Knights Landing: Second-Generation Intel Xeon Phi Product

The Knights Landing processor targets high-performance computing and other parallel workloads. It provides significantly higher performance and memory bandwidth over the previous generation. It is a standard, self-booting CPU that is binary compatible with prior Intel processors. Its innovations include a power-efficient core, a 512-bit vector instruction set, a new memory architecture, a high-bandwidth on-die interconnect, and an on-package network fabric.

Avinash Sodani
Roger Gramunt
Jesus Corbal
Ho-Seop Kim
Krishna Vinod
Sundaram Chinthamani
Steven Hutsell
Rajat Agarwal
Yen-Chen Liu
Intel

Knights Landing (KNL) is the code name for the second-generation Intel Xeon Phi product family, which targets high-performance computing (HPC) and other parallel computing segments. It is a brand-new many-core architecture that delivers massive thread parallelism, data parallelism, and memory bandwidth in a CPU form factor for high-throughput workloads. KNL brings many innovations to market. It is a standard, stand-alone processor that can boot an off-the-shelf operating system. This is a big change from the previous-generation Intel Xeon Phi co-processor, Knights Corner (KNC),[1] which was a PCI Express- (PCIe-)connected coprocessor. In addition, KNL is binary compatible with prior Intel Xeon processors, which ensures that any legacy software will run on KNL. Furthermore, KNL improves on scalar and vector performance over KNC by about three times, packing more than 3 Teraflops of double-precision and 6 Teraflops of single-precision peak floating-point performance in the chip. It also introduces an innovative memory architecture comprising two types of memory, which provides both the high bandwidth and large capacity needed to run large HPC workloads. Finally, KNL integrates the Intel Omni-Path Fabric on the package to provide a more scalable and cost-effective fabric solution for large systems.

KNL will be available in three product varieties: a KNL self-booting processor, KNL self-booting with integrated fabric, and a KNL PCIe-connected coprocessor card as a follow-on product to KNC. The primary focus of this article will be the KNL self-booting processor, which is the main product and the baseline for the other two products.

KNL Architecture Overview

We begin by providing an overview of the KNL architecture. Figure 1a shows the KNL CPU's block diagram, and Figure 2 shows its die photo. The KNL CPU comprises 38 physical tiles, of which at most 36 are active; the remaining two tiles are for yield recovery. Each tile (see Figure 1b) comprises two cores, two vector processing units (VPUs) per core, and a 1-Mbyte level-2 (L2) cache that is shared between the two cores. The core is a new

Published by the IEEE Computer Society

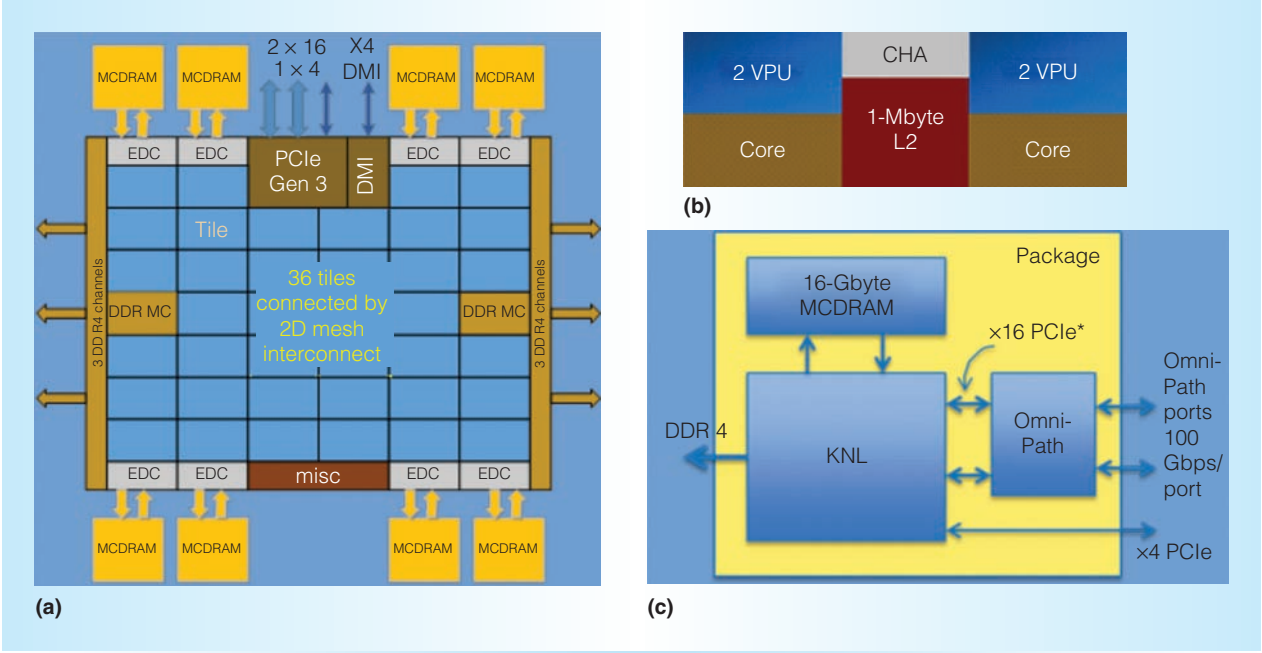

Figure 1. Knights Landing (KNL) block diagram: (a) the CPU, (b) an example tile, and (c) KNL with Omni-Path Fabric integrated on the CPU package. (CHA: caching/home agent; DDRMC: DDR memory controller; DMI: Direct Media Interface; EDC: MCDRAM controllers; MCDRAM: multichannel DRAM; PCIe: PCI Express; VPU: vector processing unit.)

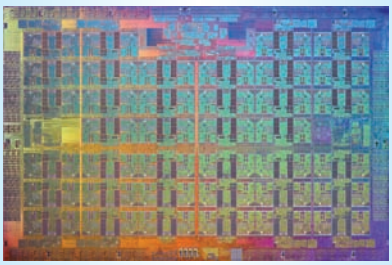

Figure 2. KNL CPU die photo.

two-wide, out-of-order core that is derived from the Intel Atom core (based on Silvermont microarchitecture).[2] It includes heavy modifications over the inherited microarchitecture to incorporate features necessary for HPC workloads, such as four threads per core, deeper out-of-order buffers, higher cache bandwidth, new instructions, better reliability, larger translation look-aside buffers (TLBs), and larger caches. KNL introduces the new Advanced Vector Extensions instruction set, AVX-512, which provides 512-bit-wide vector instructions and more vector registers.[3] In addition, it continues to support all legacy x86 instructions, making it completely binary-compatible with prior Intel processors.

KNL introduces a new 2D, cache-coherent mesh interconnect that connects the tiles, memory controllers, I/O controllers, and other agents on the chip. The mesh interconnect provides the high-bandwidth pathways necessary to deliver the huge amount of memory bandwidth provisioned on the chip to the different parts of the chip. The mesh supports the MESIF (modified, exclusive, shared, invalid, forward) cache-coherent protocol.[4] It employs a distributed tag directory to keep the L2 caches in all tiles coherent with each other. Each tile contains a caching/home agent that holds a portion of the distributed tag directory and also serves as a connection point between the tile and the mesh.

KNL has two types of memory: multichannel DRAM (MCDRAM) and double data rate (DDR) memory. MCDRAM is the 16-Gbyte high-bandwidth memory comprising eight devices (2 Gbytes each) integrated on-package and connected to the KNL die via a proprietary on-package I/O. All eight MCDRAM devices together provide an aggregate Stream triad benchmark bandwidth of more than 450 Gbytes per second (GBps; see

www.cs.virginia.edu/stream). KNL has six DDR4 channels running up to 2,400 MHz, with three channels on each of two memory controllers, providing an aggregate bandwidth of more than 90 GBps. Each channel can support at most one memory DIMM. The total DDR memory capacity supported is up to 384 Gbytes. The two types of memory are presented to users in three memory modes: *cache mode,* in which MCDRAM is a cache for DDR; *flat mode,* in which MCDRAM is treated like standard memory in the same address space as DDR; and *hybrid mode,* in which a portion of MCDRAM is cache and the remainder is flat.

KNL supports a total of 36 lanes of PCIe Gen3 for I/O, split into two x16 lanes and one x4 lane. It also has four lanes of proprietary Direct Media Interface to connect to the Southbridge chip, just like Intel Xeon processors. The Southbridge chip provides support for legacy features necessary for a self-booting system.

KNL integrates the Intel Omni-Path Fabric[5] on-package in one of the product incarnations (see Figure 1c). The Omni-Path Fabric is connected via the two x16 lanes of PCIe to the KNL die and provides two 100-Gbits-per-second ports out of the package. The typical power (thermal design power) for KNL (including MCDRAM memory) when running a computationally intensive workload is 215 W without the fabric and 230 W with the fabric.

Vision and Motivation

In this section, we provide the motivations behind some of the choices made in the KNL architecture. The most fundamental choice we made was to make it a self-booting, standard CPU. The primary motivation was to get away from the PCIe bottlenecks inherent in a coprocessor and to provide the computation and bandwidth necessary for HPC workloads in a standard CPU form factor. Once this choice was made, several other choices followed. With KNL as a standard CPU, it is expected to run all kinds of software—not just the highly parallel code—such as standard operating systems, system infrastructure code, general libraries, and debug tools. This motivated binary compatibility of KNL with existing Intel processors, because requiring that every piece of support software be recompiled to run on KNL would be a big overhead for users. Being a standard CPU also implied that KNL would run entire applications. This motivated us to develop a new core with significantly higher scalar performance than KNC to ensure KNL also ran serial portions of the applications well.

Binary compatibility motivated the need to define 512-bit vector extensions in a manner consistent with existing AVX and AVX2 instruction sets. This resulted in the introduction of AVX512 instructions.

We designed KNL's memory architecture to support its large computational capability and its stand-alone processor status. We chose MCDRAM memory to provide the high bandwidth needed to feed the large computational capability, and we chose DDR to provide the large memory capacity needed to run an entire application with all the necessary support software on KNL. We did not include a shared on-die L3 cache, because we found that the targeted HPC workloads benefited less from it compared to adding more cores; many HPC workloads either could be blocked in each per-tile L2 or were too large to fit in any reasonably sized on-die L3 cache.

Finally, the mesh interconnect had two motivations: the need to provide a network that could easily support the huge memory bandwidth provisioned on the chip, and the need to provide lower latency connections—that is, fewer average hops to go from one point to another—between different agents on the large chip.

Tile Architecture

We now provide more details on the main components within a tile: the core, VPU, and L2 cache.

Core and VPU Architecture

The KNL core differs from the parent Intel Atom cores (based on the Silvermont microarchitecture) because of the HPC modifications. Figure 3 shows the block diagram of the core and the VPU, which is divided roughly into five units: the front-end unit (FEU), the allocation unit, the integer execution unit

(IEU), the memory execution unit (MEU), and the VPU. Although the core is two-wide from the point of view of decoding, allocating, and retiring operations, it can execute up to six operations per cycle in the middle.

Front-end unit. The core's FEU comprises a 32-Kbyte instruction cache (IL1) and a 48-entry instruction TLB. In case of a hit, the instruction cache can deliver up to 16 bytes per cycle. These bytes are then sent to a two-wide decoder. Most instructions are decoded into a single micro-op, but a few complex ones that produce more micro-ops are handled by a two-wide microsequencer engine. Fetch direction is provided by a gskew-style branch predictor. Decoded micro-ops are placed into a 32-entry instruction queue.

Allocation unit. The allocation unit reads two micro-ops per cycle from the instruction queue. It assigns the necessary pipeline resources required by the micro-ops, such as reorder buffer (ROB) entries (72), rename buffer entries (72), store data buffers (16), gather-scatter table entries (4), and reservation station entries. It also renames the register sources and destinations in the micro-ops. The rename buffer stores the results of the in-flight micro-ops until they retire, at which point the results are transferred to the architectural register file. After the allocation unit, the micro-ops are sent to one of three execution units—IEU, MEU, or VPU—depending on their opcode types. Some micro-ops could get sent to more than one execution unit. For example, an Add instruction that has a memory source will be sent to the MEU to read the memory source, and then to the IEU to execute the Add operation.

Integer execution unit. The IEU executes integer micro-ops, which were defined as those that operate on general-purpose registers R0 through R15. There are two IEUs in the core. Each IEU contains one 12-entry reservation station that issues one micro-op per cycle. The integer reservation station is fully out-of-order in its scheduling. Most operations have one-cycle latency and are supported by both IEUs. But a few have three- or five-cycle latency (for example, "multiplies") and are only supported by one of the IEUs.

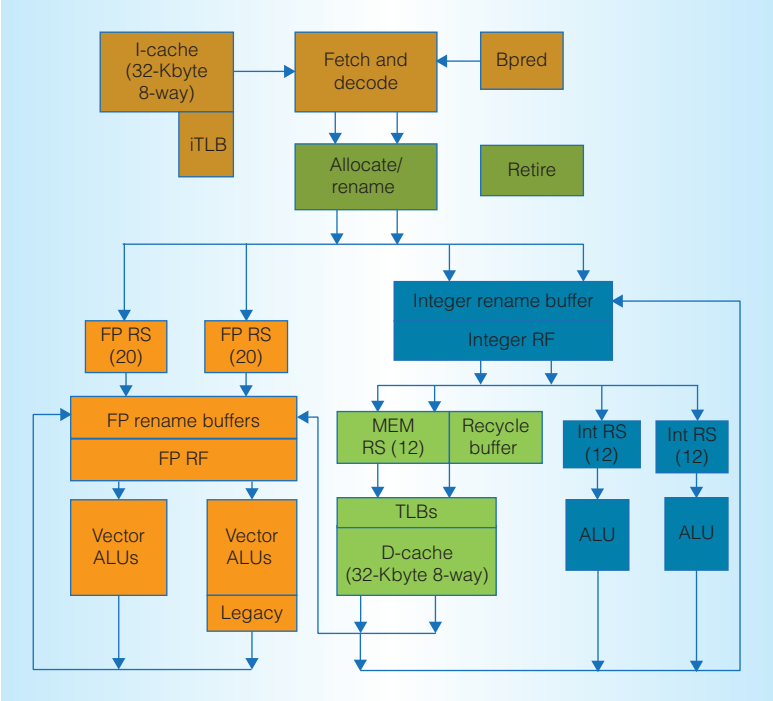

Figure 3. Core and VPU dataflow block diagram showing important microarchitectural structures and data bus connections between them. (ALU: arithmetic logic unit; iTLB: instruction translation look-aside buffer; RF: register file; RS: reservation station.)

Memory execution unit. The MEU executes memory micro-ops and also services fetch requests for instruction cache misses and instruction TLB misses.

Up to two memory operations, either load or store, can be executed in the MEU in a given cycle. Memory operations are issued in-order from the 12-entry memory reservation station, but they can execute and complete out of order. Micro-ops that do not complete successfully are allocated into the recycle buffer and reissued to the MEU pipeline once their conflict conditions are resolved. Completed loads are kept in the memory ordering queue until they are retired to maintain consistency. While stores are kept in the store buffer after address translation, they can forward data to dependent loads. Stores are committed to memory in the program order, one per cycle.

The 64-entry, eight-way set-associative L1 micro TLB is backed up by the 256-entry, eight-way set-associative L2 data TLB. The

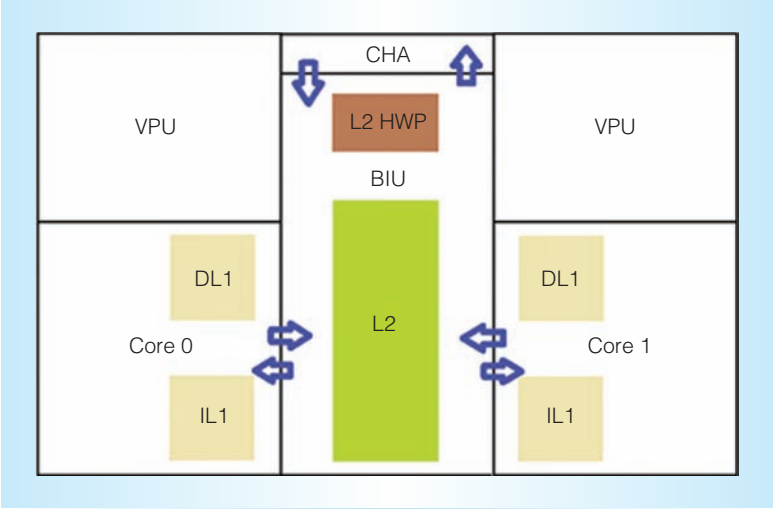

Figure 4. Cache hierarchy inside a tile. The figure shows the level-1 instruction (IL1) and data (DL1) caches in each core and the level-2 (L2) cache with its hardware prefetcher (HWP) between the two cores. (BIU: bus interface unit.)

data TLB also contains an eight-way, 128-entry table for 2-Mbyte pages and a fully associative, 16-entry table for 1-Gbyte pages. The writeback, nonblocking, 32-Kbyte, eight-way set-associative L1 data cache (DL1) supports two simultaneous 512-bit reads and one 512-bit write, with a load-to-use latency of four cycles for integer loads and five cycles for floating-point loads. The L1 hardware prefetcher monitors memory address patterns and generates data prefetch requests to the L2 cache to bring cache lines in advance.

The MEU supports unaligned memory accesses without any penalties and supports accesses that split into two cache lines with a two-cycle penalty. This helps HPC workloads where memory accesses might not always be aligned to natural data size. In addition, the MEU supports 48 virtual and 46 physical address bits (up to 39 physical bits for addressing cacheable memory and the rest for uncacheable memory) to provide the large memory addressing capability commensurate with a stand-alone, standard CPU.

Finally, the MEU contains specialized logic to handle gather-scatter instructions efficiently. A single gather-scatter instruction can access many memory locations. This generation of multiple accesses is done very close to the L1 cache pipeline. This allows maximum utilization of the two memory pipelines while consuming minimal resources in rest of the core (such as the FEU, allocation unit, reservation stations, and reorder buffer).

Vector processing unit. The VPU is KNL's vector and floating-point execution unit and provides support for x87, MMX, Streaming SIMD Extensions (SSE), AVX, and AVX-512 instructions, as well as integer divides. Two VPUs are connected to the core. They are tightly integrated into the pipeline, with the allocation unit dispatching instructions directly into the VPUs. The VPUs are mostly symmetrical, and each can provide a steady-state throughput of one AVX-512 instruction per cycle, providing a peak of 64 single-precision or 32 double-precision floating-point operations per cycle from both VPUs. One of the VPUs is extended to provide support for the legacy floating-point instructions, such as x87, MMX, and a subset of byte and word SSE instructions.

Each VPU contains a 20-entry floating-point reservation station that issues one micro-op per cycle out of order. The floating-point reservation station differs from the IEU and MEU reservation stations in that it does not hold source data to help reduce its size; the floating-point micro-ops read their source data from the floating-point rename buffer and the floating-point register file after they issue from the floating-point reservation station, spending an extra cycle between the reservation station and execution compared to integer and memory micro-ops. Most floating-point arithmetic operations have a latency of six cycles, whereas the rest of the operations have a latency of two or three cycles, depending on the operation type.

The VPU also supports the new transcendental and reciprocal instructions, AVX-512ER,[3] and the vector conflict detection instructions, AVX-512CD,[3] introduced in KNL.

L2 Architecture

On a KNL tile, the two cores share a 16-way associative, 1-Mbyte unified L2 cache. The bus interface unit (BIU) maintains intra-tile coherency (see Figure 4) and also acts as the local shared L2 cache management unit. Lines in the L2 cache are maintained in one of the MESIF states. Requests are made by

the cores to the BIU via a dedicated request interface to each core. Cacheable requests look up the L2 tags for hit, miss, and line-state evaluation, whereas other requests are bypassed directly out to the mesh to be serviced by the targeted caching/home agent.

KNL implements a unique cache topology to minimize coherency maintenance traffic. First, the L2 cache includes DL1 but not IL1. The lines brought into IL1 fill in the L2 cache, but when those lines are evicted, the corresponding IL1 line is not invalidated. This avoids invalidations due to hot-IL1/cold-L2 scenarios, in which a line in active use in IL1 gets invalidated due to eviction of the corresponding line from L2 cache due to inactivity. Second, the L2 cache stores "presence" bits per line to track which of them are actively used in DL1. This information is used to filter the coherency probes for inclusive DL1 when not required. It also factors into the L2 victim-selection algorithm to minimize eviction of in-use lines.

The BIU also contains an L2 hardware prefetcher that is trained on requests coming from the cores. It supports up to 48 independent prefetch streams. Once a stream is detected to be stable either forward or backward, prefetch requests are issued to successive cache lines in that stream (that is, the unit cache line stride distance).

Threading

A KNL core supports up to four hardware contexts or threads using hyperthreading techniques.[6] Core resources can be dynamically partitioned, shared, or replicated, and pipelines are regulated by thread selectors. The goal is to maximize resource utilization among active threads.

In general terms, threads become inactive after they execute a halt or monitor wait (mwait) instruction[6]; otherwise, they are considered to be active. Three threading modes are defined depending on the number of active threads: single-thread mode when only one thread is active (any one), dual-thread mode when any two threads are active, and quad-thread mode when any three or all four threads are active.

Dynamically partitioned resources are equally distributed among threads, depending on the thread mode. For example, in single-thread mode, the active thread is allowed to use the full 72 entries in the ROB; in dual-thread mode, each thread will get 36 entries; and in quad-thread mode, each active thread will get 18 entries. Besides the ROB, other noteworthy structures that are also dynamically partitioned are the rename buffers, reservation stations, store data buffers, instruction queue, and gather-scatter table entries.

Shared resources do not enforce partitioning, although some structures have a small number of per-thread reserved entries to avoid deadlocks. Threads get shared resources on a first-come, first-served basis. Noteworthy shared resources are caches, TLBs, most MEU structures, branch predictors, and hardware prefetcher tables. Except for caches (in which lines are thread unaware), a particular entry in a shared structure is owned by only a single thread.

Replicated structures are limited to the bare minimum. In replicated structures, each thread has its dedicated structure that is not taken away when the thread becomes inactive. These structures include rename tables, architectural registers, and other control registers.

The core pipeline has thread selectors at several points to maximize utilization and throughput of the pipeline and maintain fairness among threads. These exist mainly in the in-order portion of the pipeline. The thread selector considers the availability of resources down the pipeline and tries to make informed selections. The important thread selectors are located in the FEU, allocation unit, retire pipeline, and MEU reservation stations. The out-of-order parts of the machine are thread agnostic and pick instructions based on readiness.

Instruction-Set Architecture

Figure 5 shows how KNL's instruction-set architecture compares to those of recent Intel Xeon processors.[6] KNL supports all legacy instructions, including 128-bit SSE and SSE4.2 and 256-bit AVX and AVX2 technologies.

KNL introduces Intel AVX-512 instructions. AVX-512 provides 512-bit SIMD support, 32 logical registers, native support for true vector predication via eight new mask registers, and a generalization of indirect vector accesses via gathers (for loading sparse

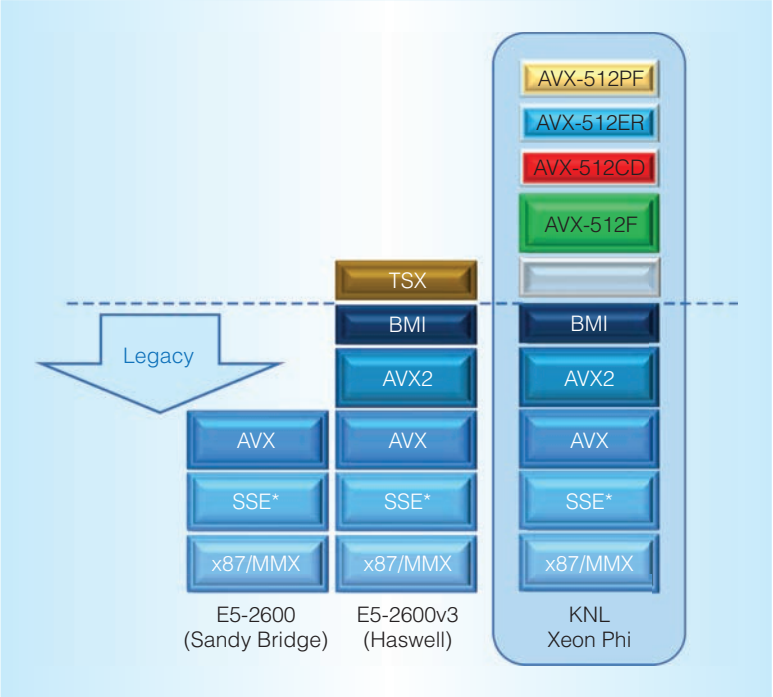

Figure 5. KNL's instruction-set architecture as compared to the architectures of prior Xeon processors. It shows that KNL supports all legacy instructions present on Xeon processors while adding new instructions. (AVX: Advanced Vector Extensions; BMI: Bit Manipulation Instructions; SSE: Streaming SIMD Extensions; TSX: Transactional Synchronization Extensions.)

data) and scatters (for storing sparse data). A single AVX-512 instruction can pack eight double-precision multiply-add operations (16 flops) or 16 single-precision multiply-add operations (32 flops), with an optional memory location used as an operand that can either be a vector or a scalar.

Intel AVX-512 instructions fall into four categories: foundation instructions (AVX-512F), which are the base 512-bit extensions described earlier; conflict-detection instructions (AVX-CD); exponential and reciprocal instructions (AVX-ER); and prefetch instructions (AVX-PF). These capabilities provide efficient conflict detection to allow more loops (such as histogram updates) to be vectorized, fast exponential and reciprocal operations to speed up transcendental functions, and new sparse prefetch capabilities, respectively.

We defined AVX-512 with programmability in mind. Most AVX-512 programming occurs in high-level languages—such as C/C++ and Fortran, with the help of vectorizing compilers and pragmas to guide the compilers—or in libraries with optimized instruction sequences, which could use intrinsics.

KNL does not implement Intel's Transactional Synchronization Extensions. The software is expected to confirm hardware support using a CPUID feature bit to ensure that it will run on machines with and without these extensions.

Mesh Architecture

The Intel mesh on-die interconnect architecture (see Figure 6a) is based on the Ring architecture, which has been widely deployed in the Intel Xeon processor family. The mesh features four parallel networks, each of which delivers different types of packets (for example, commands, data, and responses) and is highly optimized for the KNL traffic flows and protocols. The mesh can deliver greater than 700 GBps of total aggregate bandwidth.

The mesh is rows and columns of half rings, which fold upon themselves at the endpoints. The mesh enforces a YX routing rule, which means a transaction always travels vertically first until it hits the target row, makes a turn, and then travels horizontally until it reaches its destination. Messages arbitrate with the existing traffic on the mesh at injection points as well as when making a turn, with the existing traffic on the mesh taking higher priority. The static YX routing helps reduce deadlock cases and thereby simplifies the protocol. One hop on mesh takes one clock in the Y direction and two clocks in the X direction.

Cache Coherency and Distributed Tag Directory

The entire KNL chip is cache coherent and implemented with a MESIF protocol. The introduction of the F (forward) state is to allow efficient sharing. The core with the F state will be able to provide shared data directly to the requesting core without going to memory. A distributed tag directory serves as a snoop filter that tracks lines that are currently owned or shared by the cores. It is distributed among all the tiles. The directory in each tile owns a portion of the address space based on an address hash. Figure 6 shows a simple L2 cache miss flow.

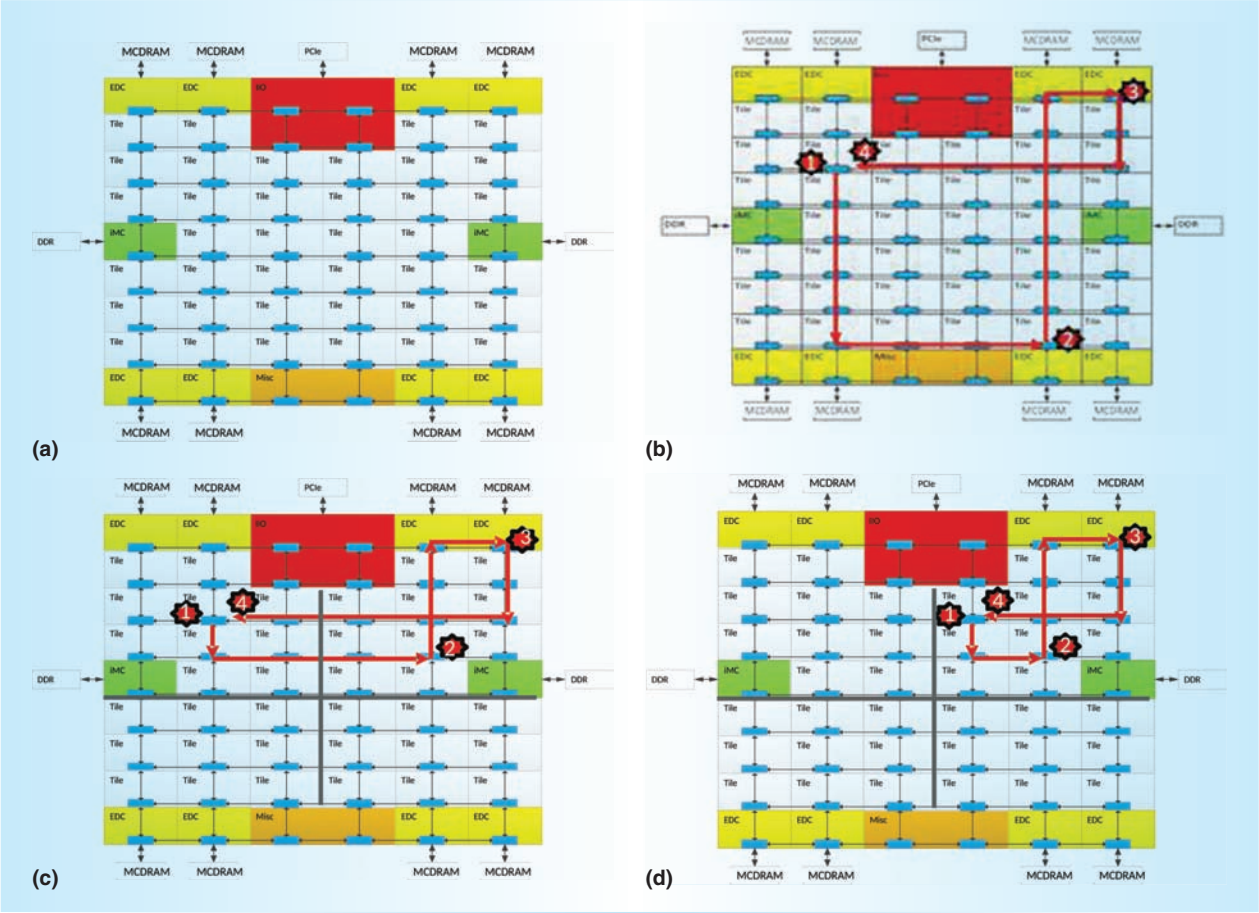

Figure 6. Example of an L2 miss flow: (1) tile with L2 miss encountered, (2) request sent to the tile with tag directory, (3) request forwarded to memory after miss in tag directory, (4) data read from memory and sent to the requester. (a) Mesh architecture, (b) all-to-all cluster mode, (c) quadrant cluster mode, and (d) sub-NUMA (nonuniform memory access) clustering (SNC) cluster mode.

Cluster Modes

The Intel mesh supports three modes of clustered operation that provide different levels of address affinity to improve overall performance. These cluster modes aim to lower latencies and improve bandwidth by lowering the distance that protocol flows traverse on the chip. These modes are selectable from the basic I/O system (BIOS) at boot time.

All-to-all mode. This mode lacks any affinity between the tile, directory, and memory. It is the most general mode and has no specific requirement for the software for memory configurations, but it typically has lower performance than the other cluster modes. Figure 6b shows how an L2 miss can traverse the mesh to fill a cache line from memory for the all-to-all cluster mode.

Quadrant mode. This mode divides the KNL chip into four virtual quadrants, each of which provides affinity between the directory and the memory. There is no affinity between a tile and the directory or the memory—that is, a request from any tile can land on any directory. However, the directory will access only the memory in its own quadrant. This cluster mode requires symmetric memory (that is, the same total capacity on both DDR memory controllers). It provides better latency than the all-to-all mode and is transparent to software support. Figure 6c shows how L2 miss transactions flow in the quadrant mode.

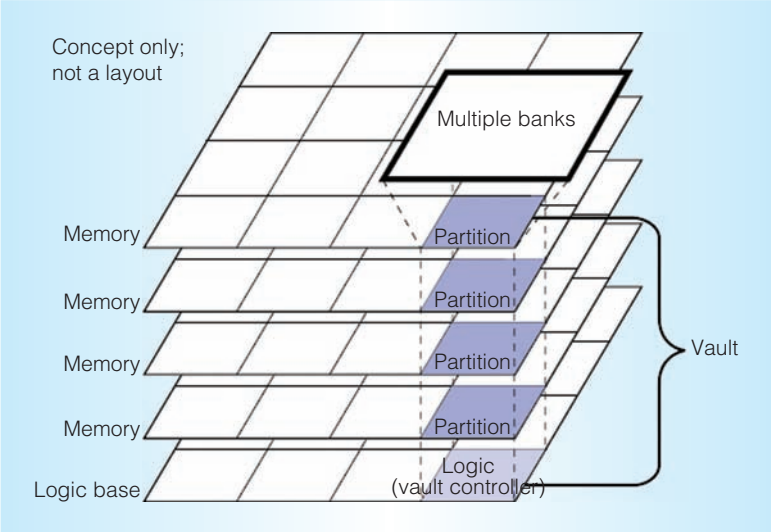

Figure 7. MCDRAM architecture. It shows a conceptual picture of DRAM dies stacked over a logic die and their organization.

Sub-NUMA clustering (SNC). This mode extends the quadrant mode further by affinitizing the tile with the directory and the memory. In this mode, the KNL chip is divided and exposed as two or four nonuniform memory access (NUMA) domains (or clusters) to the OS. For NUMA-optimized software (see http://man7.org/linux/man-pages/man7/numa.7.html), there will be an affinity between the tile, directory, and memory—that is, a request from a tile will access a directory in its cluster and, in turn, the directory will access the memory controllers that are also within that cluster (the MCDRAM controllers are labeled as EDC, and the DDR controllers are labeled as MC in Figure 6). This cluster mode has the best latency profile among all modes—especially under loaded operations—because most traffic will be contained within the local cluster. For software to exploit this mode's performance, it must be NUMA optimized—that is, it needs to allocate memory in the same NUMA cluster where it runs. Figure 6d shows the L2 miss transactions in SNC mode.

Memory Architecture

KNL supports two levels of memory: MCDRAM and DDR. MCDRAM is a stacked DRAM architecture (see Figure 7) comprising multiple channels or vaults on each DRAM die that are vertically connected with through-silicon vias.[7] Note that this is a conceptual picture and not an actual layout. Each DRAM die in the stack is partitioned with multiple banks in each partition. Corresponding partitions from all DRAM dies when taken together form a channel or vault. Each channel has its own set of command, address, and data pins. All of the channels can be accessed in parallel and provide high bandwidth for HPC applications. For each four-high stack, there are a total of 16 internal channels, with eight banks per channel, for a total of 128 banks. The large number of banks allows more parallelism. The page size is 2,048 bytes.

The DRAM dies are stacked on top of a logic buffer die that connects to the CPU through a proprietary on-package I/O, which runs at a higher frequency than that of the DRAM core, allowing for a narrower but much faster interface to the CPU. MCDRAM memory supports self-refresh and power-down modes.

Memory Modes

KNL features three primary memory modes: cache, flat, and hybrid (see Figure 8). These modes are selectable through the BIOS at boot time.

Cache mode. In this mode, MCDRAM is configured as a memory-side cache for the entire DDR memory. It is a direct-mapped cache with 64-byte cache lines. The tags are stored in the error-correcting code bits corresponding to the cache lines. The tags are read at the same time as the cache lines from the MCDRAM, allowing determination of hit or miss without requiring additional tag accesses from the MCDRAM. Because MCDRAM is much larger than traditional caches, there is no significant change in conflict misses due to the direct-mapped policy in most cases.

In this mode, all requests first go to MCDRAM for a cache lookup and then, in case of a miss, are sent to the DDR. The state of a line in the MCDRAM cache uses a modified, exclusive, garbage, invalid (MEGI) protocol. A garbage state means the line has been reserved for a future write that is guaranteed, but the current data is not valid.

Cache mode does not require any software change and works well for many

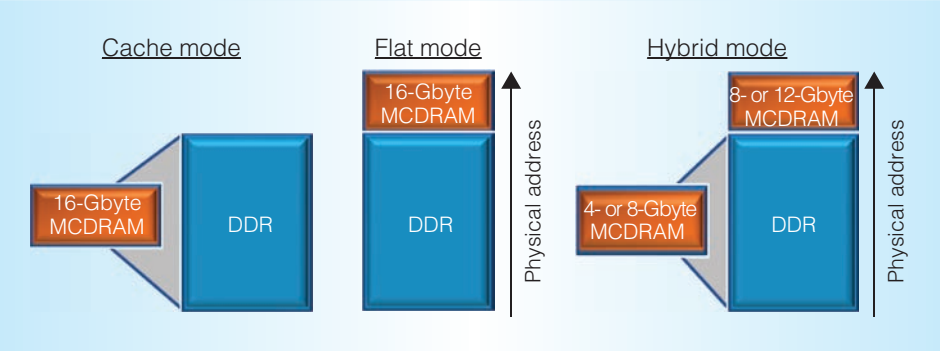

Figure 8. Three memory modes in KNL. These modes—cache, flat, and hybrid—are selectable through the BIOS at boot time.

applications. For applications that do not show a good hit rate in MCDRAM, the other two memory modes provide more control to better use MCDRAM.

Flat mode. This mode presents both MCDRAM memory and DDR memory as regular memory mapped in the same system address space. Flat mode is ideal for applications that can separate their data into a larger, lower-bandwidth region and a smaller, higher-bandwidth region. Accesses to MCDRAM in flat mode see guaranteed high bandwidth compared to cache mode, where it depends on the hit rates. Flat mode requires software support to enable the application to take advantage of this mode, unless the workload can fit entirely in MCDRAM.

Hybrid mode. In this mode, we split either half or a quarter of the MCDRAM as cache; the rest is used as flat memory. The cache portion will serve all of the DDR memory. This is ideal for a mixture of applications that benefit from general caching, but also can take advantage by storing critical or frequently accessed data in flat memory. As with the flat mode, software enabling is required to access the flat mode section of MCDRAM when software does not entirely fit into it. The cache mode section does not require any software support.

Flat MCDRAM software architecture. Flat mode requires software to explicitly allocate memory into MCDRAM. We created software architecture for exposing MCDRAM as memory by relying on mechanisms that are already supported in the existing software stack. This minimizes any major enabling effort and ensures that applications written for flat MDRAM remain portable to systems without flat MCDRAM. This software architecture is based on the NUMA memory support (see http://man7.org/linux/man-pages/man7/numa.7.html) that exists in current operating systems and is widely used to optimize software for current multisocket systems, as shown in Figure 9b. We use the same mechanism to expose the two types of memory on KNL as two separate NUMA nodes (see Figure 9a). This provides software with a way to address the two types of memory using NUMA mechanisms. By default, BIOS sets KNL cores to have higher affinity to DDR than MCDRAM. This affinity helps direct all default and noncritical memory allocations to DDR, keeping them out of MCDRAM. To allocate critical memory in MCDRAM, we provide a HBW_malloc library (https://github.com/memkind/memkind) with memory allocation function calls, and a Fortran language annotation called FASTMEM. Figure 9c shows how we use these two methods.

The code written using the HBW_malloc library and FASTMEM attribute will remain portable. On systems that do not have MCDRAM memory, these mechanisms will default to standard memory-allocation policies. MCDRAM memory behaves just like a regular NUMA memory in a two-socket system for all purposes.

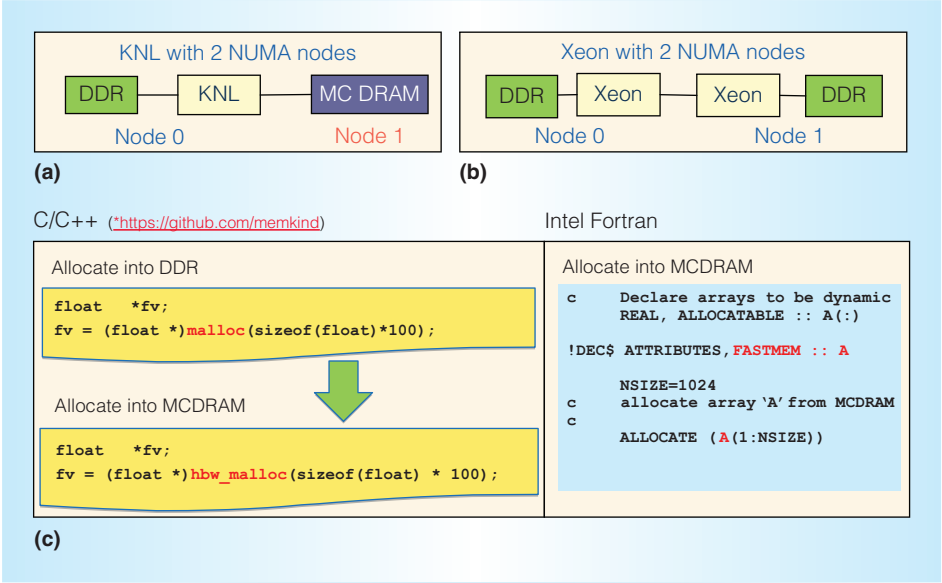

Figure 9. Flat MCDRAM analogy with two-socket Xeon and MCDRAM memory allocation example. (a) KNL in flat MCDRAM modes boots up, showing two NUMA domains. (b) Two-socket Xeon with its own memory showing two NUMA domains. (c) Simple code examples showing the use of functions in HBW_malloc library and FASTMEM attribute in Fortran.

Performance

Figure 10 compares the relative performance and performance per watt of preproduction KNL versus a one-socket ES-2697 V3 series Intel Xeon processor (codenamed Haswell) for a set of applications covering bandwidth benchmarks (Stream), dense linear algebra benchmarks (Linpack), general-purpose workloads (SpecInt and SpecFP), scientific algorithms (QCD-Wilson Dslash, MILC, AMG, SNAP, MiniFE, HPCG, and N-body), weather modeling code (WRF), earthquake modeling code (SeiSol), machine learning (CNN), and financial services code (American Option Approximation). We use cache mode for AMG, SNAP, MILC, MiniFE, SpecInt, and SpecFP, and flat mode for the rest. We use quadrant mode for Linpack, Stream, and Interest Rate Simulations, and all-to-all mode for the rest.

On Linpack and Stream, which represent computing- and bandwidth-bound workloads, KNL provides 3.7 and 8 times the performance and 2.5 and 5.5 times the performance per watt over the one-socket Haswell, respectively. For other categories of workloads that are computing and bandwidth bound to different degrees, early KNL silicon data, mostly unoptimized for KNL, show 2 to 5.3 times the performance and 1.3 to 3.6 times the performance per watt over one-socket Haswell. On SpecInt and SpecFP rates—which exemplify out-of-box, general-purpose performance—early KNL silicon data shows 1.2 to 1.7 times the performance and 0.8 to 1.1 times the performance per watt over one-socket Haswell. The performance of KNL production silicon will generally be higher than these numbers. This shows that KNL delivers respectable performance and performance per watt for general-purpose, out-of-box, unoptimized, parallel code, while providing a significant performance boost for optimized computing- and bandwidth-bound workloads.

Knights Landing's innovations include its power-efficient core, AVX-512 vector instruction set, dual MCDRAM and DDR memory architecture, high-bandwidth on-die interconnect, and an integrated on-package network fabric. These innovations enable the processor to significantly improve performance for computationally intensive and bandwidth-bound workloads while still providing good performance on unoptimized

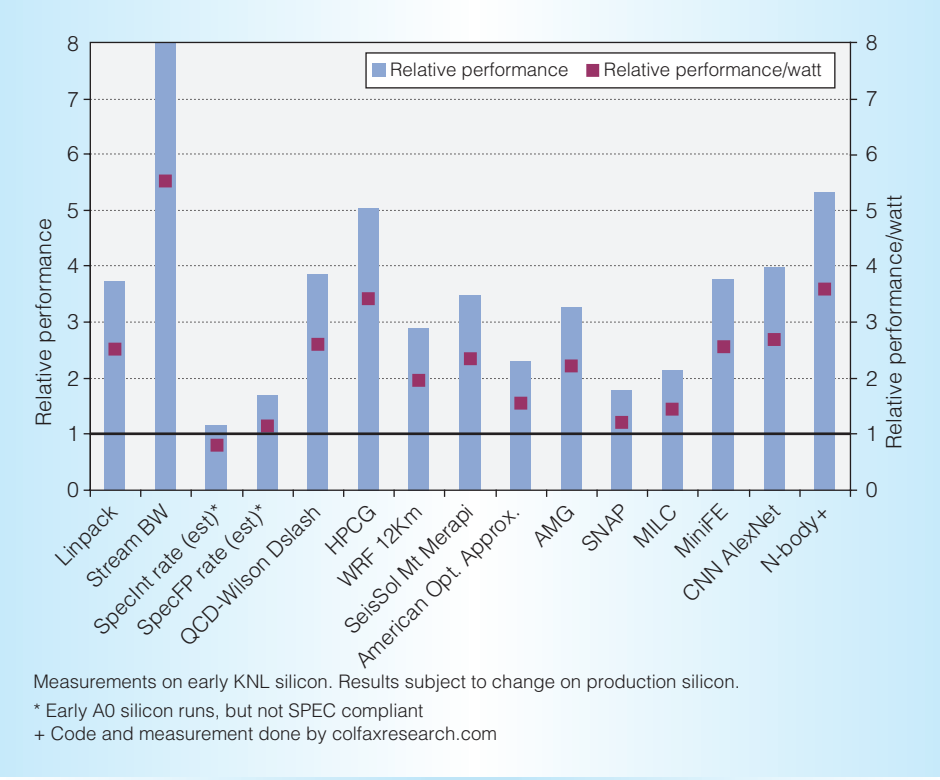

Figure 10. Performance and performance-per-watt comparison of a one-socket KNL at 215 W (preproduction part) versus one-socket Xeon E5-2697v3 at 145 W. Numbers may vary with production parts.

legacy workloads, without requiring any special way of programming other than the standard CPU programming model. MICRO

References

1. G. Chrysos, *Intel Xeon Phi X100 Family Coprocessor—The Architecture*, white paper, Intel, Nov. 2012.
2. B. Kuttanna, *Technology Insight: Intel Silvermont Microarchitecture*, Intel, 2013.
3. *Intel Architecture Instruction Set Extensions Programming Reference*, no. 319433-023, Intel, Aug. 2015.
4. *An Introduction to the Intel QuickPath Interconnect*, white paper, Intel, Jan 2009.
5. M.S. Birrittella et al., *Intel Omni-Path Architecture Technology Overview*, Intel, Aug. 2015.
6. *Intel 64 and IA-32 Architectures Software Developer's Manual: Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B, 3C and 3D*, no. 325462-057US, Intel, June 2015.
7. M. Motoyoshi, "Through-Silicon Via (TSV)," *Proc. IEEE*, vol. 97, no. 1, 2009, pp. 43–48.

Avinash Sodani is a senior principal engineer at Intel and the chief architect of the Knights Landing processor. His research interests include computer architecture, high-performance computing, parallel programming, and applications. Sodani received a PhD in computer science from the University of Wisconsin–Madison. Contact him at avinash.sodani@intel.com.

Roger Gramunt is a senior computer architect at Intel and one of the primary architects of the Knights Landing core. His research interests include core architecture, graphics architecture, binary translation, and vector instruction sets. Gramunt received an MS in computer engineering from

the Universitat Politecnica de Catalunya. Contact him at roger.gramunt@intel.com.

Jesus Corbal is a principal engineer at Intel and one of the primary architects of the Knights Landing vector processing unit. His research interests include vector microarchitectures and instruction-set architecture SIMD extensions. Corbal received a PhD in computer engineering from the Universitat Politecnica de Catalunya. Contact him at jesus.corbal@intel.com.

Ho-Seop Kim is a senior staff computer architect at Intel. His research interests include core architecture, caching and virtualization, performance monitoring, and dynamic binary translation. Kim received a PhD in electrical and computer engineering from the University of Wisconsin–Madison. Contact him at ho-seop.kim@intel.com.

Krishna Vinod is a senior computer architect at Intel and the architect of the Knights Landing L2 cache and core-to-mesh interconnect. His research interests include caching, coherency, and multicore messaging. Vinod received an MS in electrical engineering from Case Western Reserve University. Contact him at krishna.n.vinod@intel.com.

Sundaram Chinthamani is a senior performance architect at Intel and the performance lead of the Knights Landing processor. His research interests include high-performance computing, computer performance modeling using simulation, and queuing networks. Chinthamani received a PhD in computer engineering from the University of Saskatchewan. Contact him at sundaram.chinthamani@intel.com.

Steven Hutsell is a staff microprocessor architect at Intel and one of the primary memory architects for the Knights Landing processor. His research interests include CPU and graphics architecture, memory and caching systems, multichip interconnects, and quantum computing. Hutsell received a PhD in electrical and computer engineering from Portland State University. Contact him at steven.r.hutsell@intel.com.

Rajat Agarwal is a principal engineer at Intel. His research focuses on high-memory-bandwidth solutions for high-performance computing products. Agarwal received a BTech in electronics and communications engineering from the National Institute of Technology, India. Contact him at rajat.agarwal@intel.com.

Yen-Chen Liu is a senior server architect at Intel. His research interests include on-chip/off-chip interconnect fabric, multisocket cache coherency, and chip multiprocessor multicore processor designs. Liu received an MS in electrical and computer engineering from the University of Wisconsin–Madison. Contact him at yen-cheng.liu@intel.com.