The Development and Uses of Metrics for Performance, Portability, and Productivity

John Pennycook, Jason Sewall, Doug Jacobsen
Intel Corporation
P3HPC Forum 2020

Acknowledgements:
Lawrence Livermore National Laboratory, NERSC, NVIDIA*/PGI*, Sandia National Laboratories, University of Bristol

2016: Motivation

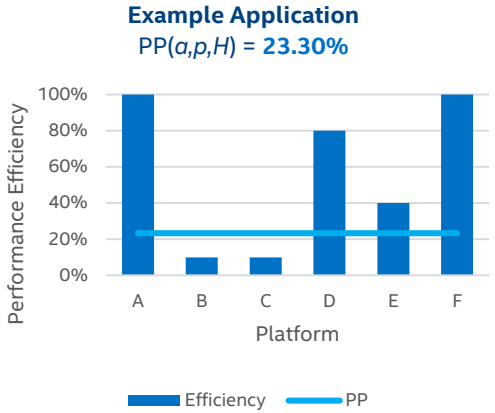
- Workshops and frameworks abound, but no consensus on the end goal:
  - Did this commit make things better or worse?
  - How do different approaches compare?
  - What does it mean if PP appears in an RFP?

- We decided to take an application-centric view:
  1. Is it performance portable?
  2. What performance does it achieve “on average” (over platforms/inputs)?
  3. How similar is the performance efficiency achieved on different platforms?
  4. What performance can I expect if I introduce a new platform?
  5. How difficult is it to write/maintain?

2017: Definition and Metric

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

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



Progress

- Yes/No answer for "is it PP?"
- Captures "average" performance in H
- Architectural and Application Efficiency

Challenges & Future Work

- Doesn't account for productivity
- Loses information about distribution
- Computing efficiency can be difficult

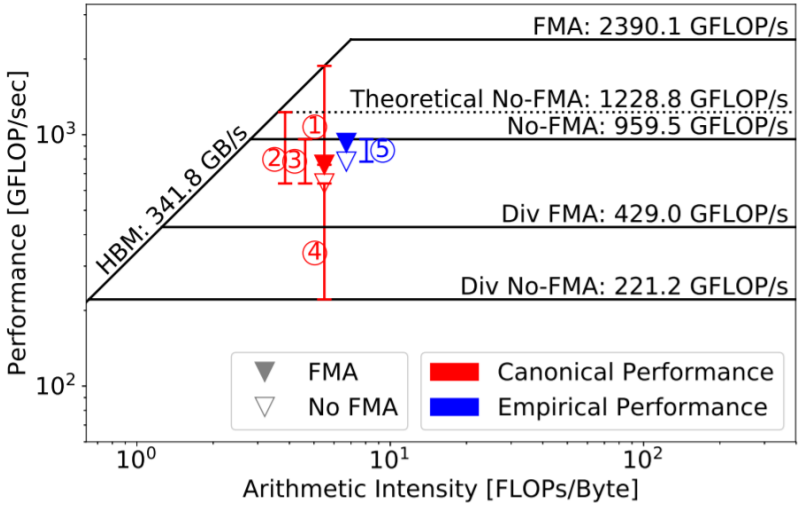
S. J. Pennycook, J. D. Sewall and V. W. Lee, "A Metric for Performance Portability", PMBS 2017

2018: Architectural Efficiency from Roofline Model

Plug in Roofline model in place of architectural efficiency:

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

May need to select a **different bound** for different platforms.



Progress

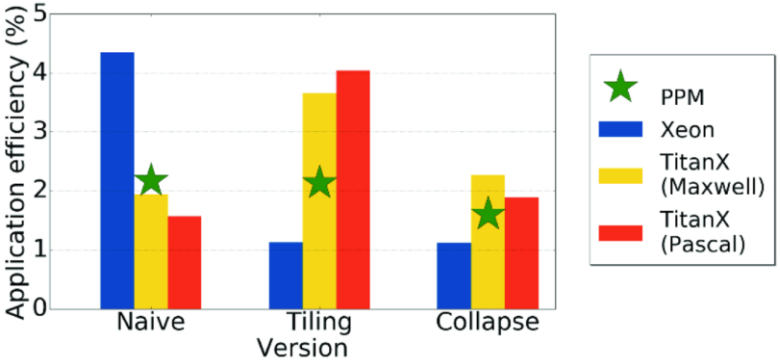
- Automatic computation of efficiency with higher accuracy than simple throughput
- Demonstrated importance of choosing correct ceiling when computing efficiency

Challenges & Future Work

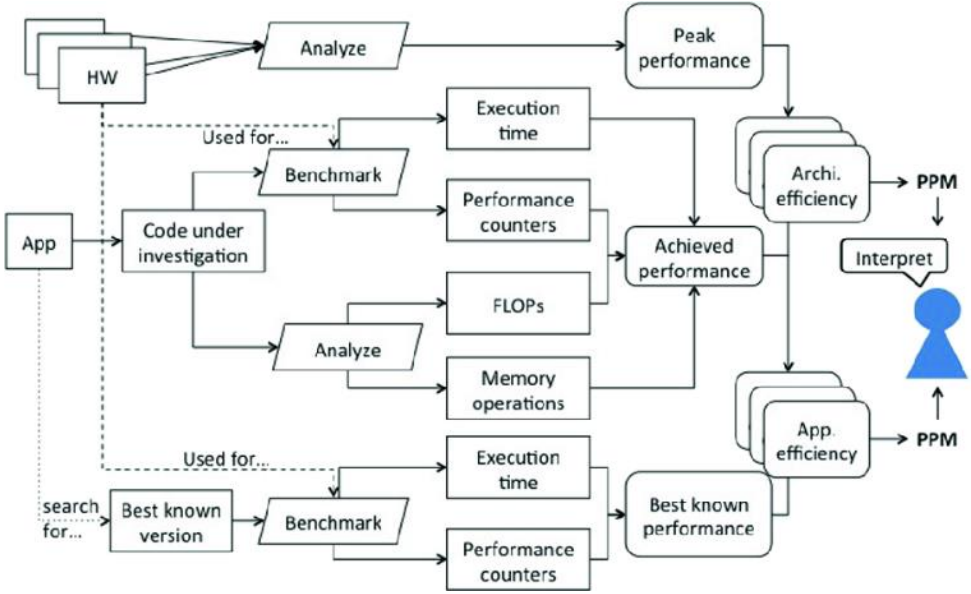
- Refining roofline eventually guarantees 100% architectural efficiency!

C. Yang et al., “An Empirical Roofline Methodology for Quantitatively Assessing Performance Portability”, P3HPC 2018
T. Zhao et al., “Delivering Performance-Portable Stencil Computations on CPUs and GPUs Using Bricks”, P3HPC 2018

2018: A Beginner's Guide



(f) Matrix Multiplication (app.)



Progress

- Identified that PP can be skewed by using many similar platforms
- Highlighted tension in optimizing for PP

Challenges & Future Work

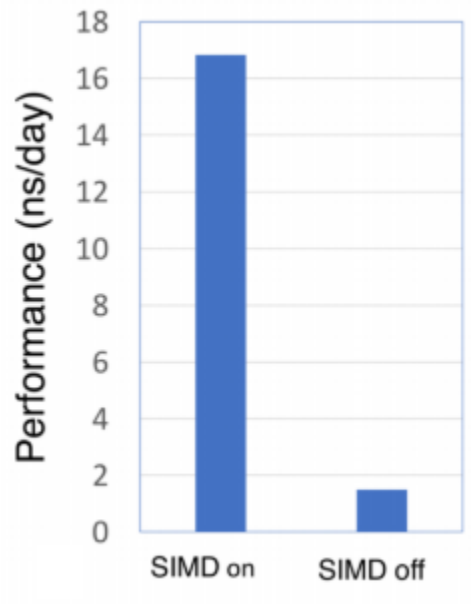
- Proposed idea of a heterogeneity metric as a confidence score
- Proposed categorization of optimizations and a database of best-known versions

2018: PP MD

$$PP_{MD_c}(a,p,Q) = \begin{cases} \frac{|Q|}{\sum_{i \in Q} S_i(a,p)}, & \text{if } |G| - |Q| \neq 0 \\ & \text{and } |G| - |Q| \neq |G| \\ 1, & \text{if } |G| - |Q| = |G| \\ 0 & \text{if } |G| - |Q| = 0, \end{cases}$$

where:

- G captures components significantly improving performance
- Q captures non-portable components
- S is speed-up.



Progress

- Penalizes codes that would require significant effort to port

Challenges & Future Work

- Requires manual identification of application components
- Metric does not support multiple platforms without additional averaging

A. Sedova et al., "High-Performance Molecular Dynamics Simulation for Biological and Materials Sciences: Challenges of Performance Portability", P3HPC 2019

2018: Productivity Logs & Code Divergence

https://github.com/lanl/SHELTIE/

$$\binom{|H|}{2}^{-1} \sum_{\{i,j\} \in H \times H} d(c_i, c_j)$$

where:
- H = set of platforms
- c_i = code required to compile and execute correctly on platform i.

Progress

- git-hooks for tracking LOC changes and performance portability of each commit
- Partial productivity metrics

Challenges & Future Work

- Difficult to compute divergence
- Assumes each platform is a distinct code base (or git branch)

S.L Harrell, J. Kitson et al., "Effective Performance Portability", P3HPC 2018

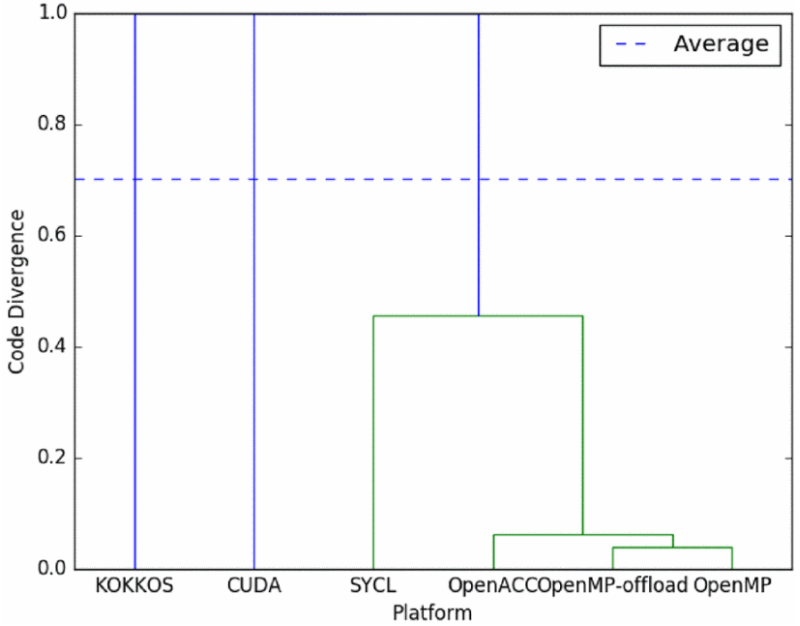
2019: Code Base Investigator

Compute code divergence using Jaccard distance between different implementations.

$$(c_i, c_j) = \frac{|c_i \cup c_j| - |c_i \cap c_j|}{|c_i \cup c_j|}$$

"The ratio of **platform-specific code** to code used by both platforms."

https://github.com/intel/code-base-investigator



Progress

- Divergence no longer tied to git commit
- Intuitive visualization of code similarity between platforms

Challenges & Future Work

- Only captures static specialization
- Doesn't support C++ templates

S. J. Pennycook, J. D. Sewall, D. W. Jacobsen, "Code Base Investigator", P3HPC Forum 2019
I. Z. Reguly, "Performance Portability of Multi-Material Kernels", P3HPC 2019

2019: PP Divergence

$$\delta(a,\alpha) = \frac{|f_i(a,p,s) - f_i(\alpha,p,s)|}{f_i(\alpha,p,s)}$$

$$\Delta_{RMS} = \sqrt{\frac{\sum_{s\in S}\delta(a,\alpha)^2}{|S|}}$$

$$P_D = \frac{\sum_{i\in H}\Delta_{RMS}}{|H|}$$

TABLE VII
Δ_{RMS} AND $\mathcal{P}_\mathcal{D}$ OF THE BENCHMARKS

	App.	Δ_{RMS} K80	P100	E5-2630	E5-2699	$\mathcal{P}_\mathcal{D}$
OpenACC	*ep*	24.02%	0%	26.96%	25.10%	19.02%
	cg	14.51%	70.24%	7.57%	2.47%	23.70%
	sp	0%	0%	0%	0%	0%
	bt	0%	0%	23.14%	0%	5.78%
	stencil	9.13%	0%	105%	0%	28.69%
	lbm	0.34%	1.21%	14.92%	27.63%	11.02%
	mri-q	13.97%	0.93%	0%	0%	3.73%
Kokkos	*ep*	0.90%	27.34%	0%	0%	7.06%
	cg	0.81%	1.50%	4.82%	7.70%	3.71%
	sp	588%	218%	55.48%	155%	254%
	bt	966%	764%	0.44%	110%	460%
	stencil	44.46%	46.53%	36.74%	42.88%	42.65%
	lbm	1.24%	7.07%	0%	0%	2.08%
	mri-q	0%	5.09%	190%	116%	78.04%

Progress

- Summarizes performance over platforms **and** input problems
- Represents average distance from the best-known implementation

Challenges & Future Work

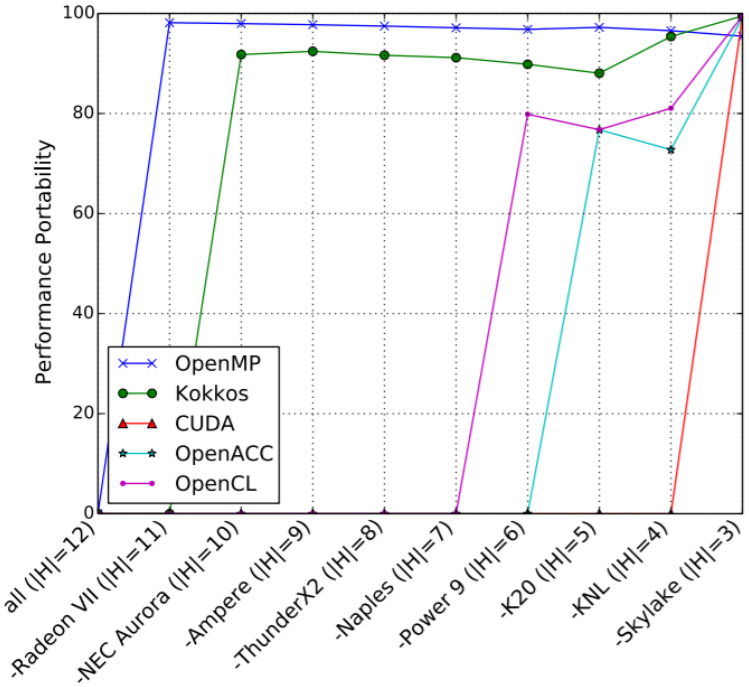
- Score can be >100% if performance is reported as time; differs from throughput
- Needs to be evaluated for more platforms

D. Daniel et al., "On Applying Performance Portability Metrics", P3HPC 2019

2019: Bristol Case Studies

Higher is better

	BabelStream	TeaLeaf	CloverLeaf	Neutral	MiniFMM	Mean	Std. Dev.
OpenMP CPU	98.4%	100.0%	100.0%	100.0%	100.0%	99.7	0.6
Kokkos CPU	83.0%	49.8%	60.7%	77.6%	66.1%	67.5	11.9
OpenMP GPU	95.5%	22.5%	0.0%	0.0%	0.0%	23.6	37.0
Kokkos GPU	99.5%	64.3%	85.7%	51.1%	60.4%	72.2	17.7
OpenMP all	97.3%	43.6%	0.0%	0.0%	0.0%	28.2	38.5
Kokkos all	88.5%	54.4%	68.2%	65.0%	63.9%	68.0	11.2



Progress

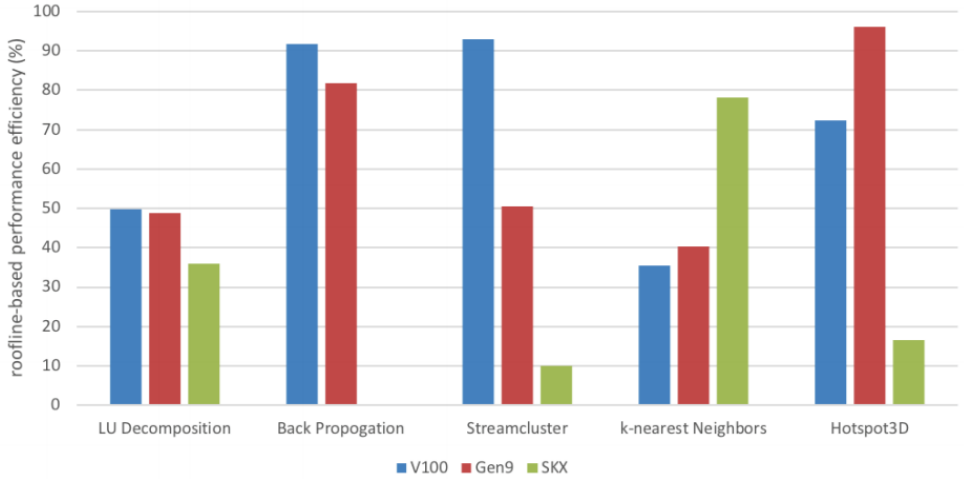
- Comparison of languages backed by enough data to be interesting
- Interesting visualization for comparing impact of platform selection on PP

Challenges & Future Work

- Order of platforms on x-axis of graph is application-specific
- Lots of data to explore and interpret!

T. Deakin et al, “Performance Portability Across Diverse Computer Architectures”, P3HPC 2019

2020: Argonne OpenCL Case Study



Benchmark kernel	Perf. portability (%)	Std. Dev.
LUD-internal	43.815	8.39
BP-AW*	86.419*	7.00*
SC-pgain	22.947	25.93
KNN-nearest	45.617	16.82
HS-opt1	35.332	35.1

* Only considering the V100 and Gen9

Progress

- Augments PP with standard deviation to capture spread of results

Challenges & Future Work

- Standard deviation is typically defined relative to arithmetic mean
- Unclear what it “means” to calculate standard deviation from PP

C. Bertoni et al, “Performance Portability Evaluation of OpenCL Benchmarks Across Intel and NVIDIA Platforms”, IPDPSW 2020

Lessons Learned

1. We need different tools for different analyses

2. Growing consensus around P3 terminology

3. The community is interested in two different kinds of productivity:
   - Effort required to develop (performance) portable codes today
   - Effort required to move codes to new machines

4. How much specialization is okay is subjective

Impact: OpenMP* Variants

- Maximizing PP requires good performance everywhere
  - Specialization is unavoidable
- Minimizing CD requires specialization to be simple to express:
  - Should avoid boilerplate dispatch
  - Shouldn't "pollute" remaining code

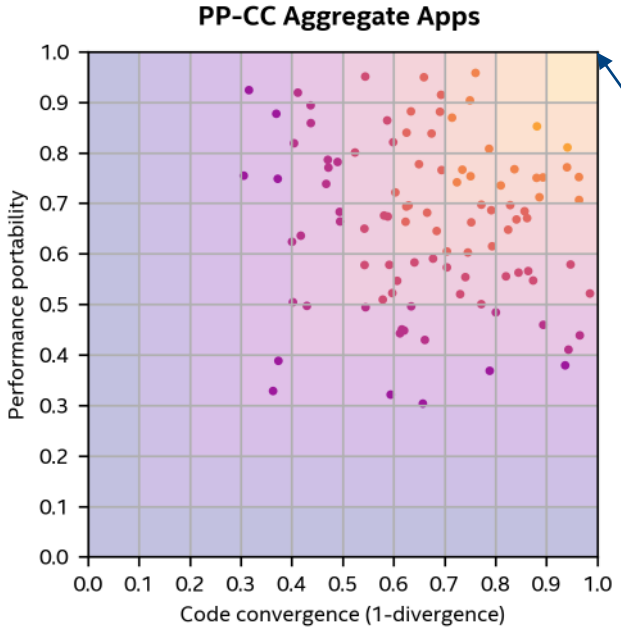
```
__m128i _mm_add(__m128i a,
                __m128i b)
{ /* Specialized code using SSE */ }

__m256i _mm256_add(__m256i a,
                   __m256i b)
{ /* Specialized code using AVX2 /* }

#pragma omp declare variant(_mm_add) \
 match(construct={simd}, arch={sse})
#pragma omp declare variant(_mm256_add) \
 match(construct={simd}, arch={avx2})
int add(int a, int b);
```

S. J. Pennycook, J. D. Sewall, A. Duran, "Supporting Function Variants in OpenMP", IWOMP 2018

Impact: oneAPI and DPC++

- Intel is tracking DPC++ compiler development using PP and CD

- Encourages questions like:
  - Is this feature supported across different platforms?
  - Do these concepts have the same interpretation across platforms?
  - Do we need to provide a library for this functionality to minimize divergence?



"Data Parallel C++ (DPC++) ... enables **high productivity and performance** across CPU, GPU, and FPGA architectures, **while permitting accelerator-specific tuning**." - http://software.intel.com/oneapi

Results shown are for illustrative purposes only, and do not reflect the current or targeted state of the DPC++ compiler.
See our other talk at P3HPC Forum 2020 for more details on combined usage of PP and CD.

Next Steps

- We're not there yet†:
  1. Is it performance portable?
  2. What performance does it achieve "on average" (over platforms/inputs)?
  3. How similar is the performance achieved on different platforms?
  4. What performance can I expect if I introduce a new platform?
  5. How difficult is it to write/maintain?

- Need more feedback and evaluations of proposed metric and tools

- Lots of interesting avenues for future research and tool development

† Many papers addressing 1 and 2, some papers addressing 3 and 5, only one or two papers addressing 4

Legal Notices and Disclaimers

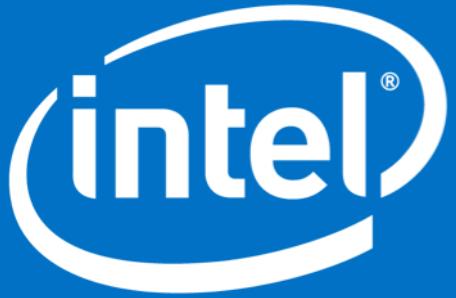
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

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 the publication date of the referenced papers and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. Configurations:
Slide 4 – Measured by NERSC; C. Yang et al., "An Empirical Roofline Methodology for Quantitatively Assessing Performance Portability", P3HPC 2018
Slide 5 – Measured by University of Amsterdam; H. Dreuning, et al., "A Beginner's Guide to Estimating and Improving Performance Portability", ISC Workshops 2018
Slide 6 – Measured by ORNL; A. Sedova et al., "High-performance Molecular Dynamics Simulation for Biological and Materials Sciences: Challenges of Performance Portability", P3HPC 2018
Slide 8 – Measured by PPCU ITK; I. Z. Reguly, "Performance Portability of Multi-Material Kernels", P3HPC 2019
Slide 9 – Measured by ITA Brazil; D. Daniel et al., "On Applying Performance Portability Metrics", P3HPC 2019
Slide 10 – Measured by University of Bristol; T. Deakin et al, "Performance Portability Across Diverse Computer Architectures", P3HPC 2019
Slide 11 – Measured by ANL; C. Bertoni et al, "Performance Portability Evaluation of OpenCL Benchmarks Across Intel and NVIDIA Platforms", IPDPSW 2020

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