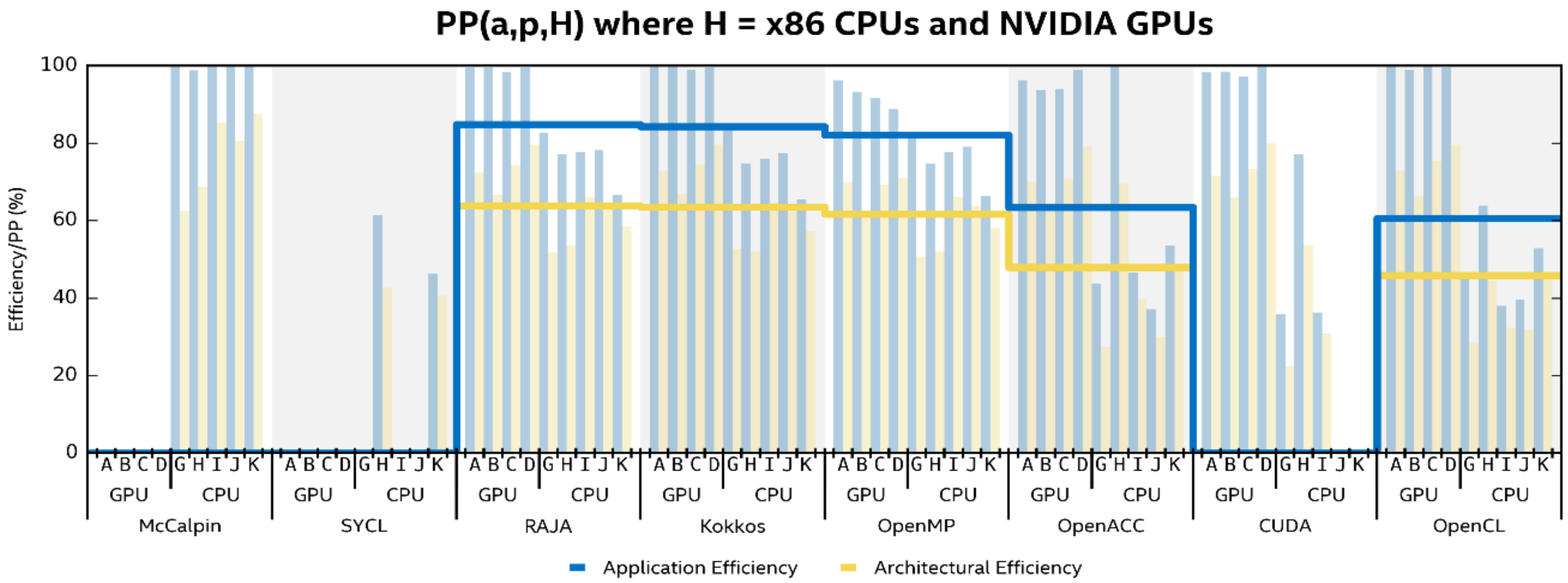
Performance Portability of BabelStream (2016)



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.

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 23.

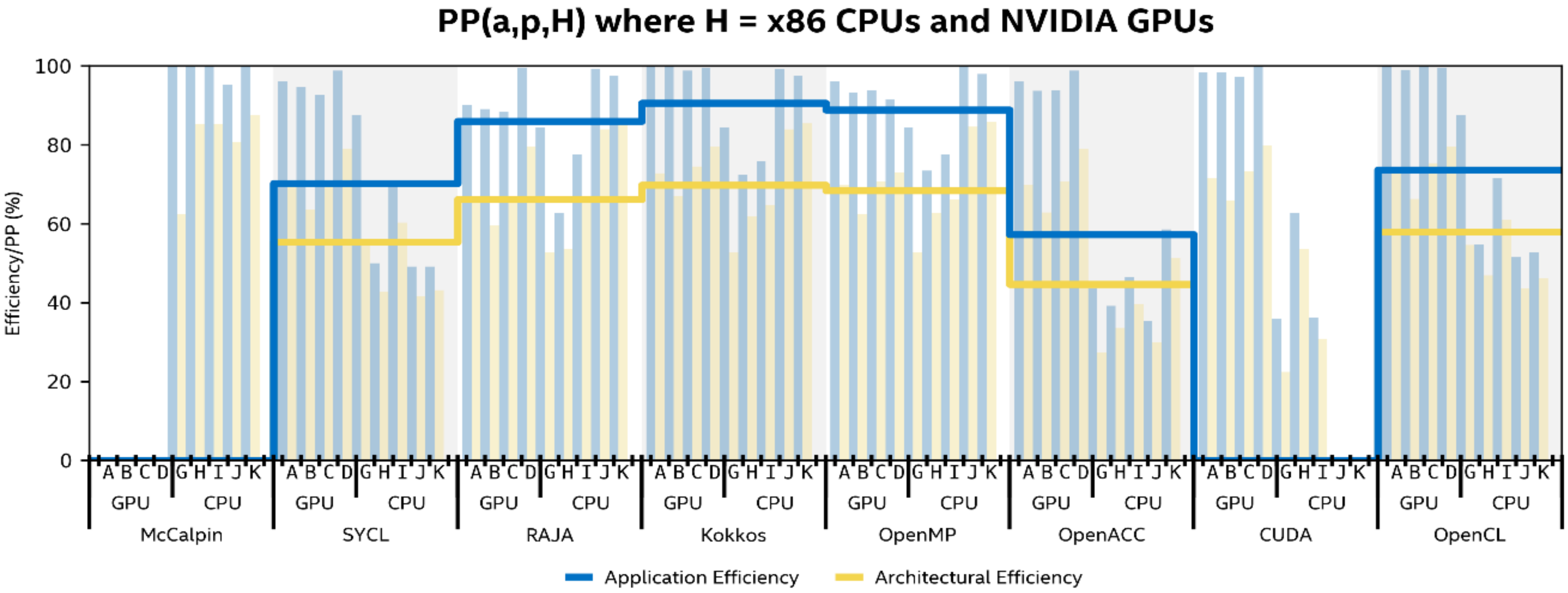
Performance Portability of BabelStream (2016)



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.

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 23.

Performance Portability of 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.

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 24.

Implications of a Metric for Performance Portability

- Enables users to:
  - Compare PP applications/libraries/framework support for their platforms
  - Reason about which of many PP options to choose
  - Pressure developers to focus on platforms with poor support

- Enables developers to ask:
  - What value of PP is realistic/achievable?
  - What value of PP should we be aiming for?
  - **What are the best development practices for achieving high values of PP?**

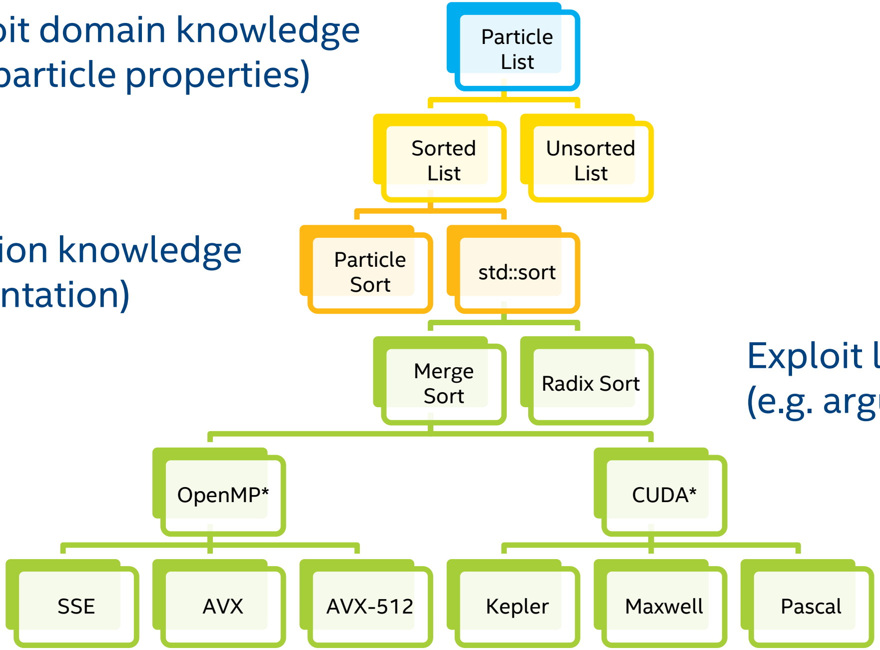
Performance Portability => Specialization

Exploit domain knowledge
(e.g. particle properties)

Exploit application knowledge
(e.g. list representation)

Exploit library knowledge
(e.g. argument types)

Exploit compiler knowledge
(e.g. instruction latencies)



All approaches to PP specialize code for a target platform; the distinction is how/where.

Disclaimer

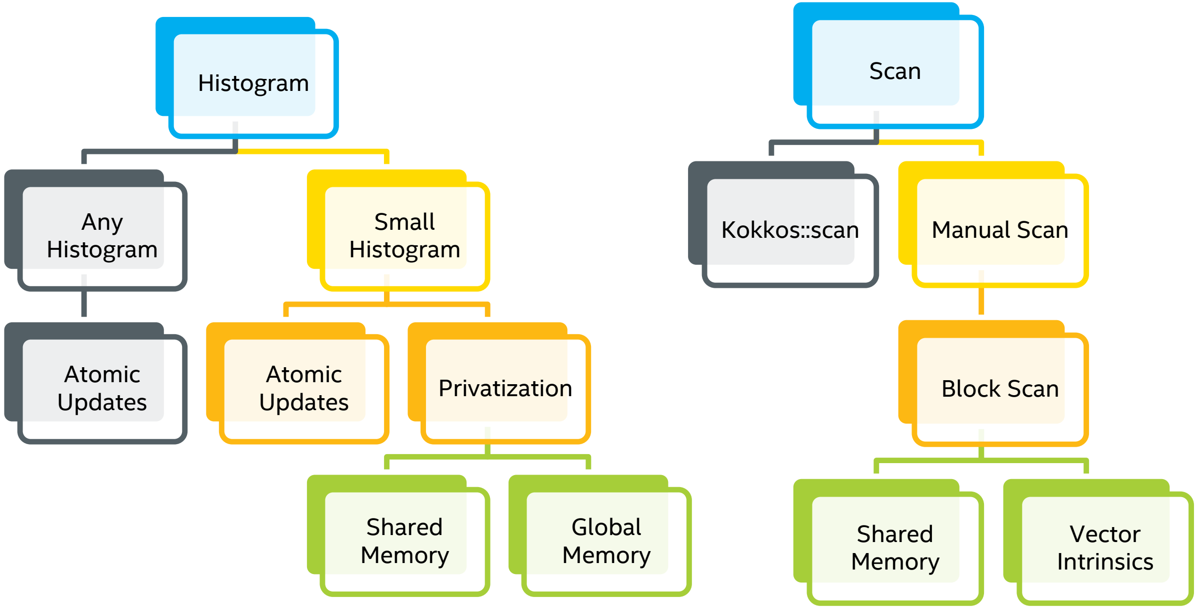
- My level of familiarity with languages frequently associated with PP is:
  - OpenMP*
  - CUDA* / OpenCL*
  - Kokkos
  - Thrust
  - C++17
  - RAJA
  - OpenACC*
  - …all the others
- Focus (and correctness) of remaining slides follows from the above.

Specialization Case Study (1/2)

- Developed four variants of several benchmarks from the CUDA* SDK:
  1. CUDA
  2. OpenMP*
  3. Kokkos
  4. Kokkos (Specialized)

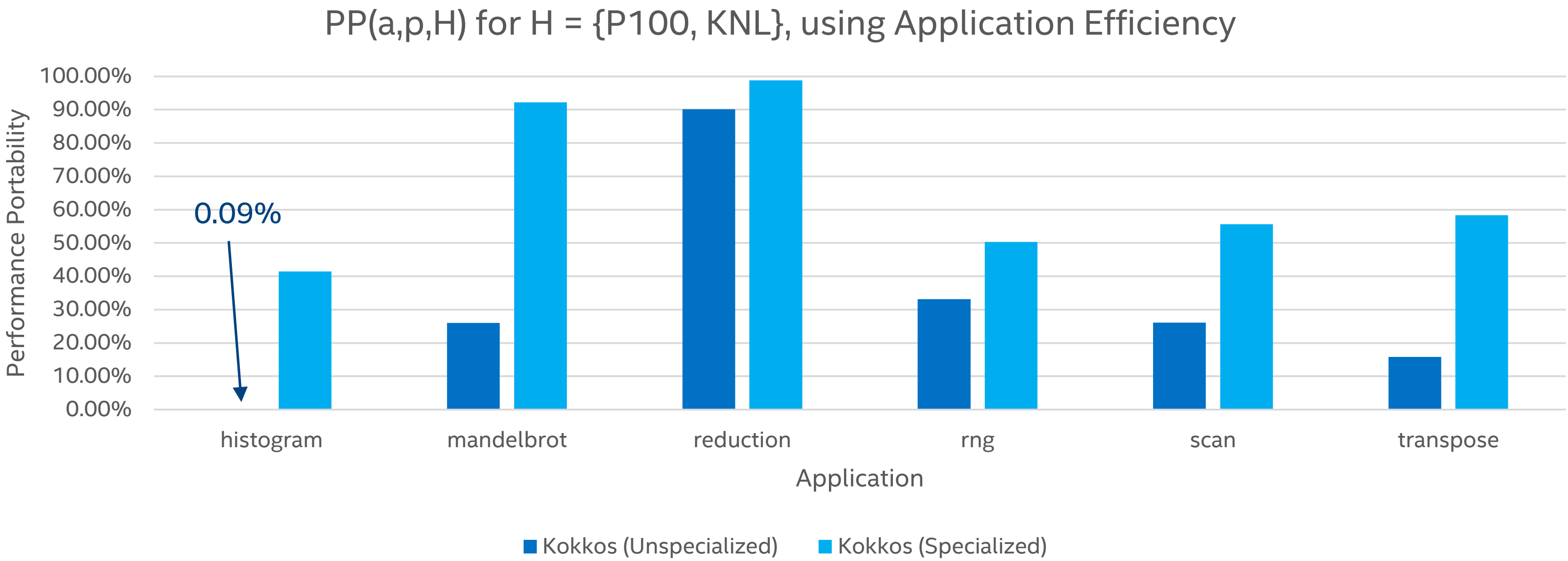
- 4 is a "single-source" code augmented with specialized variants of some functions.
  - All functions have the same API irrespective of target device
  - Each application uses a different API: "**Application-Specific Abstraction**"
  - Specializations include: data layout, data accessors, functors, execution policies

Specialization Case Study (2/2)



Left-most path is the "base" version, a generic implementation that can run on any device.

Impact of Specialization on PP(a,p,H)



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

For configuration details, see Slide 25.

User-Driven Specialization Today†

- CUDA*/OpenCL*: Query API + Just-in-Time (JIT) compilation.
- OpenMP*: Override function calls by SIMD/allocator traits.
  - `#pragma omp declare simd [clauses]`
  - `#pragma omp declare alloc [clauses]` ‡
- Kokkos: Specialize functionality by "Device" ("Backend")
  - `template <class Device>`
  - `void operator(Tag& tag, …) (…)`
- RAJA/C++/Thrust: Specialize functionality by "Policy" ("Runtime"/"Schedule")
  - `void ParallelFor(ExecutionPolicy& policy, int begin, int end, Functor f);`

† Inexact syntax used to highlight similarities/differences between approaches.
‡ Under consideration for OpenMP TR6.

User-Driven Specialization Tomorrow†? (1/2)

- Directives: Override function calls by matching traits.
  - `#pragma pp declare variant(variant-name) implements(base-name)`
    `match(trait-name:trait-value[,trait-name:trait-value]*)`
  - `#pragma pp dispatch match(trait-name[,trait-name]*)`

- Example:

```
#pragma pp declare function match(isa)
#pragma pp declare function variant(_mm_add) match(isa:sse)
double add(double a, double b);

#pragma pp declare function implements(add) match(isa:avx512)
__m512 _mm512_add(_mm512 a, __m512 b);

#pragma pp dispatch match(isa)
c = add(a, b);
```

† Syntax proposed here is at the draft/prototype stage and has not been accepted by any standards or language committee.

User-Driven Specialization Tomorrow†? (2/2)

- C++ (and C++ Frameworks): Override functionality by matching traits.
  - Traits could be standardized (C++20XX) or specific to PP framework(s).

- Example:

```
struct functor : public pp::base
{
  // inherits isa = pp::traits();
  …
};

struct functor_avx512 : public pp::specialization
{
  static constexpr auto isa = pp::traits(avx512);
  …
};

pp::dispatch<functor,Context> f;
ParallelFor(policy, start, end, f);
```

† Syntax proposed here is at the draft/prototype stage and has not been accepted by any standards or language committee.

Summary

- Shared definitions and metrics have many benefits and we should develop them
  - Agree or disagree at "Performance, Portability and Productivity: Definitions & Metrics"

- Proposed a realistic approach to achieving high performance portability PP(a,p,H)
  - Maintain a single "base" code that is expected to work anywhere (portability)
  - User-driven specialization to override functionality for important cases (performance)
  - Add support for this approach to standard programming languages/frameworks (productivity / maintainability)

- For more detail, see:
  - S. J. Pennycook, J. D. Sewall, V. W. Lee, "A Metric for Performance Portability", in *Proceedings of the International Workshop on Performance Modeling, Benchmarking and Simulation*, 2016 https://arxiv.org/abs/1611.07409
  - S. J. Pennycook, J. D. Sewall, V. W. Lee, "Implications of a Metric for Performance Portability", in *Future Generation Computer Systems*, 2017 https://doi.org/10.1016/j.future.2017.08.007

Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Intel, Xeon, Xeon Phi and the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries.

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

Experimental Setup (1)

Results on Slides 8-10 from:

- 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*, 2017 (Configuration: see Section 4)

Experimental Setup (2)

Results on Slide 11 are a combination of results from:

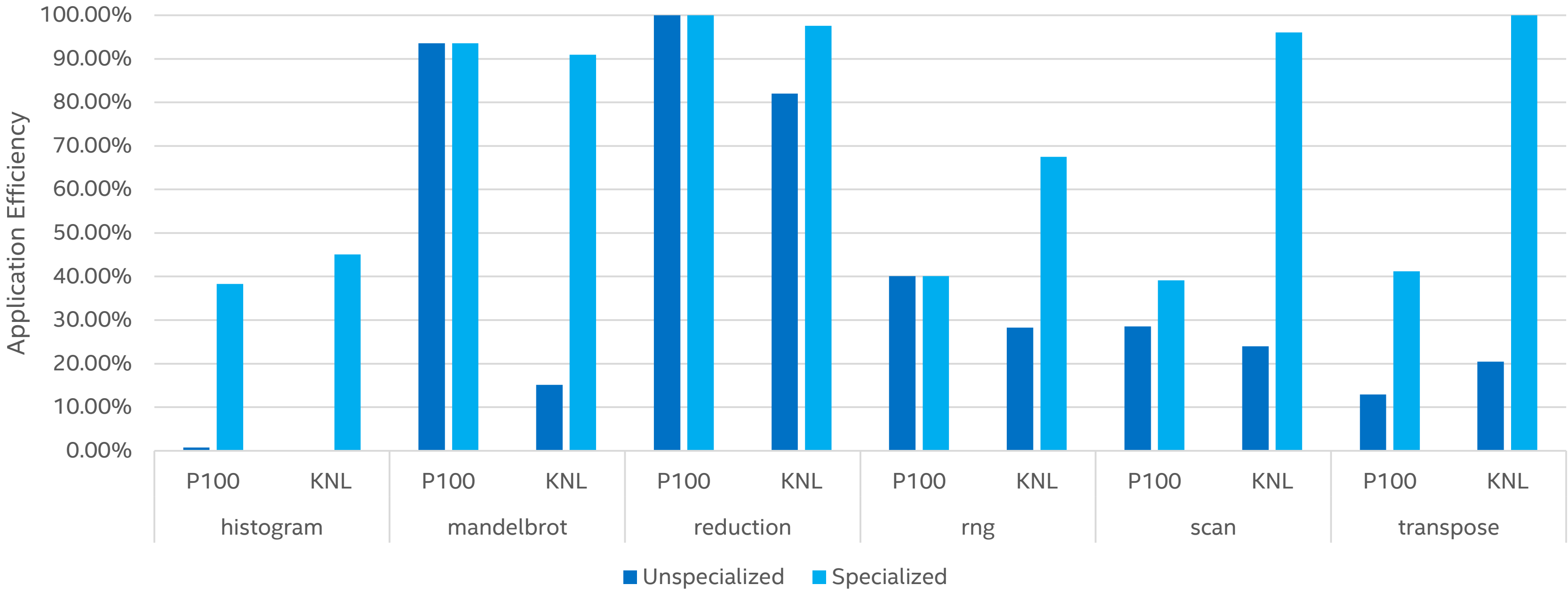
- T. Deakin, J. Price, M. Martineau and S. McIntosh-Smith, “GPU-STREAM v2.0: Benchmarking the Achievable Memory Bandwidth of Many-Core Processors Across Diverse Parallel Programming Models”, in *Proceedings of the Workshop on Performance Portable Programming Models for Accelerators*, 2017 (Configuration: 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 (to appear)
- 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)

Experimental Setup (3)

Results on Slide 17 and 26 use the following experimental setup:

- **P100:** Intel® Xeon® processor E5-1697 v4, 2.3 GHz, 2 sockets x 18 cores + Tesla P100-PCIE-16GB, BIOS: 86.00.26.00.01, ECC Enabled, Persistence Mode Disabled, Graphics/SM 405 MHz, Memory 715 MHz, CUDA 8.0.44
- **KNL:** Intel® Xeon Phi™ processor 7250, 68 core, 272 threads, 1400 MHz core freq. (turbo on), 1700 MHz uncore freq., MCDRAM 16 GB 7.2 GT/s, DDR4 96GB 2400 MHz, CentOS 7.2.1511, Quad cluster mode, MCDRAM Flat memory mode
- **Versions:**
  - Kokkos: git commit da3144
  - gcc: 4.8.5 20150623
  - icc: 18.0.0 20170510
- **Compiler Flags**:
  - P100: KOKKOS_ARCH=Maxwell KOKKOS_DEVICES=Cuda
  - KNL: KOKKOS_ARCH=KNL KOKKOS_DEVICES=OpenMP, icpc -O3 –xMIC-AVX512

Impact of Specialization on PP(a,p,H)



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

For configuration details, see Slide 25.

intel®