IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 14, 2018, accepted January 7, 2019, date of publication January 29, 2019, date of current version February 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2895701

Analysis of Memory System of Tiled Many-Core Processors

YE LIU[1], SHINPEI KATO[2], AND MASATO EDAHIRO[1]
[1]Graduate School of Information Science, Nagoya University, Nagoya 464-8601, Japan
[2]Graduate School of Information Science and Technology, The University of Tokyo, Tokyo 113-8654, Japan

Corresponding author: Ye Liu (liuyeos@gmail.com)

ABSTRACT Tiled many-core processors are designed to integrate simple cores onto a single chip to take advantage of software-level parallelism, and these cores are interconnected via mesh-based networks to mitigate overheads such as limited throughput derived from traditional interconnects. As these processors become more prevalent, one unnoticed problem is that it is more likely for operating system (OS) designers to believe that these processors, which have multiple on-chip memory controllers, belong to the non-uniform memory access (NUMA) system. In this paper, we define novel models regarding the differentiation between uniform memory access and NUMA on tiled many-core processors from the perspective of the cache system to facilitate OS designers and application programmers in fully understanding the underlying hardware. Whether or not a tiled many-core processor belongs to the NUMA system, is determined by the cache system rather than how many memory controllers it has. The experimental results together with the novel models are able to explain why the (non-)significant performance difference can be observed on KNL and TILE-Gx72.

INDEX TERMS NUMA, UMA, memory system, tiled many-core processors.

I. INTRODUCTION

KNL (Knights Landing) [1] from Intel, and TILE-Gx series of processors, including TILE-Gx36 [2] and TILE-Gx72 [3] from Mellanox Technologies, have recently emerged in the market as real tiled many-core processors. These processors, of which cores are fitted onto a single chip and interconnected via mesh-based networks, are different from the well-studied traditional multicore systems described in Section V, from the perspective of computer architecture. Diagrams of KNL and TILE-Gx72 are illustrated in Figure 1. Because memory access latency between cores on separate columns (rows) and an identical memory controller is non-uniform, processors of Figure 1 belong to the NUMA (non-uniform memory access) system. OS (operating system) designers, thus, can believe that an OS designed for traditional multicore (many-core) systems works well on tiled many-core processors. For instance, Barrelfish [4] (a research OS) that treated cores as processors of a distributed system and assumed that there was no inter-core sharing at the lowest level, aimed at solving the scalability problem of the OS for many cores and was believed to be attractive. However, if researchers do not take into consideration the underling shared cache system of tiled many-core processors discussed in this paper and create a Barrelfish-like OS, expected experimental results cannot be achieved. Moreover, a technical report [5] regarding KNL's cluster modes suggests not using KNL as a UMA (uniform memory access) system, and the Linux kernel treats TILE-Gx36/72 as a NUMA system by default. OS designers (or users) may directly treat these tiled many-core processors as the NUMA system and thus devote themselves to mitigating the overhead from memory controllers. On the other hand, KNL and TILE-Gx36/72 can be viewed as another type of processor, of which cores and a single virtual memory controller are fitted onto the single chip, even though multiple on-chip physical memory controllers are available. Therefore, according to conventional models of UMA and NUMA described in Section V, processors of Figure 1 belong to the UMA system as well.

To help better understand the underling memory (cache) system, in this paper, we define novel models regarding the differentiation between UMA and NUMA on tiled many-core processors from the perspective of the cache system. When the home tile (processing core) (and the owner tile together for KNL-like processors) of a given memory block can be any tile(s) on a single chip, a tiled many-core processor belongs to the UMA_{cache} system; when the home tile (and the owner tile together for KNL-like processors) of a given memory block can be designated from a portion of tiles correlated with a memory controller, a tiled many-core processor belongs

The associate editor coordinating the review of this manuscript and approving it for publication was Honghao Gao.

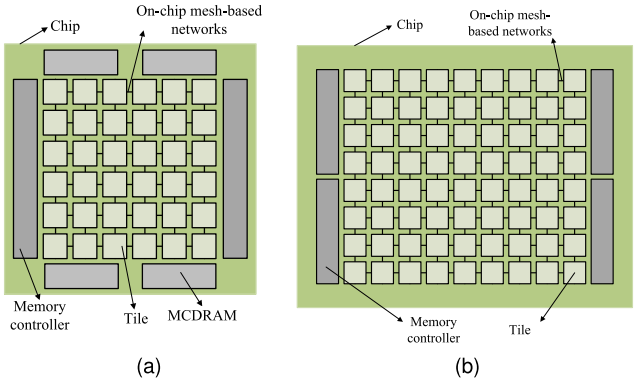

FIGURE 1. Overview of tiled many-core processors. (a) KNL. (b) TILE-Gx72.

to the $NUMA_{cache}$ system. Therefore, TILE-Gx36/72 is a UMA_{cache} processor because it does not support distributing a page to a portion of tiles correlated with a memory controller. Insignificant performance difference between UMA and NUMA systems (those described in Section V) on TILE-Gx36/72 can be predicted when performance is dominated by memory access latency between processing cores (tiles) and main memory. This is because on both UMA and NUMA systems of TILE-Gx36/72, a memory request behaves identically: it is (1) forwarded to the home tile and (2) sent to the memory controller if data is not present in the home tile. In contrast, KNL is viewed as UMA_{cache} and $NUMA_{cache}$ systems with distinct hardware supports (all-to-all and SNC-2/4 cluster modes), and relatively better program performance under UMA_{cache} can be anticipated when (1) on-chip network congestion is not a problem, (2) a program is not aggressive to the cache system, and (3) the main overhead is from the memory system. This is because on KNL the virtual last level cache (LLC) used by a multi-threaded application is larger for the UMA_{cache} system than for $NUMA_{cache}$. With a full understanding of the underlying shared hardware, OS designers can take advantage of characteristics of UMA_{cache} from both TILE-Gx36/72 and KNL to design their light-weight OS. They can make use of features of $NUMA_{cache}$ with KNL as well to design their specific-purpose OS. In general, OSes for UMA_{cache}-based tiled many-core processors seem relatively easier to design than for $NUMA_{cache}$-based ones because there is one node for UMA_{cache} instead of multiple nodes for $NUMA_{cache}$.

Another potential problem arises if OS designers and application programmers highly rely on the cache system of tiled many-core processors. This is because it is well-studied that cache sharing problem between applications is able to greatly degrade performance on a system of which LLC is shared by multiple cores [6]–[10]. One cache-aggressive application can evict cache blocks of another co-scheduled application, even though those evicted cache blocks will be accessed by the co-scheduled application later. OS designers for UMA_{cache}-based tiled many-core processors should adopt feasible methods to prevent a non-cache-aggressive application from being affected by another co-scheduled application as much as possible. Meanwhile, OS designers for $NUMA_{cache}$-based tiled many-core processors can benefit from the hardware support of dividing cores into multiple nodes, but the benefit is limited because the cache sharing problem still happens when the number of applications is higher than the node count. To alleviate the burden placed on OS designers and application programmers, we propose a hybrid memory system that combines benefits from traditional UMA and novel $NUMA_{cache}$ systems, though the relative simulation work will be completed in the future. The main idea is that the node count of tiled many-core processors is managed dynamically by software (i.e., OS/hypervisor) rather than hardware, and a virtual memory controller, for which a page is distributed across multiple physical memory controllers, is supported.

Contributions: The most significant contribution shown in this paper is that we define novel models (UMA_{cache} and $NUMA_{cache}$) on tiled many-core processors. It is based on the cache access latency between a requester tile and the home tile (and the owner tile together for KNL-like processors), instead of on the memory access latency for conventional models (UMA and NUMA). This is because a memory request is first forwarded to the home tile and then sent to the memory controller when the data is not present on the whole cache system. A physical page can be distributed across the whole cache system for a UMA_{cache} processor, while the page can be stored onto a portion of the total caches for a $NUMA_{cache}$ processor. Furthermore, experimental results shown in Section IV correspond to the anticipated program performance difference between conventional UMA and NUMA systems on tiled many-core processors (TILE-Gx72 and KNL). The novel models can help OS designers understand whether or not a specific tiled many-core processor is UMA_{cache} ($NUMA_{cache}$) and rethink what the organization of OSes on these tiled many-core processors should be. For instance, creating a Barrelfish-like OS on TILE-Gx36/72 is not a good choice because of the shared caches. Moreover, the novel models help us easily find out where the overhead is, for instance, a program itself or the OS, when a performance comparison between conventional UMA and NUMA systems reveals an anomaly. It is worth noting that the analysis proposed in this paper mainly aims at helping OS designers (or researchers whose goal is to find out or eliminate performance overheads) understand the effect of the underling cache system of tiled many-core processors.[1]

[1]One specific example of tiled many-core processors is that knights mill (KNM) [11], which is the successor of KNL and was initially released in December, 2017, was designed for the deep learning area. However, according to the public documents [11]–[14], the cache system of KNM is identical to that of KNL. Without a full understanding of the underlying cache system of tiled many-core processors, directly deploying existing frameworks for deep learning to KNM may not achieve the expected experimental results. Note that at the time of writing, we cannot evaluate performance on KNM because of the unavailable machine. However, according to our analysis proposed in this paper, we would like to conjecture that the expected experimental results cannot be achieved without a full understanding of the underlying cache system on KNM.

The remainder of this paper is organized as follows. Section II describes background and motivations of our work. The novel models (UMA_{cache} and $NUMA_{cache}$) are shown in Section III. Experimental results and a hybrid memory system for future tiled many-core processors are exhibited in Section IV. Related work is described in Section V. Future work and a conclusion are given in Section VI.

II. BACKGROUND AND MOTIVATIONS

A. USE CASES ON KNL[2]

As suggested by the technical report [5] regarding KNL's cluster modes, users avoided to use KNL as a UMA system. Byun *et al.* [15] evaluated performance of data-analysis and machine-learning applications on KNL. For the experiments, they avoided to use KNL as a UMA_{cache} (with all-to-all cluster mode) system. However, they failed to exhibit experiment results (from others work if existed) to explain why a UMA_{cache} tiled many-core processor was not a good choice. You *et al.* [16] redesigned algorithms for the deep learning area to expedite the training process and evaluated them on GPU and KNL clusters. They directly used KNL as a $NUMA_{cache}$ system. Awan *et al.* [17] evaluated performance of CPU- and GPU-based DNN training, but failed to state if KNL was treated as a UMA_{cache} or $NUMA_{cache}$ system. Allen *et al.* [18] evaluated performance of workloads on KNL and Haswell architectures, but failed to explore performance difference between UMA_{cache} and $NUMA_{cache}$ systems as well. Calore *et al.* [19] focused on performance and energy efficiency on KNL, but also failed to discuss performance difference between UMA_{cache} and $NUMA_{cache}$ systems. Moreover, researchers concentrated on performance improvement from MCDRAM-based cache. However, experimental results shown in [17] demonstrated that the improvement was trivial. Our analysis described in this paper reveals that the effect from the cache system cannot be ignored. Therefore, we need to pay attention to whether or not a tiled many-core processor is a UMA_{cache} ($NUMA_{cache}$) system. To understand why it is necessary to define the novel models discussed in Section III, we first mainly describe the cache coherence protocol employed by traditional multicore systems in Section II-B and then exhibit its counterpart employed by emerging tiled many-core processors in Section II-C.

B. TRADITIONAL MULTICORE SYSTEM

Figure 2 illustrates the organization of a multi-socket system that is a typical example of traditional multicore systems adopted in the HPC (high performance computing) area, on which nodes are interconnected via 8/16-bit point-to-point links, and each node embraces eight cores and one on-chip memory controller. According to the conventional models of UMA and NUMA described in Section V, the multi-socket system exhibited in Figure 2 is a NUMA system. A non-uniform memory access latency clearly exists, since latency between a core from one node and a memory controller from another node is larger than that generated when the core accesses a memory controller from the same node, due to wire delay and limited interconnect throughput. Research work has been done to take advantage of the non-uniform characteristics of NUMA systems. For instance, in [20], it was pointed out that bandwidth caused by an asymmetric interconnect between nodes matters more than distance between them. That is, the asymmetric interconnect between a 16-bit point-to-point link connecting intra-socket nodes and an 8-bit point-to-point link connecting inter-socket nodes in Figure 2, is the main overhead to program performance.

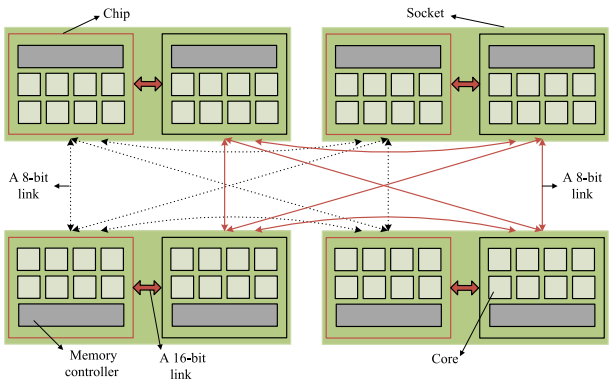

FIGURE 2. Multi-socket system.

To guarantee the memory consistency [21] in terms of a whole cache system composed of node-based LLCs, the snooping-based cache coherence protocol is adopted on the traditional NUMA-based multicore system. When a cache miss takes place on the LLC (i.e., a shared L2/L3 cache) on one node, a snoop message is broadcast on the node-to-node interconnect(s). The requested data is subsequently transferred from the corresponding main memory when no copy exists on the whole cache system, or from the cache when one node detects the snoop message and finds that it has a copy. The snooping-based cache coherence protocol can have separate implementations depending on the design requirements. For instance, home snoop mode, of which a home node issues a snoop to a node that may have a copy of the requested data, and source snoop mode, of which the requester node issues snoops to all other node(s), are both supported on Intel processors [22] with QPI [23], [24].

Since a directory that tracks on which node(s) a copy of a cache line exists is adopted by home snoop mode on Intel processors with QPI and its implementation is a little similar to the directory-based cache coherence protocol on KNL, we, therefore, take it as an example to explain how the snooping-based cache coherence protocol works. On each node of a NUMA-based traditional multicore system, a CA (caching agent) connects to a cache controller, and a HA (home agent) connects to its on-chip memory controller. When a cache

[2]As discussed in this paper, KNL supports both UMA_{cache} and $NUMA_{cache}$ models, while TILE-Gx36/72 supports the UMA_{cache} model. Only use cases on KNL are discussed in this subsection to exhibit that users avoided to use KNL with the UMA_{cache} model, even though they could treat it as a UMA_{cache} processor.

miss happens, on the basis of home snoop mode, the CA on the requester node is responsible for sending a request to the HA on the home node. The requested data exists in the main memory associated with the home node when there is no copy on the whole cache system. The HA then checks the directory to see if other nodes hold a copy of the requested data, and it asks that node to transfer data to the requester node if so. Otherwise, data is transferred from HA's associated main memory to the requester node directly. Readers can refer to [25] for further information.

C. TILED MANY-CORE PROCESSORS

Figure 1 exhibits a brief overview of emerging tiled many-core processors (TILE-Gx72 and KNL). One common feature belonging to these processors is that cores interconnected via mesh-based networks are fitted onto a single chip. That is the main characteristic of these processors that differentiates them from traditional multicore systems, and this can be clearly seen from comparing Figures 2 and 1. On TILE-Gx72, each tile (core) has an L1 cache and an L2 cache, and on KNL, each tile consisting of two cores has an L2 cache, but each core has an L1 cache. That is, on KNL, the L2 cache of each tile is shared by two cores, but on TILE-Gx72, the L2 cache is accessed by one core. Non-uniform memory access latency also exists on tiled many-core processors because of wire delay. According to the conventional models on UMA and NUMA described in Section V, TILE-Gx72 and KNL in Figure 1 both belong to the NUMA system. However, since all cores are integrated onto a single chip, both tiled many-core processors support a virtual memory controller from the perspective of hardware, which might not be easily implemented on the traditional multicore system. Therefore, both tiled processors belong to the UMA system.

A virtual memory controller is supported by TILE-Gx36/72 via the technique of memory stripping, of which a page is interleaved across available physical memory controllers with a stripping granularity of 512 bytes. That is, if a physical address A is allocated from the physical memory controller N, then the physical address A + 512 is allocated from the memory controller N + 1. Enabling/disabling memory stripping on TILE-Gx36/72 can be managed by system software (called hypervisor). When memory stripping is disabled by default, designers of TILE-Gx36/72 believe that it belongs to the NUMA system. Meanwhile, a virtual memory controller, is supported by KNL via the distribution of memory addresses across all on-chip per-tile tag directories (TDs). When cores are divided into multiple groups in accordance with the memory-controller count, and memory addresses of a given page can only be distributed across TDs of a specific group, designers of KNL believe that it belongs to the NUMA system. The technical report [5] regarding KNL's cluster modes suggests not using KNL as UMA, and the Linux kernel treats topology of TILE-Gx36/72 as NUMA by default. If researchers (or users) follow these guides (i.e., [5] and the source code of the Linux kernel for TILE-Gx36/72), and ignore the UMA characteristic of tiled many-core processors, creating a Barrelfish-like OS or devoting themselves to eliminating the overhead from memory controllers is possible. Therefore, it is more likely to see that the expected experimental results can not be achieved, and/or the root cause of the performance problems is missed. We notice that KNL and TILE-Gx36/72 have distinct techniques for supporting the virtual memory controller and they may be able to help us explore the novel models of UMA_{cache} and $NUMA_{cache}$ on tiled many-core processors.

To help fully understand the novel models of UMA_{cache} and $NUMA_{cache}$, we thus introduce directory-based cache coherence protocols employed by KNL and TILE-Gx36/72. On KNL, when a cache miss happens, a request (from a requester tile) is forwarded to a designated tile (called home tile in this paper), on which a directory tracks where the owner tile (a tile that keeps a copy of the requested data) is; then a message is sent from the home tile to the owner tile in order to transfer data to the requester tile when a valid copy exists. If no data is present in the cache system, the home tile will forward the request to the associated physical memory controller. It seems that the home snoop mode of the snooping-based cache coherence protocol is similar to the directory-based cache coherence protocol for KNL; the main difference is dependent on how the owner node (tile) detects a request message and responds to it. On TILE-Gx36/72, when a cache miss takes place, a request (from the requester tile) is forwarded to a designated tile (which is home tile and owner tile as well), and data is transferred from the home tile to the requester tile when a valid copy exists. Otherwise, the home tile will forward the request to the associated physical memory controller as well. Therefore, we can find that it is the home tile rather than the requester tile that sends a request to the memory controller when the requested data is not present in the whole cache system. An exceptional example, in which the requester tile is local to the memory controller but the home tile is remote, is common to tiled many-core processors. Thus, we need to rethink the models of UMA and NUMA for tiled many-core processors.

III. NOVEL MODELS

In this section, we define the novel models (UMA_{cache} and $NUMA_{cache}$), which are dominated by the cache system on shared-memory tiled many-core processors. Compared with traditional multicore systems described in Section II-B, current technology guarantees that cores can be fitted onto a single chip for tiled many-core processors and the on-chip cache system is shared by them. It is worth noting that the memory consistency [21], which is supported by the cache coherence protocol, is important to a shared-memory system in order to guarantee correct execution of applications from the HPC area. However, cache coherence protocols (i.e., a two-hop protocol employed by TILE-Gx36/72 and a three-hop counterpart employed by KNL (Intel processor)) vary among different architectures. The way the cache coherence protocol works determines how the cache system is shared by cores on tiled many-core processors.

TABLE 1. Comparison between conventional models and novel models on tiled many-core processors.

Tiled many-core processor	Conventional models	Novel models
TILE-Gx72	UMA (A virtual memory controller is supported)	UMA_{cache} (Hash-for-home strategy across all tiles)
	NUMA (Multiple memory controllers are available)	
KNL	UMA (A virtual memory controller is supported)	UMA_{cache} (All-to-all cluster mode and flat memory mode)
	NUMA (Multiple memory controllers are available)	$NUMA_{cache}$ (SNC-4 cluster mode and flat memory mode)

On the basis of how the cache coherence protocol works, the novel models are defined as follows: if a physical page can be distributed across all available tiles (caches) of a tiled many-core processor, the processor belongs to a UMA_{cache} system (with a relatively larger virtual LLC); if a physical page can be distributed across a portion of tiles (caches) associated with a memory controller, the processor belongs to a $NUMA_{cache}$ system (with a relatively smaller virtual LLC). That is, on a UMA_{cache} tiled many-core processor, on-chip caches can be viewed as a single virtual cache and thus memory blocks can be evicted later compared with a $NUMA_{cache}$ processor, when the data layout is carefully designed. In contrast, multiple LLCs coexist on a $NUMA_{cache}$ tiled many-core processor and it can help divide separate-purpose data into multiple groups. However, a $NUMA_{cache}$ tiled many-core processor is less likely to be beneficial to a multi-threaded application from the HPC area when that application is not cache-aggressive. It is worth noting that an appropriate cache system[3] (i.e., the one reducing effects from system noise [26]–[29] and/or solving the cache sharing problem caused by co-scheduled applications, and employing a scalable cache coherence scheme [30]) on future shared-memory tiled many-core processors poses challenges to computer architects and software designers. When a new tiled many-core processor is released and details of the cache system including the cache coherence protocol are available, it is easy for researchers to treat it as a UMA_{cache} or $NUMA_{cache}$ processor, on the basis of the above novel models.

To further understand the definition, a comparison between conventional models and novel models on KNL and TILE-GX72 is shown in Table 1. Note that the SNC-2 cluster mode of KNL is not included in Table 1 as NUMA, since it is similar to the SNC-4 cluster mode but with two NUMA nodes. On the basis of the novel models of UMA_{cache} and $NUMA_{cache}$, we are able to anticipate the performance comparison between conventional UMA and NUMA systems when a single multi-threaded application runs on KNL and TILE-Gx72 respectively, assuming that access to the memory system is the main overhead. Because TILE-Gx72 belongs only to the UMA_{cache} system, we can predict that there will be an observable, insignificant performance difference between traditional UMA and NUMA systems, unless on-chip network congestion is a serious problem. Meanwhile, we can predict that, on KNL, memory-intensive and non-cache-aggressive programs will benefit from UMA_{cache} since all available caches can be used to keep a copy of a memory block, and thus, it has a larger virtual LLC than $NUMA_{cache}$.

[3]Note that the cache system design which is differentiated from the main-memory (i.e., DRAM) system design in this paper, is complicated. For example, computer architects should consider their design choices between an inclusive LLC and a non-inclusive LLC, and the selection between a private cache and a shared cache. Especially, designing an appropriate cache coherence protocol [31] (i.e., a scalable cache coherence scheme [30]) for many cores can be an individual research topic. Our analysis in this paper, on the one hand, helps computer architects understand that their designs have an influence on software frameworks designed by OS designers and/or application programmers on many cores; on the other hand, it helps OS designers and application programmers understand the effect of the underlying shared cache system on emerging tiled many-core processors (i.e., KNL and TILE-Gx36/72) when considering their software frameworks in order to pursue high performance.

IV. PERFORMANCE ANALYSIS

To help understand that it is not unimportant to define the novel models of UMA_{cache} and $NUMA_{cache}$, we selected programs from PARSEC [32], SPLASH-2X [33], NAS Parallel Benchmarks [34], and BigDataBench [35] to run on KNL and TILE-Gx72. Detailed hardware information is shown in Table 2. We run programs with settings from the perspective of conventional models but explain the experimental results in terms of novel models.

A. EXPERIMENTAL SETUP

The running settings for the experiment are shown in Table 3. Note that the experimental settings with conventional models and novel models on KNL are identical. The selected applications are listed in Table 4. Canneal, raytrace, vips, and x264 from the PARSEC benchmark suite could not be correctly compiled for TILE-Gx72. CONFIG_NUMA was disabled when compiling the Linux kernel (Linux-4.5) for the UMA model. With every setting combination (for instance, memory stripping was disabled and the first-touch memory allocation policy was adopted for the UMA model on TILE-Gx72) exhibited in Table 3, each program was run three times. Average values were used to plot figures shown in Figures 3 and 4. HPC experts may argue that program performance is dominated by fetching data from a remote memory controller, and/or burdening a specific memory controller rather than the effect of the cache system of tiled many-core processors, on the basis of their experiences on traditional many-core systems. We therefore evaluated if the program performance was improved when tasks and data

TABLE 2. Detailed hardware information for emerging tiled many-core processors (KNL[4] nd TILE-Gx72).

	TILE-Gx72	KNL
Count of on-chip tiles	72	32
Count of cores of each tile	1	2
Core frequency	1.0 GHz	1.30 GHz
Capacity of L1 cache	32 KB	32 KB
Capacity of L2 cache	256 KB	1 MB
Capacity of total L2 caches	18 MB	32 MB
Size of the cache line	64 bytes	64 bytes
On-chip interconnection	Mesh-based networks	Mesh-based networks
On-chip memory controller	DDR3	DDR4
Equipped with MCDRAM?	no	yes
Memory capacity	32 GB	96 GB (DDR4) + 16 GB (MCDRAM)
Message routing scheme	Static XY routing	Static YX routing
Message routing overhead	One cycle if packets go straight Two cycles if packets make a turn	One clock in the Y direction Two clocks in the X direction
Processor type	PCIe-connected coprocessor	Stand-alone processor

TABLE 3. Settings for the experiment.

	TILE-Gx72	KNL
Compiler and library	GCC-4.8.2 (Glibc-2.12) MPICH-3.2	GCC-4.8.5 (Glibc-2.17) MPICH-3.2
UMA	Memory stripping is enabled	All-to-all cluster mode and flat memory mode
NUMA	Memory stripping is disabled	SNC-4 cluster mode and flat memory mode
Count of cores	72	64 (without Hyper-Threading)[5]
Memory allocation policy	first-touch (UMA) first-touch (NUMA)	first-touch (UMA) first-touch (NUMA) Automatic NUMA balancing (NUMA)

were closer with the automatic NUMA balancing policy. It is important to note that the goal of the automatic NUMA balancing policy supported by the OS (Linux) is to make tasks (threads/processes) running on the cores and data present in the main memory closer. Readers can refer to [36] for more detailed information. Since the automatic NUMA balancing policy is not supported by the Linux kernel for TILE-Gx36/72, we keep only the performance evaluation on the NUMA model with the first-touch policy for TILE-Gx72. The experimental results in Section IV-B can help us explain that the novel models are feasible.

[4]The hardware information shown in this table exhibits KNL on which the experiment in this paper is done, though up to 68 cores can be supported on another KNL.

[5]Hyper-Threading, which supports multiple (i.e., 4 for KNL) logical cores on an identical physical core, is guaranteed by Intel processors rather than TILE-Gx36/72. The initial and the most significant goal of Hyper-Threading is to take advantage of instruction-level parallelism and thus it is expected to improve performance. However, scalability problem, which presents that application performance cannot be improved when more threads are configured on many cores, is observed on tiled many-core processors as well, as illustrated in Figures 3 and 4. Enabling Hyper-Threading on KNL can further degrade performance due to the shared cache system. That is, tasks of parallel applications, which run on logical cores of the same physical core, are able to evict cache lines belonging to each other and thus data must be fetched from the high-overhead main-memory system as a result of increased cache misses. Since our goal is to help understand the underlying shard cache system of tiled many-core processors and make use of it to improve performance, we, therefore, do not evaluate application performance when Hyper-Threading is enabled.

B. PERFORMANCE

Figures 3 and 4 exhibit program performance on TILE-Gx72 and KNL, respectively. The x-axis shows the thread (process) count designated to run a program, which is able to be less than the total active thread (process) count. For instance, the active thread count for dedup (a pipeline program from PARSEC) is 3(n+1), but the thread count shown in Figures 3c and 4c to run this program is n (for each parallel stage). The y-axis illustrates the speedup normalized to a baseline, for which only one thread (process) is adopted to run the program. Label F in the parentheses represents that the first-touch policy is employed. Label A in the parentheses stands for the automatic NUMA balancing policy. The abnormal performance with the NUMA model for radiosity in Figure 3q was caused by the program being incorrectly executed. Similar abnormal performance with 51 and 59 threads for cholesky with UMA and NUMA models was also caused by failed execution, as shown in Figures 3k and 4o. Note that UMA and NUMA were recognized with conventional models in the experiment. This is because the intention of the experiment is to see whether or not it is necessary to redefine models of UMA and NUMA for tiled many-core processors and to meanwhile evaluate whether or not the novel models are feasible. #Ts in Tables 5, 6, 7, and 8 refers to the number of threads presented in Figure 3. # of threads in Tables 9, 10, 11, and 12 refers to the thread count shown in Figure 4. SD means the standard deviation.

TABLE 4. Selected applications for the experiment.

Application	Benchmark suite	Programming model	Problem size	Experimental platform
blackscholes	PARSEC	Pthreads	Native	KNL, TILE-Gx72
bodytrack	PARSEC	Pthreads	Native	KNL, TILE-Gx72
canneal	PARSEC	Pthreads	Native	KNL
dedup	PARSEC	Pthreads	Native	KNL, TILE-Gx72
facesim	PARSEC	Pthreads	Native	KNL, TILE-Gx72
ferret	PARSEC	Pthreads	Native	KNL, TILE-Gx72
fluidanimate	PARSEC	Pthreads	Native	KNL, TILE-Gx72
freqmine	PARSEC	OpenMP	Native	KNL, TILE-Gx72
raytrace	PARSEC	Pthreads	Native	KNL
streamcluster	PARSEC	Pthreads	Native	KNL, TILE-Gx72
swaptions	PARSEC	Pthreads	Native	KNL, TILE-Gx72
vips	PARSEC	Pthreads	Native	KNL
x264	PARSEC	Pthreads	Native	KNL
barnes	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
cholesky	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
fft	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
lu_cb	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
lu_ncb	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
ocean_cp	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
ocean_ncp	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
radiosity	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
raytrace	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
water_nsquared	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
water_spatial	SPLASH-2X	Pthreads	Native	KNL, TILE-Gx72
bt	NPB-OMP	OpenMP	Class C	KNL, TILE-Gx72
cg	NPB-OMP	OpenMP	Class C	KNL, TILE-Gx72
dc	NPB-OMP	OpenMP	Class W	KNL, TILE-Gx72
ep	NPB-OMP	OpenMP	Class D	KNL, TILE-Gx72
is	NPB-OMP	OpenMP	Class C	KNL, TILE-Gx72
lu	NPB-OMP	OpenMP	Class C	KNL, TILE-Gx72
sp	NPB-OMP	OpenMP	Class C	KNL, TILE-Gx72
ua	NPB-OMP	OpenMP	Class C	KNL, TILE-Gx72
Grep	BigDataBench	MPI	10 GB	KNL, TILE-Gx72
Sort	BigDataBench	MPI	10 GB	KNL, TILE-Gx72
WordCount	BigDataBench	MPI	10 GB	KNL, TILE-Gx72

TABLE 5. MPKI values of streamcluster on TILE-Gx72.

#Ts	UMA (F)				NUMA (F)			
	LOCAL	SD	REMOTE	SD	LOCAL	SD	REMOTE	SD
63	0.334	0.012	4.913	0.154	0.356	0.002	5.188	0.023
64	0.334	0.012	4.957	0.140	0.352	0.004	5.154	0.042
65	0.330	0.009	4.872	0.118	0.356	0.002	5.194	0.026
66	0.326	0.011	4.747	0.128	0.354	0.006	5.078	0.076
67	0.321	0.011	4.650	0.130	0.351	0.013	5.024	0.168
68	0.327	0.010	4.741	0.127	0.351	0.005	5.017	0.068
69	0.318	0.027	4.607	0.335	0.342	0.007	4.889	0.087
70	0.297	0.013	4.360	0.144	0.321	0.040	4.621	0.550
71	0.300	0.014	4.361	0.144	0.321	0.012	4.599	0.147
72	0.301	0.003	4.409	0.040	0.308	0.006	4.487	0.099

TABLE 6. MPKI values of cholesky on TILE-Gx72.

#Ts	UMA (F)				NUMA (F)			
	LOCAL	SD	REMOTE	SD	LOCAL	SD	REMOTE	SD
63	0.011	0.001	0.346	0.044	0.017	0.002	0.524	0.059
64	0.010	0.000	0.344	0.009	0.014	0.002	0.455	0.050
65	0.012	0.001	0.379	0.014	0.016	0.002	0.514	0.063
66	0.012	0.001	0.365	0.030	0.016	0.001	0.488	0.027
67	0.009	0.001	0.357	0.041	0.012	0.002	0.446	0.049
68	0.012	0.001	0.374	0.027	0.011	0.001	0.347	0.046
69	0.012	0.001	0.372	0.041	0.012	0.002	0.381	0.056
70	0.013	0.002	0.394	0.055	0.015	0.003	0.445	0.084
71	0.010	0.000	0.378	0.006	0.010	0.001	0.371	0.037
72	0.012	0.001	0.359	0.016	0.011	0.001	0.319	0.030

TABLE 7. MPKI values of bt on TILE-Gx72.

#Ts	UMA (F)				NUMA (F)			
	LOCAL	SD	REMOTE	SD	LOCAL	SD	REMOTE	SD
63	0.101	0.000	1.885	0.003	0.102	0.000	1.884	0.005
64	0.102	0.001	1.883	0.002	0.101	0.001	1.890	0.003
65	0.102	0.001	1.890	0.003	0.101	0.001	1.893	0.004
66	0.102	0.001	1.898	0.006	0.101	0.001	1.896	0.004
67	0.103	0.002	1.901	0.005	0.102	0.001	1.906	0.001
68	0.102	0.002	1.910	0.004	0.102	0.000	1.911	0.006
69	0.104	0.000	1.911	0.003	0.103	0.001	1.913	0.003
70	0.102	0.000	1.919	0.004	0.102	0.001	1.924	0.006
71	0.104	0.001	1.933	0.003	0.103	0.001	1.930	0.003
72	0.103	0.002	1.936	0.009	0.105	0.001	1.935	0.004

TABLE 8. MPKI values of Grep on TILE-Gx72.

#Ts	UMA (F)				NUMA (F)			
	LOCAL	SD	REMOTE	SD	LOCAL	SD	REMOTE	SD
63	0.036	0.001	1.970	0.029	0.036	0.002	2.028	0.023
64	0.037	0.000	2.004	0.026	0.037	0.000	2.028	0.038
65	0.037	0.001	1.977	0.037	0.038	0.001	2.037	0.036
66	0.038	0.002	1.996	0.033	0.038	0.001	2.050	0.033
67	0.037	0.001	1.993	0.035	0.038	0.002	2.034	0.037
68	0.038	0.001	2.010	0.025	0.038	0.001	2.012	0.022
69	0.037	0.001	1.973	0.035	0.038	0.002	2.018	0.010
70	0.038	0.003	2.011	0.041	0.039	0.002	2.011	0.019
71	0.038	0.001	2.002	0.024	0.039	0.002	2.015	0.023
72	0.035	0.001	1.859	0.025	0.034	0.001	1.821	0.042

1) PERFORMANCE ON TILE-GX72

On average, for most programs running on TILE-Gx72, as illustrated in Figure 3, an insignificant difference in performance between UMA and NUMA models is observed. The observation corresponds to the anticipation on performance comparison between conventional UMA and NUMA

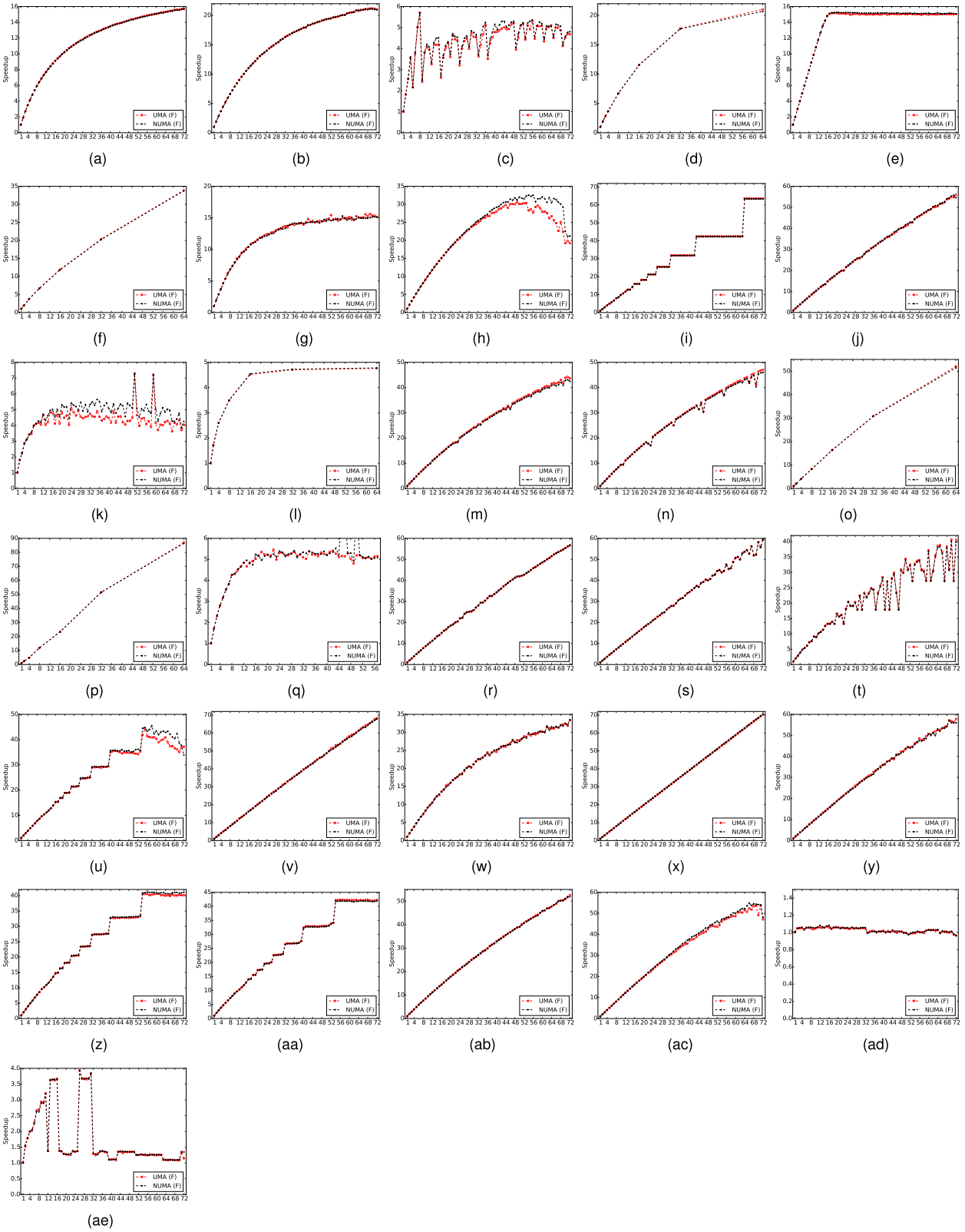

FIGURE 3. Speedup on TILE-Gx72. (a) blackscholes. (b) bodytrack. (c) dedup. (d) facesim. (e) ferret. (f) fluidanimate. (g) freqmine. (h) streamcluster. (i) swaptions. (j) splash2x.barnes. (k) splash2x.cholesky. (l) splash2x.fft. (m) splash2x.lu_cb. (n) splash2x.lu_ncb. (o) splash2x.ocean_cp. (p) splash2x.ocean_ncp. (q) splash2x.radiosity. (r) splash2x.raytrace. (s) splash2x.water_nsquared. (t) splash2x.water_spatial. (u) bt.C.x. (v) cg.C.x. (w) dc.W.x. (x) ep.D.x. (y) is.C.x. (z) lu.C.x. (aa) sp.C.x. (ab) ua.C.x. (ac) Grep. (ad) Sort. (ae) WordCount.

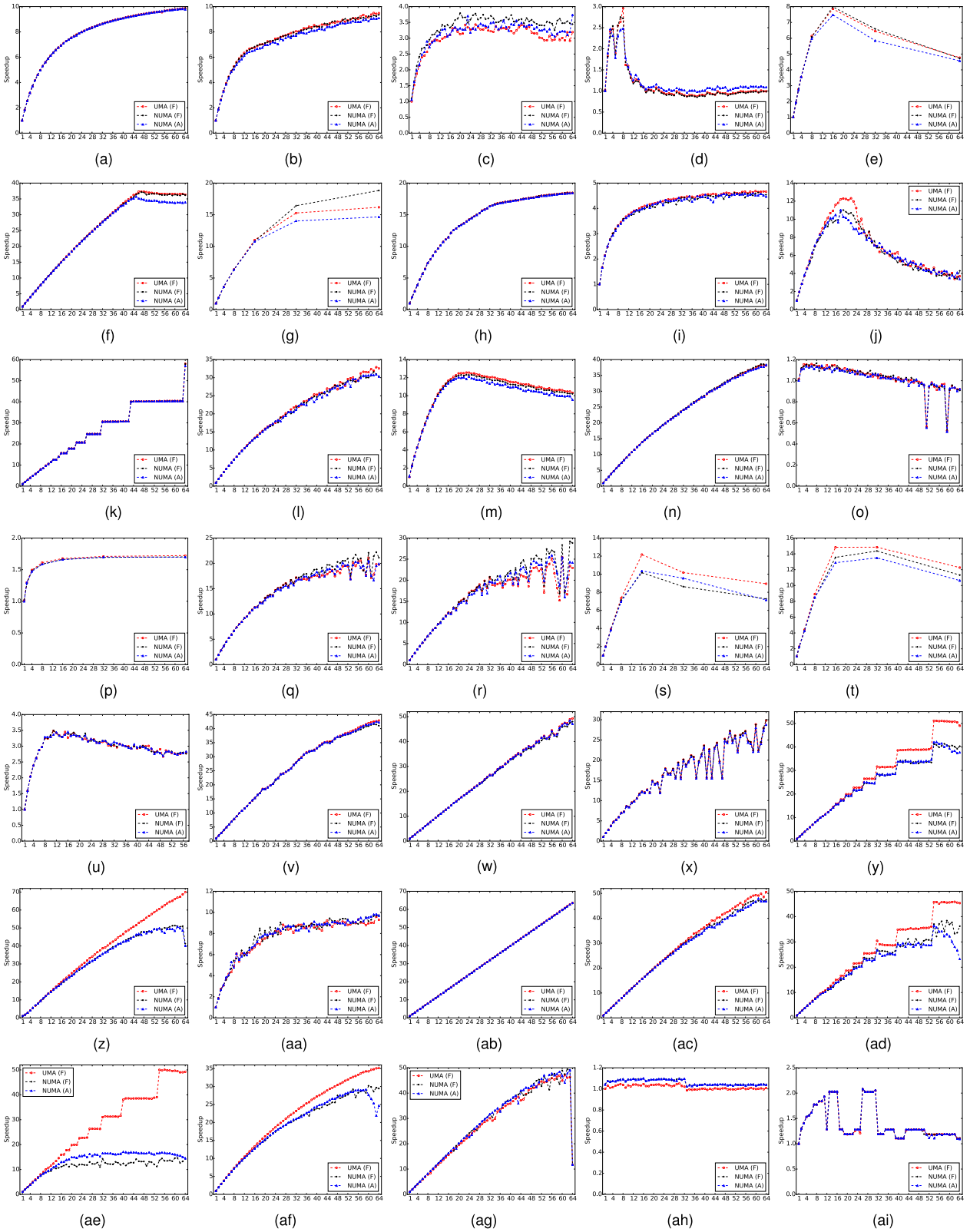

FIGURE 4. Speedup on KNL. (a) blackscholes. (b) bodytrack. (c) canneal. (d) dedup. (e) facesim. (f) ferret. (g) fluidanimate. (h) freqmine. (i) raytrace. (j) streamcluster. (k) swaptions. (l) vips. (m) x264. (n) splash2x.barnes. (o) splash2x.cholesky. (p) splash2x.fft. (q) splash2x.lu_cb. (r) splash2x.lu_ncb. (s) splash2x.ocean_cp. (t) splash2x.ocean_ncp. (u) splash2x.radiosity. (v) splash2x.raytrace. (w) splash2x.water_nsquared. (x) splash2x.water_spatial. (y) bt.C.x. (z) cg.C.x. (aa) dc.W.x. (ab) ep.D.x. (ac) is.C.x. (ad) lu.C.x. (ae) sp.C.x. (af) ua.C.x. (ag) Grep. (ah) Sort. (ai) WordCount.

TABLE 9. MPKI values of fluidanimate on KNL.

# of threads	UMA (F)		NUMA (F)		NUMA (A)	
	MPKI	SD	MPKI	SD	MPKI	SD
1	0.718	0.001	0.722	0.002	0.722	0.004
2	0.842	0.001	0.842	0.006	0.845	0.001
4	0.863	0.001	0.859	0.007	0.869	0.003
8	0.991	0.002	0.987	0.005	1.004	0.004
16	1.007	0.001	0.950	0.003	1.040	0.003
32	1.236	0.003	1.172	0.004	1.316	0.004
64	1.324	0.002	1.270	0.005	1.407	0.005

TABLE 10. MPKI values of canneal on KNL.

# of threads	UMA (F)		NUMA (F)		NUMA (A)	
	MPKI	SD	MPKI	SD	MPKI	SD
55	23.144	0.312	23.131	0.490	22.297	0.719
56	23.257	0.258	23.265	0.426	22.702	0.089
57	23.531	0.056	23.113	0.475	22.873	0.177
58	23.243	0.503	22.936	0.387	22.800	0.122
59	23.035	0.113	23.184	0.464	22.334	0.308
60	23.355	0.421	23.321	0.321	22.623	0.173
61	23.318	0.354	22.888	0.620	22.560	0.227
62	22.960	0.147	23.251	0.445	22.573	0.544
63	23.356	0.392	23.177	0.159	22.104	0.405
64	23.217	0.479	22.821	0.148	22.352	0.234

TABLE 11. MPKI values of sp.C.x on KNL.

# of threads	UMA (F)		NUMA (F)		NUMA (A)	
	MPKI	SD	MPKI	SD	MPKI	SD
58	1.812	0.006	2.241	0.117	2.062	0.015
59	1.812	0.011	2.200	0.122	2.058	0.028
60	1.802	0.010	2.220	0.114	2.061	0.040
61	1.807	0.013	2.245	0.082	2.081	0.014
62	1.802	0.010	2.229	0.093	2.086	0.036
63	1.798	0.004	2.251	0.089	2.031	0.007
64	1.787	0.005	2.301	0.090	2.029	0.031

TABLE 12. MPKI values of Sort on KNL.

# of threads	UMA (F)		NUMA (F)		NUMA (A)	
	MPKI	SD	MPKI	SD	MPKI	SD
55	0.135	0.002	0.127	0.002	0.128	0.003
56	0.130	0.001	0.125	0.002	0.125	0.000
57	0.128	0.002	0.125	0.001	0.129	0.001
58	0.130	0.003	0.127	0.002	0.129	0.002
59	0.130	0.001	0.128	0.002	0.127	0.002
60	0.137	0.002	0.128	0.003	0.132	0.002
61	0.129	0.001	0.126	0.003	0.128	0.002
62	0.129	0.004	0.126	0.003	0.129	0.002
63	0.128	0.002	0.126	0.002	0.129	0.003
64	0.212	0.146	0.204	0.133	0.202	0.126

models on TILE-Gx72, as described in Section III. This is because for the conventional UMA and NUMA models supported by TILE-Gx36/72, they belong to the same UMA_{cache} model on the basis of novel models proposed in this paper. Streamcluster (Figure 3h), cholesky (Figure 3k), bt (Figure 3u), and Grep (Figure 3ac) seem to be exceptions. The experimental results exhibit that streamcluster, cholesky, bt, and Grep perform better with the conventional NUMA model than UMA. However, since we know that the cache behaviors of the UMA model are similar to those of NUMA, we can infer that the performance degradation is triggered by other parts instead of the memory system.

An insignificant difference in the numbers of cache misses per kilo instructions (MPKI) between UMA and NUMA models can be clearly observed from Tables 5, 6, 7, and 8. These MPKI values support the inference. Two kinds of cache misses (LOCAL and REMOTE) were calculated for TILE-Gx72. LOCAL, as exhibited in Tables 5, 6, 7 and 8, represents that the cache block is not present in the virtual LLC and must be fetched from the main memory. REMOTE means that the cache block is not present in the L2 cache of the current processing core but may reside in the L2 cache of the home tile. A LOCAL cache miss refers to a real cache miss, but a REMOTE cache miss does not eventually cause a costly cache miss. Our further analysis regarding the performance difference of streamcluster and cholesky between UMA and NUMA models on TILE-Gx72, demonstrates that the difference is triggered by the Linux kernel. The root cause and the solution to the problem will be presented in the next work. We therefore believe that there is no difference between UMA and NUMA models when the significant overhead is from the main memory. The novel models of UMA_{cache} and $NUMA_{cache}$ are able to explain why most programs exhibit almost identical performance between UMA and NUMA models. When other applications rather than those evaluated in this paper perform abnormally between UMA and NUMA models, those people who concern the performance can find out the root cause easily with the help of other tools since the memory system is excluded from being the overhead. However, without a full understanding of the effect of the underlying cache system (i.e., the way the cache coherence protocol works is important) of TILE-Gx36/72, HPC experts may think that it is a little weird due to the observation that insignificant performance difference between the conventional UMA and NUMA models appears.

2) PERFORMANCE ON KNL

As illustrated in Figure 4, nine programs clearly benefit from the UMA (also UMA_{cache}) model and three programs (fluidanimate, canneal, and Sort) benefit from the NUMA (also $NUMA_{cache}$) model on KNL. This is different from what is observed on TILE-Gx72 from Figure 3. It is easy to understand why performance with the UMA model is better than NUMA on KNL. This is because the virtual LLC is larger for a UMA_{cache} processor than for a $NUMA_{cache}$ one. We take sp.C.x as an example to explain the performance benefit from a UMA_{cache} processor. Table 11 exhibits the MPKI values of sp.C.x, one of the nine programs that benefit from the UMA_{cache} model on KNL. The MPKI values with the UMA_{cache} model from 58 to 64 threads are lower than those with the $NUMA_{cache}$ model with/without the automatic NUMA balancing policy. Less counts of cache misses with the UMA_{cache} model exhibited in Table 11 correspond to the performance improvement shown in Figure 4ae. The observation reflects that the guides in [5] (i.e., suggesting not using KNL as a UMA (also UMA_{cache}) system) can mislead researchers and/or users.

Since the virtual LLC of the UMA_{cache} system is composed of all on-chip L2 caches, it has a two-fold effect on program performance. When a program reuses cache blocks in the near

future and/or exhibits good data locality, the future-accessed cache blocks can be kept in the larger virtual LLC of the UMA_{cache} system, and thus, the performance is improved compared with the $NUMA_{cache}$ system. When a program is aggressive toward the cache system, which means that most future-accessed cache blocks are not present in the virtual LLC, the smaller virtual LLC of the $NUMA_{cache}$ system can prevent the L2 caches of other node(s) from being polluted. Tables 9, 10, and 12 present the MPKI values for fluidanimate, canneal, and Sort on KNL, respectively. The MPKI values of fluidanimate with 32 and 64 threads with the UMA model are higher than those with NUMA with and without the automatic NUMA balancing policy. The higher MPKI values correspond to the worse performance with 32 and 64 threads in Figure 4g. Knowing that the $NUMA_{cache}$ model on KNL is able to protect L2 caches of other node(s) from being polluted, compared with the UMA_{cache} model, we can infer that fluidanimate is aggressive to the cache system. When a cache miss takes place, L2 caches of the whole system are affected for the UMA_{cache} model, but for the $NUMA_{cache}$ model, only L2 caches of the current node are influenced. That is why the MPKI values for the UMA_{cache} model are higher when the program is aggressive to the cache system. Sort exhibits similar MPKI values to fluidanimate as presented in Table 12. Contrary to fluidanimate, canneal with the UMA_{cache} model does not exhibit clearly higher MPKI values than the $NUMA_{cache}$ model, as shown in Table 10. We then can infer that the main overhead is not related to the memory system. People who want to improve performance of a canneal-like application need to find the root cause(s) from the OS, the runtime system, and the application itself.

Other programs exhibit insignificant performance comparison between UMA_{cache} and $NUMA_{cache}$ models on KNL. The automatic NUMA balancing policy does not help improve program performance on KNL compared with the first-touch policy. Canneal and dedup are exceptions, as illustrated in Figures 4c and 4d. The MPKI values shown in Table 10 for the automatic NUMA balancing policy exhibit that the performance improvement is not from the memory system. Further analysis is left for the future work. On the other hand, it demonstrates that a UMA_{cache} processor does not hurt performance compared with a $NUMA_{cache}$ one, if the program is not cache-aggressive. This observation contradicts the guides described in [5]. In summary, a compute-intensive (i.e., cg.C.x) application benefits from a UMA_{cache} processor because of a larger virtual LLC. A cache-aggressive application benefits from a $NUMA_{cache}$ processor due to that other LLCs are protected from being polluted.

C. DISCUSSION

From Section IV-B, we know that the novel models (UMA_{cache} and $NUMA_{cache}$) can explain the insignificant performance difference between the conventional UMA and NUMA models on TILE-Gx72 and the significant performance comparison on KNL. Because the novel models can help OS designers and application programmers focus on the cache system instead of both the cache system and the main-memory system, we therefore believe that it is feasible (or necessary) to redefine what a (non-)uniform memory access system should be for tiled many-core processors, especially when more and more cores are integrated onto a single chip to further improve performance. OS designers (application programmers) can treat the tiled many-core processor as a UMA_{cache} system, but they need to rethink what the OS (application) should be in order to take advantage of data locality in the cache system as much as possible. The cache replacement policy, which determines which cache block needs to be evicted before a new cache block is installed, should be reconsidered as well in order to keep the most-accessed cache blocks, i.e., the cache blocks belonging to a shared area between threads, in the cache system as long as possible. The performance analysis in Section IV shows that we are able to analyze the (non-)significant performance difference on emerging tiled many-core processors using the novel models of UMA_{cache} and $NUMA_{cache}$, with a full understanding of the underlying hardware. Since it is impractical to evaluate hundreds (or thousands) of applications on emerging tiled many-core processors to conclude that the conventional UMA (or NUMA) system is better, the novel models can be adopted to predict whether or not a given program benefits from the UMA (or NUMA) system. The performance of a program is more likely to benefit from the UMA_{cache} model on tiled many-core processors because of the larger-capacity LLC, as illustrated in Figure 4. OS designers may benefit from that feature because it seems relatively easier to design UMA_{cache}-based OSes than $NUMA_{cache}$-based ones.

However, another problem arises on tiled many-core processors if OS designers and application programmers highly rely on the cache system. It is well-studied that the performance of one application is influenced by another co-scheduled one when LLC is shared between them. One cache-aggressive application is able to evict cache lines of another co-scheduled one, even though those cache lines will be accessed by the co-scheduled application later. Therefore, how do OS designers design their own OS to schedule multiple applications when underlying caches are shared by them? One simple approach is to run them on $NUMA_{cache}$-based tiled many-core processors, divide applications into multiple groups and map each group to each node. That might be feasible if the group count is no more than the node count, and the burden placed on one specific memory controller is not a serious overhead. When the performance of one application is sensitive to the cache capacity, this simple approach degrades the performance undoubtedly. Moreover, when each group includes more than one application, the cache sharing problem of UMA_{cache}-based tiled many-core processors happens again. OS designers should be careful to design their OS on tiled many-core processors.

To alleviate the burden placed on OS designers and application programmers, and to eliminate the potential overhead of burdening one specific physical memory controller on tiled many-core processors, we propose a hybrid memory system

in this paper, though the simulation work will be completed in the future. Note that we do not differentiate cache and physical memory when discussing the hybrid system. The system is inspired by the performance comparison between conventional models and novel models discussed in this paper. It combines traditional UMA model and $NUMA_{cache}$ model, and thus, a physical page is interleaved across multiple physical memory controllers with modeled granularity (such as 512 bytes adopted by TILE-Gx36/72), and cores are clustered as a node. Furthermore, the node count is managed by software (OS/hypervisor) dynamically to facilitate the work of OS designers. The cache coherence protocol can be similar to that used by TILE-Gx36/72, and thus, a requester tile and the home tile are included in the same node. One might be concerned that this will cause the same problem for cache-capacity-intensive applications. One more aggressive idea, which is inspired by cooperative caching [37] and hybrid shared/private caching [38], is to separate the L2 cache from a tile with one bit managed by software, and not to change the physical location of each L2 cache. That is, cores from one node can access their L2 caches when bits are set to 1. When bits of some L2 caches of one node are set to 0, then these L2 caches can be borrowed by other (neighboring) node(s) on demand. Moreover, the experimental results shown in Figure 4 demonstrate that program performance degraded by on-chip network congestion is not serious compared with performance loss caused by a lower-capacity LLC. Thus, we believe that the hybrid memory system, which is supported by a virtual memory controller and a dynamic software-managed node count, is feasible for future tiled many-core processors. Further work will be completed in the future to analyze modeling parameters, such as interleaving granularity and maximum core count.

V. RELATED WORK

Because the analysis is based on previous research work, in this section, we mainly describe aspects associated with the novel models and the proposed hybrid memory system.

A. TRADITIONAL MULTICORE SYSTEM

Figure 2 exhibits an example of a multi-socket system, which is one of the typical traditional multicore systems in the HPC area, of which cores can be fitted onto multiple chips [22], [39] and the interconnect between cores on the same chip can be rings [40]. According to conventional models of UMA and NUMA, in which UMA is determined by the uniform memory access latency between cores (processors) and a memory system, and NUMA is non-uniform, the multi-socket system of Figure 2 belongs to the NUMA system. Non-uniform memory access latency is the main feature of NUMA-based traditional multicore systems. However, Dashti *et al.* [41] observed that remote access latency was not the key overhead to program performance because of reduced wire delay. They pointed out that the congestion on the memory controller and node-to-node interconnect hurt the performance more than the wire delay. Moreover, Lepers *et al.* [20] noticed that an asymmetric point-to-point interconnect mattered more than the wire delay. Diener *et al.* [42] evaluated how performance was affected by access locality and balanced memory accesses. They observed that the mixture of locality and balance was able to provide the highest performance improvement. These studies further inspired us to pay attention to the overhead from the memory controller and interconnect.

B. CACHE SYSTEM

Huh *et al.* [43] depicted the organization of private cache and shared cache with non-uniform cache architecture (NUCA)-based L2 cache. They observed that a dynamically migratory NUCA approach was able to improve program performance for a portion of workloads. Zhang and Asanovic [44] mentioned private cache (local L2 cache slice is private to a tile) and shared cache (all L2 cache slices are combined together as a large L2 cache shared by all tiles) as well. They proposed a policy called ''victim replication'' to take advantage of the advantages of private and shared caches. On the basis of shared cache, they attempted to keep a copy of a victim in a tile's local L2 cache slice when the cache line was evicted because of a conflict or lack of capacity. Chang and Sohi [37] presented cooperative caching to combine the strengths of private and shared caches as well. Their work was based on private cache but exploited L2 cache resources as sufficiently as possible via policies including cache-to-cache transfers of clean data, replication-aware data replacement, and global replacement of inactive data. Cho and Jin [38] proposed an OS-level page allocation approach to mapping data to L2 cache slices at memory page granularity. They further pointed out that L2 cache slices from other cores could be borrowed to increase the caching space. These studies motivated us to propose a more aggressive method in Section IV-C to separate L2 caches from tiles with one software-managed bit.

C. CACHE SHARING PROBLEM

Chandra *et al.* [45] observed that the number of cache misses was increased and that of instruction per cycle (IPC) was reduced, to respective degrees, when one application was co-scheduled with other applications on a system with a shared L2 cache. They proposed three performance models to investigate the impact of cache sharing on co-scheduled threads. Xie and Loh [46] proposed a new classification algorithm that predicted when cache sharing problems might appear on a multi-core processor. Kim *et al.* [47] observed an increase in the number of cache misses and a reduction in IPC of one application when it was co-scheduled with other applications as well. They proposed five metrics to measure the degree of cache sharing fairness, and found that optimizing fairness mattered more than maximizing throughput on co-scheduled benchmark pairs. These studies motivated us to propose a hybrid memory system to dynamically manage tiled many-core processors when multiple applications are co-scheduled.

VI. CONCLUSION AND FUTURE WORK

We define novel models (UMA_{cache} and $NUMA_{cache}$) that are based on the cache coherence protocol on tiled many-core processors. The UMA_{cache} model is determined when one physical page is distributed across all available on-chip tiles with the purpose of maintaining cache coherence, while the $NUMA_{cache}$ model is recognized by distributing a physical page across a portion of on-chip tiles. With the novel models, we conclude that KNL belongs to UMA_{cache} and $NUMA_{cache}$ systems, and TILE-Gx72 belongs to the UMA_{cache} system, though both tiled many-core processors are composed of on-chip tiles interconnected via mesh-based networks. Experimental results demonstrate that the novel models of UMA_{cache} and $NUMA_{cache}$ can explain the observed (non-)significant performance difference between conventional UMA and NUMA models for tiled many-core processors. Moreover, we propose a hybrid memory system, the goal of which is to solve the cache sharing problem when multiple applications are co-scheduled, for future tiled many-core processors. Because (future) tiled many-core processors are more complicated and distinct from traditional multicore systems, we believe that the analysis in this paper will help OS designers rethink what the organization of an OS on tiled many-core processors should be. Well-designed OSes on tiled many-core processors can further facilitate the work of application programmers, and/or motivate the design of new programming model(s). In the future, simulation work will be done for the hybrid memory system on tiled many-core processors. Parameters such as interleaving granularity will be modeled, and the sensitivity to varied parameters will be analyzed.

REFERENCES

[1] A. Sodani *et al.*, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, Mar./Apr. 2016.
[2] (2015). *TILE-Gx36 Porcessor*. [Online]. Available: http://www.mellanox.com/related-docs/prod_multi_core/PB_TILE-Gx36.pdf
[3] (2015). *TILE-Gx72 Porcessor*. [Online]. Available: http://www.mellanox.com/related-docs/prod_multi_core/PB_TILE-Gx72.pdf
[4] A. Baumann *et al.*, "The multikernel: A new OS architecture for scalable multicore systems," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ.* New York, NY, USA: ACM, 2009, pp. 29–44.
[5] (2016). *Clustering modes in knights landing processors*. [Online]. Available: https://colfaxresearch.com/knl-numa/
[6] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2006, pp. 423–432.
[7] A. Fedorov, M. Seltzer, and M. D. Smith, "Improving performance isolation on chip multiprocessors via an operating system scheduler" in *Proc. 16th Int. Conf. Parallel Archit. Compilation Techn.*, Sep. 2007, pp. 25–38.
[8] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *Proc. IEEE 14th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2008, pp. 367–378.
[9] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm, "RapidMRC: Approximating L2 miss rate curves on commodity systems for online optimizations," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 1, pp. 121–132, 2009.
[10] X. Zhang, S. Dwarkadas, and K. Shen, "Towards practical page coloring-based multicore cache management," in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, 2009, pp. 89–102.
[11] D. Bradford, S. Chinthamani, J. Corbal, A. Hassan, K. Janik, and N. Ali. (2017). *Knights Mill: Intel XEON Phi Processor for Machine Learning*. [Online]. Available: https://www.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.21-Monday-Pub/HC29.21.40-Processors-Pub/HC29.21.421-Knights-Mill-Bradford-Intel-APPROVED.pdf
[12] I. Cutress. (2017). *Intel Lists Knights Mill Xeon Phi on ARK: Up to 72 cores at 320W with QFMA and VNNI*. [Online]. Available: https://www.anandtech.com/show/12172/intel-lists-knights-mill-xeon-phi-on-ark-up-to-72-cores-at-320w -with-qfma-and-vnni
[13] (2018). *Knights Mill-Microarchitectures-Intel*. [Online]. Available: https://en.wikichip.org/wiki/intel/microarchitectures/knights_mill
[14] P. Kennedy. (2017). *Intel Xeon Phi Knights Mill for Machine Learning*. [Online]. Available: https://www.servethehome.com/intel-knights-mill-for-machine-learning/
[15] C. Byun *et al.* (2017). "Benchmarking data analysis and machine learning applications on the Intel KNL many-core processor." [Online]. Available: https://arxiv.org/abs/1707.03515
[16] Y. You, A. Buluç, and J. Demmel, "Scaling deep learning on GPU and knights landing clusters," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.* New York, NY, USA: ACM, 2017, p. 9.
[17] A. A. Awan, H. Subramoni, and D. K. Panda, "An in-depth performance characterization of CPU-and GPU-based DNN training on modern architectures," in *Proc. Machine Learn. HPC Environ.* New York, NY, USA: ACM, 2017, p. 8.
[18] T. Allen, C. S. Daley, D. Doerfler, B. Austin, and N. J. Wright, "Performance and energy usage of workloads on KNL and Haswell architectures," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation* (Lecture Notes in Computer Science), vol. 10724, S. Jarvis, S. Wright, and S. Hammond, Eds. Cham, Switzerland: Springer, 2018. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-72971-8_12#citeas
[19] E. Calore, A. Gabbana, S. F. Schifano, and R. Tripiccione, "Software and DVFS tuning for performance and energy-efficiency on Intel KNL processors," *J. Low Power Electron. Appl.*, vol. 8, no. 2, p. 18, 2018.
[20] B. Lepers and V. Quéma, and A. Fedorova, "Thread and memory placement on NUMA systems: Asymmetry matters," in *Proc. USENIX Annu. Tech. Conf.*, 2015, pp. 277–289.
[21] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," *Computer*, vol. 29, no. 12, pp. 66–76, Dec. 1996.
[22] D. Molka, D. Hackenberg, and R. Schöne, and W. E. Nagel, "Cache coherence protocol and memory performance of the intel haswell-ep architecture," in *Proc. 44th Int. Conf. Parallel Process. (ICPP)*, Sep. 2015, pp. 739–748.
[23] (2009). *An Introduction to the Intel Quickpath Interconnect*. [Online]. Available: https://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html
[24] K. David, "The common system interface: Intel's future interconnect," 2007. [Online]. Available: https://www.realworldtech.com/common-system-interface/
[25] S. Kottapalli, H. G. Neefs, R. Pal, M. K. Arora, and D. Nagaraj, "Extending a cache coherency snoop broadcast protocol with directory information," 2012. [Online]. Available: https://patentimages.storage.googleapis.com/58/a6/a1/22988f4452fe56/US8656115.pdf
[26] F. Petrini, D. J. Kerbyson, and S. Pakin, "The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q," in *Proc. ACM/IEEE Conf. Supercomput.*, Nov. 2003, p. 55.
[27] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick, "System noise, OS clock ticks, and fine-grained parallel applications," in *Proc. 19th Annu. Int. Conf. Supercomput.* New York, NY, USA: ACM, 2005, pp. 303–312.
[28] A. Nataraj, A. Morris, A. D. Malony, M. Sottile, and P. Beckman, "The ghost in the machine: Observing the effects of kernel operation on parallel application performance," in *Proc. ACM/IEEE Int. Conf. Supercomput.* New York, NY, USA: ACM, 2007, p. 29.
[29] K. B. Ferreira, P. Bridges, and R. Brightwell, "Characterizing application sensitivity to OS interference using kernel-level noise injection," in *Proc. ACM/IEEE Conf. Supercomput.* Piscataway, NJ, USA: IEEE Press, 2008, p. 19.
[30] D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *Proc. IEEE Int. Symp. High-Perform. Comp Archit.*, Feb. 2012, pp. 1–12.
[31] A. Ros, M. E. Acacio, and J. M. Garcia, "A direct coherence protocol for many-core chip multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 12, pp. 1779–1792, Dec. 2010.

[32] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Architectures Compilation Techn.* New York, NY, USA: ACM, 2008, pp. 72–81.
[33] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Sep. 2008, pp. 47–56.
[34] D. H. Bailey *et al.*, "The NAS parallel benchmarks," *Int. J. High Perform. Comput. Appl.*, vol. 5, no. 3, pp. 63–73, 1991.
[35] L. Wang *et al.*, "Bigdatabench: A big data benchmark suite from Internet services," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*. IEEE, 2014, pp. 488–499.
[36] (2014). *Automatic Non-Uniform Memory Access (NUMA) Balancing*. [Online]. Available: https://doc.opensuse.org/documentation/leap/tuning/html/book.sle.tuning/cha.tuning.numactl.html
[37] J. Chang and G. S. Sohi, *Cooperative Caching for Chip Multiprocessors*, vol. 34, no. 2. New York, NY, USA: ACM, 2006.
[38] S. Cho and L. Jin, "Managing distributed, shared L2 caches through OS-level page allocation," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2006, pp. 455–468.
[39] Z. Majo and T. R. Gross, "Memory system performance in a NUMA multicore multiprocessor," in *Proc. 4th Annu. Int. Conf. Syst. Storage*. New York, NY, USA: ACM, 2011, p. 12.
[40] D. Molka, D. Hackenberg, and R. Schöne, "Main memory and cache performance of Intel sandy bridge and AMD bulldozer," in *Proc. Workshop Memory Syst. Perform. Correctness*. New York, NY, USA: ACM, 2014, p. 4.
[41] M. Dashti *et al.*, "Traffic management: A holistic approach to memory placement on NUMA systems," *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 381–394, 2013.
[42] M. Diener, E. H. M. Cruz, and P. O. A. Navaux, "Locality vs. Balance: Exploring data mapping policies on NUMA systems," in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process. (PDP)*, Mar. 2015, pp. 9–16.
[43] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA substrate for flexible CMP cache sharing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 8, pp. 1028–1040, Aug. 2007.
[44] M. Zhang and K. Asanovic, "Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors," *ACM SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 336–345, 2005.
[45] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *Proc. 11th Int. Symp. High-Perform. Comput. Architectur (HPCA)*, Feb. 2005, pp. 340–351.
[46] Y. Xie and G. H. Loh, "Dynamic classification of program memory behaviors in CMPs," in *Proc. 2nd Workshop Chip Multiprocessor Memory Syst. Interconnects*, 2008, pp. 1–9.
[47] S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in *Proc. 13th Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2004, pp. 111–122.

YE LIU received the M.S. degree from the University of Chinese Academy of Sciences, in 2013. She is currently pursuing the Ph.D. degree with the Graduate School of Information Science, Nagoya University. Her research interests include operating systems and memory systems.

SHINPEI KATO received the B.S., M.S., and Ph.D. degrees from Keio University, in 2004, 2006, and 2008, respectively. He was with Carnegie Mellon University, and with the University of California at Santa Cruz, from 2009 to 2012. He is currently an Associate Professor with the Graduate School of Information Science and Technology, The University of Tokyo. His research interests include operating systems, real-time systems, and parallel and distributed systems.

MASATO EDAHIRO received the Ph.D. degree in computer science from Princeton University, Princeton, NJ, USA, in 1999. He joined NEC Corporation, in 1985, and was with its research center for 26 years, and moved to Nagoya University, Nagoya, Japan, in 2011. His research interests include graph and network algorithms and software for multi- and many-core processors.