SOLUTION BRIEF

Intel Connectivity
Ethernet Networking Division

Performance Testing Application Device Queues (ADQ) with Redis*

Innovative Intel® Ethernet technology improves open source Redis performance in benchmark testing.

Modern distributed data center applications rely on powerful processors, massive storage, and fast connectivity to achieve high performance and availability. Connectivity performance has traditionally been measured in terms of latency and throughput—that is, the average speed and quantity of data movement. As data centers scale, a third metric becomes increasingly important: predictability.

The Challenge of Predictability at Scale

When implementing a database application, end users typically expect a response in a given amount of time. Network architects will design the application and network system to respond within that response time. The predictability of an application's response time is typically measured in terms of jitter. Jitter refers to the *variability* of latency—the slight variation (earlier or later) in turnaround time for when a response will be received—rather than the average latency in a server. The distribution of jitter increases with scale. Thus, the tail latency, the slow outliers in an otherwise fast system, becomes an increasing factor for consideration as the data center scales with more servers.

For an analogy, consider flipping a coin. The odds of getting a heads on the first flip are 1/2 or 50 percent. As you flip more coins, the probability of getting all heads goes down exponentially: 1/2, 1/4, 1/8, 1/16, and so on. Likewise with a server-based system, the probability of receiving every response within an acceptable response time goes down as the number of requests increases. While the designed likelihood of getting a response within the desired time in a server-based system is much higher (more than 99 percent), the magnitude of trials is also much higher, with tens, hundreds, thousands, or even tens of thousands of servers in use and thousands of transaction requests. As a result, network designers must guard band by limiting the number of servers assigned to a parallelized task, or by limiting the number of end users, to stay within the desired response time.

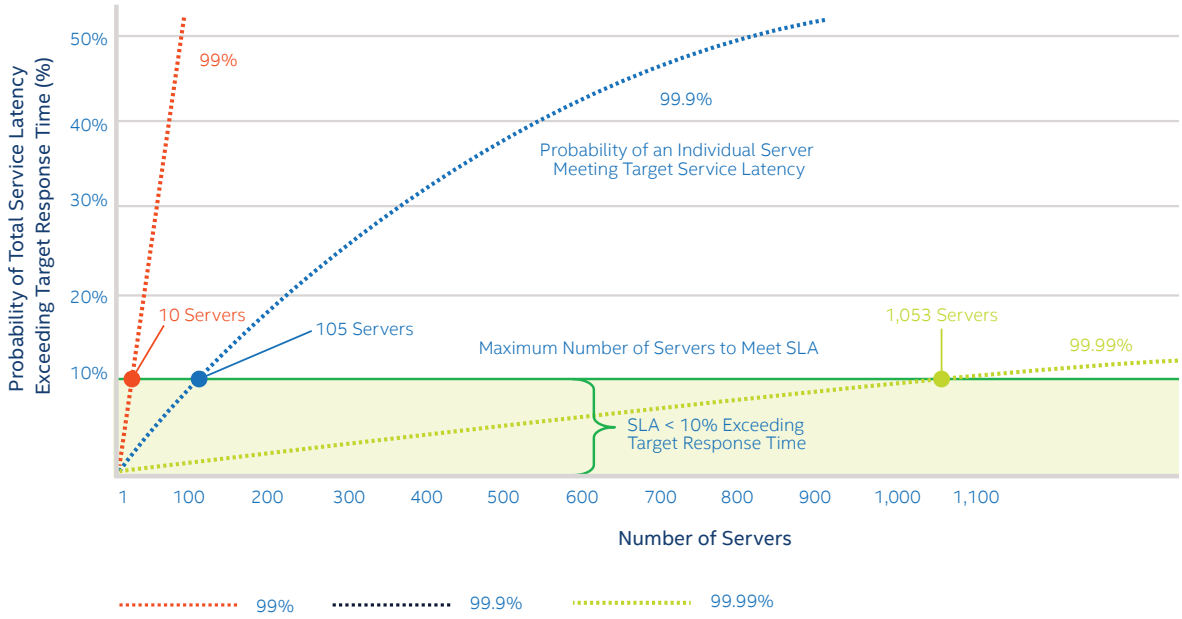

Figure 1. Higher predictability enables more servers working in parallel within the desired response time[1]

Jitter becomes a limiting factor to scalability when adding more servers, causing system latency to exceed the user-experience requirements (see Figure 1). This is why, at scale, low latency and high throughput are not enough; predictability is also required. Increasing the predictability of application response times by lowering jitter enables more compute servers to be assigned to a task and can allow more users to access the system, providing a better end-user experience. Even applications that are not large scale can benefit from higher consistency, enabling them to meet service-level agreements (SLAs).

Introducing Application Device Queues for Predictable, Scalable High Performance

Application Device Queues (ADQ) is a new Intel innovation in system-level network input/output (I/O) performance that improves application response predictability and scalability in a cost-effective manner. ADQ dedicates queues and shapes traffic for the transfer of data over Ethernet for critical applications using standard operating system networking stacks and interfaces supplemented with Intel® hardware technologies for improved performance. The goal of ADQ is to ensure that high-priority applications receive predictable high performance through dramatically reduced jitter.

Data Express Lanes

A good way to understand ADQ at a high level is by analogy to freeway traffic. Imagine you want to get from your home to the airport in time to make your flight. If you take the freeway when traffic is light, the trip takes 30 minutes; but if the traffic is heavy, it might take as long as 90 minutes. This means that, to be safe, you need to plan 90 minutes for the trip, which might waste up to an hour of your time. This is like a data-application developer who, not knowing how long it will take to receive requested data, must design around a worst-case scenario.

Figure 2. ADQ is like a dedicated express lane for your application data

Now imagine there are dedicated express lanes available on the freeway allocated only to people traveling from your location to the airport. Onramps are metered, offramps are limited, lanes are assigned, and speed is fixed so that the trip always takes 30 minutes—unaffected by any other traffic on its way to different destinations.

ADQ provides fast and predictable data-transfer performance that application developers can rely on so they, and network operators, can optimize their applications accordingly. By creating dedicated pipes between application threads and device queues, ADQ not only reduces contention for resources, but it also minimizes or eliminates synchronization operations such as locks and multi-thread sharing. Interrupts and context switching can also add traffic turmoil. ADQ uses busy polling to reduce the number of interrupts and context switches. ADQ uses a combination of these methods to improve application performance and reduce jitter.

ADQ offers application-specific, uncontended, smooth-flowing traffic, because there is no sharing of traffic from other applications on these queues. Traffic shaping reduces jitter by avoiding contention (no traffic jams), rate-limiting traffic (metered ramps), and reducing the number of interrupts and context switches per second (no lane changing).

ADQ Requirements

To take full advantage of ADQ, you'll need the latest Intel technologies and the updated Linux* kernel, as shown in Table 1.

Table 1. ADQ deployment requirements

Component	Requirement
Application	Case 1: No change (for example, Redis*) Case 2: Reference code provided by Intel
Configuration	Standard operating system tools (Traffic configuration [TC], ethtool*)
Operating System	Linux 4.19* or later
Ethernet Driver	Intel® Ethernet 800 Series driver
Ethernet NIC	Intel Ethernet 800 Series

Testing Redis with ADQ

Redis* is a popular open source in-memory data structure store used by Craigslist, GitHub, Stack Overflow, Twitter, and many others.[2] It is a representative application for testing back-end functions in both the caching and database tiers.

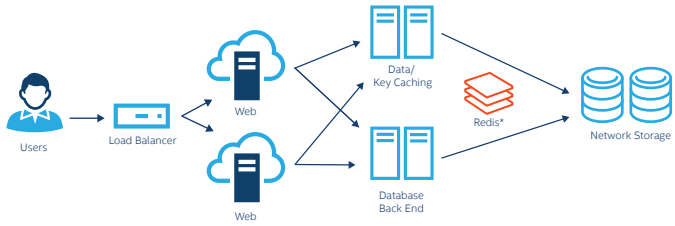

Figure 3. Redis* is a representative application for testing both key caching and a database back end

Redis scales out by deploying multiple single-threaded instances, so, by design, there is no sharing across the isolated instances. However, other elements, such as the networking stack, don't necessarily extend this instance isolation. ADQ extends instance isolation all the way down the stack to improve performance.

Test Setup

The Redis server ran on the system under test (SUT) and exposed as many listening ports as there were cores on the SUT, less 2 cores for system maintenance. The Redis server was scaled out to 94 instances composed of one master and 93 replicating read-only clones. Clients were configured with 94 instances across 11 systems. Tests were conducted at 10 different connections-per-instance rates (10–100) and a key data size of 100 bytes, each completed with and without ADQ enabled on the server. Each 30-second test run was measured for throughput (requests per second), round-trip latency (microseconds to return the request), and predictability (variation in latency over a test run).

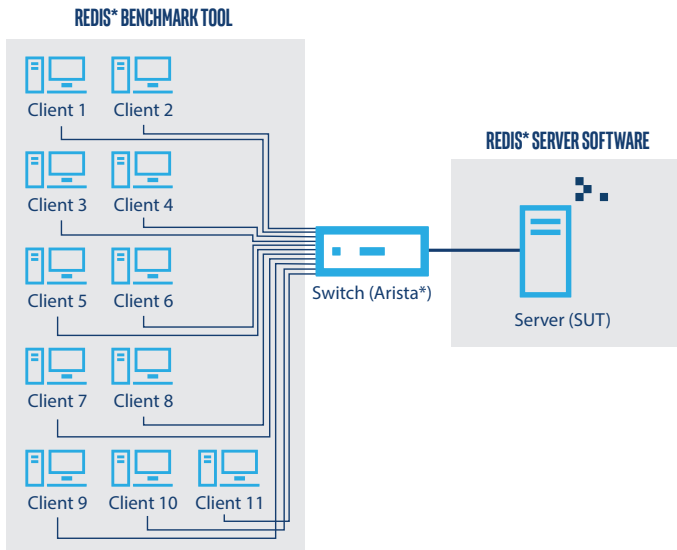

Figure 4. Redis* testing setup

Test Results

Tests were performed for predictability, latency, and throughput, with results as follows:

Predictability Results: More Than 50 Percent Increased[3]

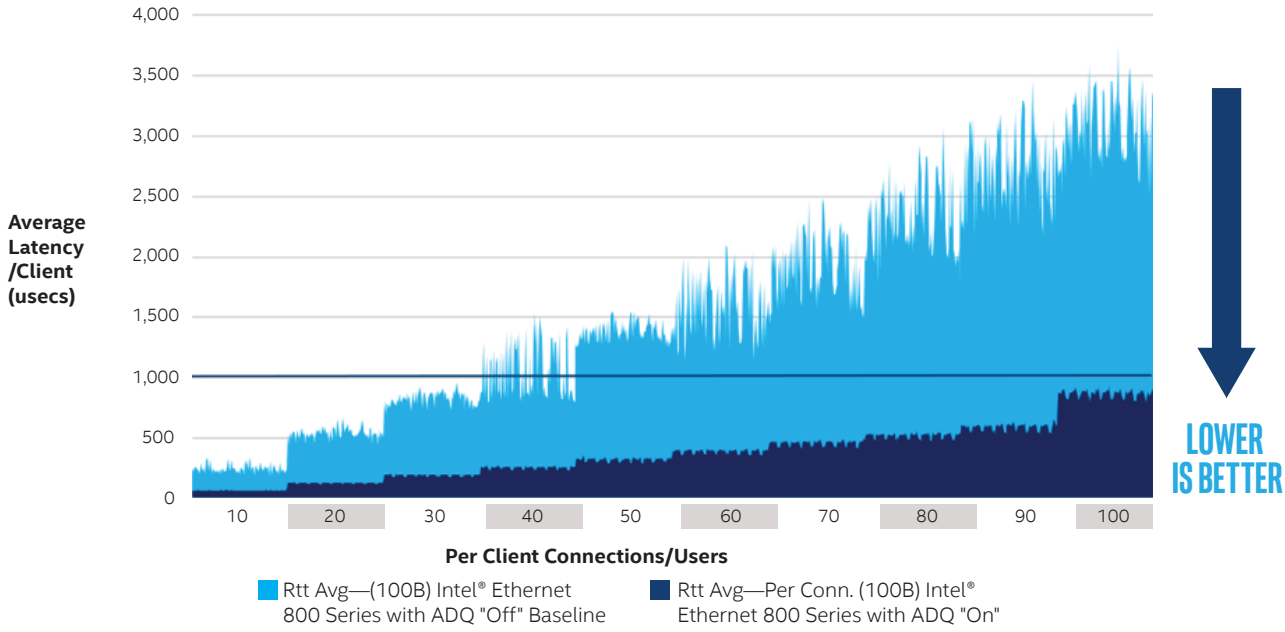

Figure 5. Predictability performance results (variation in latency)

A remarkable improvement in predictability is achieved, as shown by the ADQ test results on Redis.

Predictability in this chart is represented by the straightness of the ADQ line compared to the increasing jaggedness (jitter) of the baseline results. At lower connections per instance, the difference is noticeable; and at higher connection levels, the difference becomes dramatic. An important pattern to notice is that the ADQ line appears virtually unchanged, whereas the baseline becomes more and more jittery when the number of connections increases.

Note that these results are only for a single-server configuration. Redis can also be configured in a cluster mode, where these results would be expected to be even more pronounced.

Latency Results: More Than 45 Percent Lower[4]

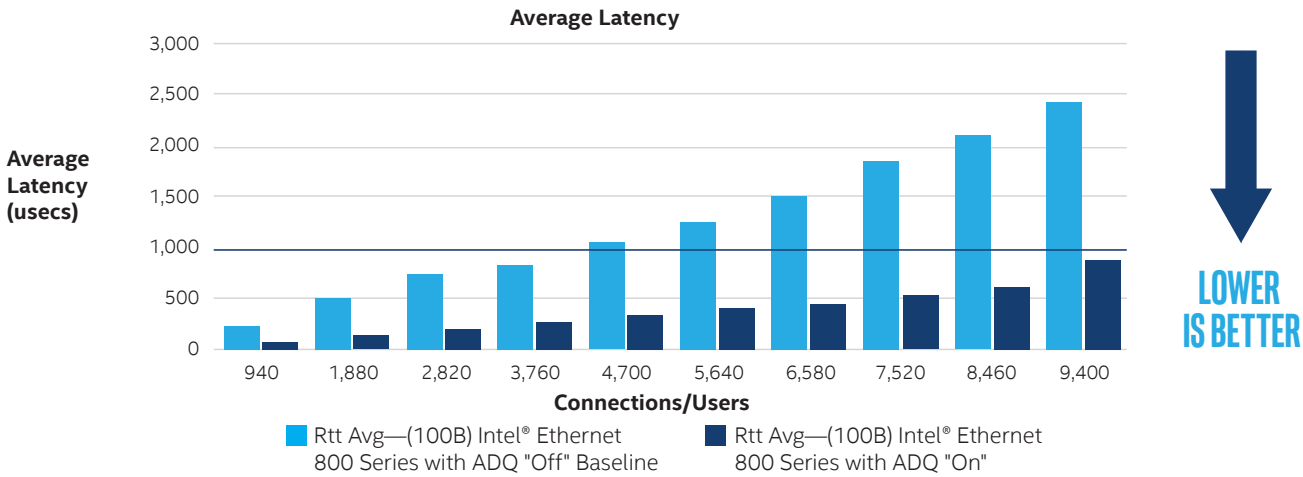

Figure 6. Latency performance results

Both the baseline and ADQ test configurations, not surprisingly, showed increasing average latency as the load increased. But the ADQ latency was lower in every case, and the rate at which the latency increased using ADQ was remarkably lower.

Throughput Results: More Than 30 Percent Improved[5]

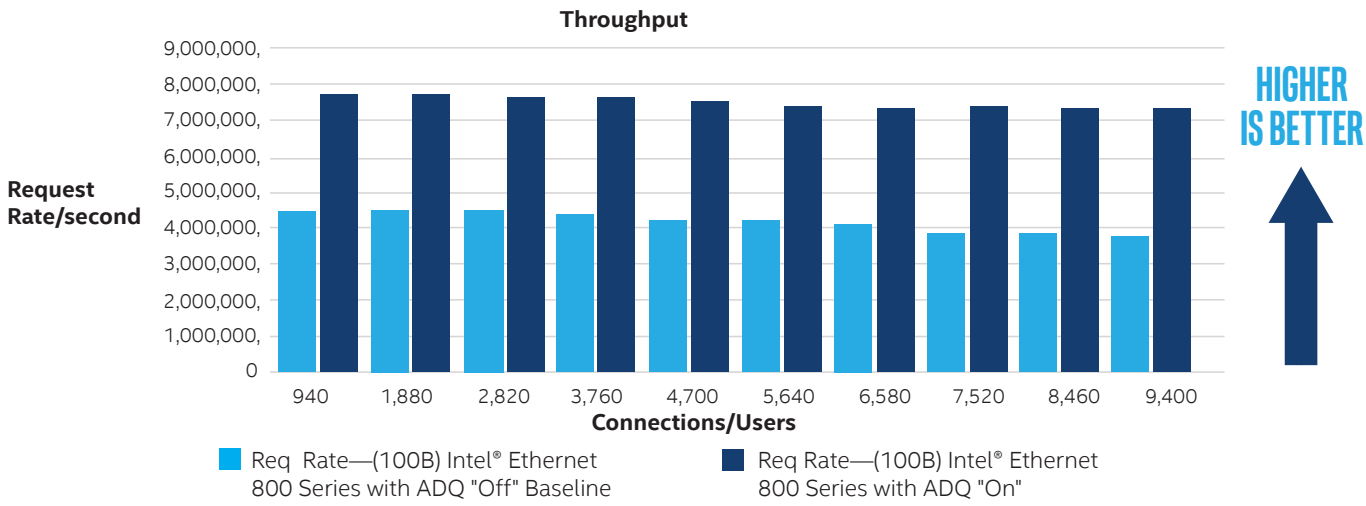

Figure 7. Throughput performance results

Final Analysis

Tests clearly show that implementing ADQ on a system running Redis in a single-server configuration increases application predictability (by reducing jitter) by more than 50 percent, reduces application latency by more than 45 percent, and improves application throughput by more than 30 percent. Throughput improvements were fairly consistent across loads, and average latency and predictability showed significantly better improvements at higher loads (connections per instance).

ADQ improves performance in a number of ways. ADQ provides improved throughput and lower latency, which will benefit most applications. The dramatic improvement in network predictability is of critical importance as data centers reach large scale, but it is also an advantage for scaling a wide range of applications because it enables more servers to be added to a compute problem or more end users to be served, providing greater consistency in meeting SLAs.

Learn More

Contact your Intel sales representative or distributor for more details about ADQ.

Table 2. Details of the test equipment and configuration

	Client	SUT
Test By	Intel	Intel
Test Date	2/11/2019	2/11/2019
Platform	Dell PowerEdge* R720	Intel® Server Board S2600WFTF
# Nodes	11	1
# Sockets	2	2
CPU	Intel® Xeon® processor E5-2697 v2 @ 2.7 GHz	2nd Generation Intel® Xeon® Platinum 8268 processor @ 2.8 GHz
Cores/Socket, Threads/Socket	12/24	24/48
ucode	0x428	0x3000009
Intel® Hyper-Threading Technology (Intel® HT Technology)	On	On
Intel® Turbo Boost Technology	On	On
BIOS Version	2.5.4	SE5C620.86B.01.00.0833.051120182255
System DDR Mem Config: slots/cap/run-speed	16 slots/128 GB/1,600 megatransfers per second (MT/s)	8 slots/128 GB/2,400 MT/s
System DCPMM Config: slots/cap/run-speed	–	2 slots/1,024 GB
Total Memory/Node (DDR+DCPMM)	128 GB	1,024 GB
Storage—boot	1x Dell* (OS Drive 512 GB)	1x Intel® SSD (OS Drive 64 GB)
Storage—application drives	–	–
NIC	1x Intel® Ethernet Converged Network Adapter X520-DA2	1x Intel® Ethernet Network Adapter E810-CQDA2
PCH	Intel® C600 Series Chipset	Intel® C620 Series Chipset
Other HW (Accelerator)		
OS	CentOS* 7.4	CentOS 7.6
Kernel	3.10.0-693.21.1.el7	4.19.18 (Linux.org Stable)
IBRS (0=disable, 1=enable)	0	1
eIBRS (0=disable, 1=enable)	0	0
Retpoline (0=disable, 1=enable)	0	1
IBPB (0=disable, 1=enable)	0	1
PTI (0=disable, 1=enable)	1	1
Mitigation Variants (1,2,3,3a,4, L1TF)	1,2,3,L1TF	1,2,3,L1TF
Workload & Version	Redis-benchmark 4.0.10	Redis 4.0.10
Compiler	–	gcc (GCC) 4.8.5 20150623
NIC Driver	Intel® Ethernet 500 Series ixgbe 4.4.0-k-rh7.4	Intel® Ethernet 800 Series ice Alpha 0.8.15
Redis* Configuration	Redis-benchmark v4.0.10 + Enhanced reporting patch 94 instances spread across 11 systems Connections per instance: 10 to 100 Key range: 10000 Duration: 30 seconds GET key distribution: random Data size: 100 bytes	Redis-Server: v4.0.10 malloc=jemalloc-4.0.3 bits=64 build=43de2cb93108516d 94 Redis instances in a HA cluster with one master. Memory store only, no DB persistance, no evictions Connections per instance: 10 to 100 Transaction rate: Unlimited nTuple configured for each instance ADQ enabled for Intel® Ethernet Network Adapter E810-CQDA2
Network Switch: Arista 7060CX	Connected at 10G	Connected at 100G

[1] Jeffrey Dean and Luiz Andrée Barroso. "The Tail at Scale." Communications of the ACM. February 2013. https://cseweb.ucsd.edu/~gmporter/classes/fa17/cse124/post/schedule/p74-dean.pdf.

[2] Redis. "Who's using Redis?" January 2019. https://redis.io/topics/whos-using-redis.

[3] >50% predictability improvement with open source Redis* using 2nd Gen Intel® Xeon® Scalable processors and Intel® Ethernet 800 Series with ADQ vs. without ADQ. Source: Intel internal testing as of February 2019; open source Redis on 2nd Gen Intel® Xeon® Scalable processors and Intel® Ethernet 800 Series, 100 GbE on Linux* 4.19.18 kernel (see Table 2).
Calculation: (new - old) / old x 100% for reduction in variance of Standard Deviation of Rtt Average Latency across all runs (10 to 100) for baseline vs ADQ.
(229-739)/739 * 100% = -69% Reduction in Variance.

[4] >45% latency reduction with open source Redis* using 2nd Gen Intel® Xeon® Scalable processors and Intel® Ethernet 800 Series with ADQ vs. without ADQ. Source: Intel internal testing as of February 2019; open source Redis on 2nd Gen Intel® Xeon® Scalable processors and Intel® Ethernet 800 Series, 100 GbE on Linux* 4.19.18 kernel (see Table 2).
Calculation: (new - old) / old x 100% for Rtt Average Latency across all runs for baseline vs ADQ (382-1249)/1249 * 100% = -69% Reduction in Rtt Average Latency.

[5] >30% throughput improvement with open source Redis* using 2nd Gen Intel® Xeon® Scalable processors and Intel® Ethernet 800 Series with ADQ vs. without ADQ. Source: Intel internal testing as of February 2019; open source Redis on 2nd Gen Intel® Xeon® Scalable processors and Intel® Ethernet 800 Series, 100 GbE on Linux* 4.19.18 kernel (see Table 2).
Calculation: (new - old) / old x 100% for Average Transaction Request Rate across all runs for baseline vs ADQ (79601-44345)/44345 * 100% = 80% Throughput Improvement.

Performance results are based on testing as of February 2019 and may not reflect all publicly available security updates. See configuration disclosure for details.
No product can be absolutely secure.

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 intel.com/benchmarks.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Cost reduction scenarios described are intended as examples of how a given Intel- based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

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

Printed in USA 0319/TK/PRW/PDF Please Recycle 338670-001US