IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 13, 2018, accepted November 12, 2018, date of publication November 15, 2018, date of current version April 19, 2019.

*Digital Object Identifier 10.1109/ACCESS.2018.2881460*

# Capability for Multi-Core and Many-Core Memory Systems: A Case-Study With Xeon Processors

**YUXUAN XING[1], FANG LIU[2], NONG XIAO[1,2], ZHIGUANG CHEN[2], AND YUTONG LU[2]**
[1]College of Computer, National University of Defense Technology, Changsha 410073, China
[2]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China

Corresponding author: Fang Liu (liufang25@mail.sysu.edu.cn)

This work was supported in part by the National Key R&D Program of China under 2016YFB1000302, in part by the National Natural Science Foundation of China under Grant 61433019, Grant U1435217, and Grant U1611261, and in part by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2016ZT06D211.

**ABSTRACT** The general volume of data has exploded to unimaginable levels in the past decade. Therefore, the big data analytics has become an area of focus. Many frameworks have been developed for data analytics, such as Hadoop, Spark, etc. Most of the frameworks are built on multi-core or many-core memory systems, requiring developers and users to have an in-depth understanding of the architectures to take full advantage of the hardware. In this paper, we present a comprehensive study of both multi-core and many-core memory systems and discuss the different characteristics, including core, cache, memory, and the on-chip network. Furthermore, we propose a simple but effective mechanism for cache false-sharing overhead that can reduce a large number of LLC-load/store instructions and LLC cache-misses. In addition, we conduct detailed experiments with Ligra, a graph analytics framework, on four different-sized datasets. The results show that it can achieve up to a 2.5× and 9.5× speed-up for multi-core and many-core memory systems, respectively. We finally share our key findings and discuss the platform development on multi-core and many-core memory systems.

**INDEX TERMS** Data analytics, multi-core, many-core, cache false-sharing.

## I. INTRODUCTION

There has been an increasing need to process large-scale data efficiently for valuable information in both academic and industrial communities. Many popular frameworks have been developed for data analytics, such as Hadoop [1], Spark [2], GraphLab [3], etc. Most of the frameworks are built on commercial machines with multi-core or many-core processors, such as the Intel Xeon-E series and the Intel Xeon Phi series. These processors always have complex mesh connect, cache, and memory hierarchies. There are many differences in the architecture characteristics between multi-core and many-core memory systems. To make full use of the hardware resources, developers and programmers must exploit the full capabilities of the memory systems. However, users who want to analyze system performance are often faced with a lack of detailed documentation. Thus, we propose the use of a system of micro-benchmarks to capture the characteristics of multi-core and many-core memory systems; this system can express the features of architecture analytically so that they can be used along with the application requirement models to thoroughly analyze performance. To demonstrate the methodology, we develop an extensive memory capability model for two memory systems: the xeon-server is based on the Intel Xeon E5-2692v2 multi-core processor, and the knl-server is based on the Intel Xeon Phi 7210 many-core processor. The Xeon E5-2692v2 is also deployed on the Tianhe-2 supercomputer [19], and the Trinity supercomputer is based on the Xeon Phi 7250 (an updated version of the 7210) processor [20].

The main contributions of this paper are:

- We derive and parametrize capability models for the memory systems of multi-core and many-core processors. Then, we present a complex comparison regarding the architecture characteristics of the models.
- We put forward the cache false-sharing issue which is often overlooked by developers. Further, we propose an effective mechanism to reduce the overhead of false-sharing for multi-core and many-core memory systems.

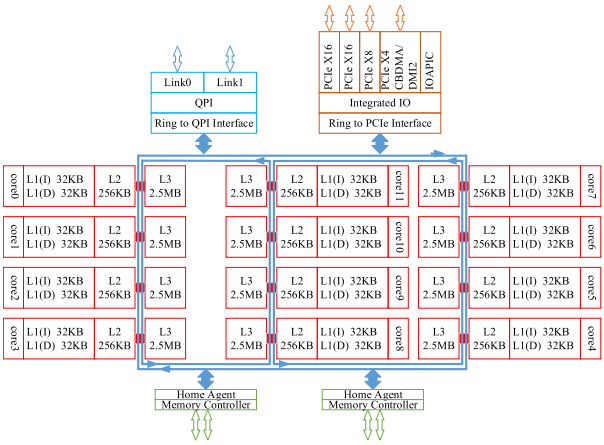


**FIGURE 1.** Xeon-E5 Architecture.

The graph analytics experiment results show that our method can achieve up to a 2.5× and 9.5× increase in speed for multi-core and many-core memory systems, respectively.

- We summarized our key findings and discussed the platform development on multi-core and many-core memory systems.

## II. MULTI-CORE AND MANY-CORE ARCHITECTURE

We used the Intel Xeon E5-2692v2 and the Intel Xeon Phi 7210 to demonstrate our methodology, and we refer to them as Xeon-E5 and KNL (Knights Landing), respectively, in the following sections.

As shown in Figure 1, the Ivy Bridge-based Xeon-E5 has 12 cores and is clocked at 2.2 GHz, with a peak performance of 211.2 Gflops. Each core has 64 KB of L1 cache (32 KB data and 32 KB instruction), and 256 KB of L2 cache. All twelve cores share 30 MB of last level cache (2.5 MB ∗ 12), also called L3 cache. The on-chip memory controller supports four DDR3 channels and 768 GB of the maximum capacity. The Xeon E5 processor has two QPI links that can connect with other processors to form a non-uniform-memory access architecture [18]. The QPI link runs at 8 GT/s, and 2 bytes can be transferred in each direction. Each link runs at 16 GB/s simultaneously, for an aggregate of 32 GB/s. KNL is a new ×86-based many-core processor; its predecessor is the Knights Corner (KNC), which is well known as the Xeon Phi. One of the major changes regarding KNC is that KNL is shipped not only as a PCIe accelerator but also as a standalone processor. The architecture of KNL is shown in Figure 2. The KNL used in our experiments has 64 tiles and two cores on each tile and is clocked at 1.3 GHz, providing a peak performance of 5.3 Tflops. Each core has 64 KB of L1 cache the same as the Xeon-E5. Two cores on one tile share 1 MB of L2 cache and the tiles are connected into a 2D mesh that provides cache coherence between the L2 caches. The L2 cache is the last level cache for KNL. The KNL processor has six DDR4 channels with 384 GB of maximum capacity, as well as eight MCDRAM controllers

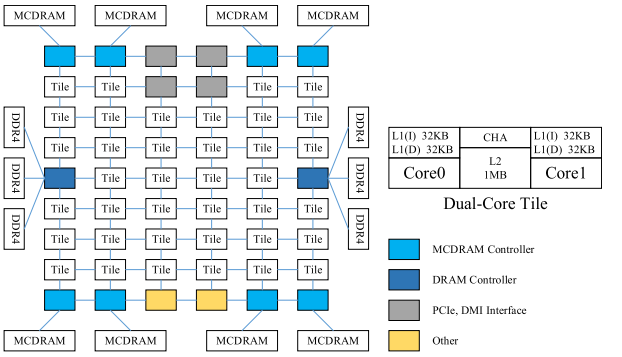


**FIGURE 2.** KNL Architecture.

**TABLE 1.** Benchmark results.

| Parameters \ Processors | | | Xeon-E5 | KNL |
|---|---|---|---|---|
| # of cores | | | 12 | 64*2 |
| Frequency (GHz) | | | 2.2 | 1.3 |
| Interconnection | | | Bidirectional Ring Bus | 2D Mesh |
| Cache | L1 | Size (KB) | 64 | 64 |
| | | Latency (ns) | 1.67 | 3.10 |
| | L2 | Size (KB) | 256 | 1024*64 |
| | | Latency (ns) | 4.01 | 13.20 |
| | L3 | Size (MB) | 2.5*12 | - |
| | | Latency (ns) | 15.39 | - |
| MCDRAM | Bandwidth (GB/s) | All Read | - | 336.56 |
| | | 1:1 R-W | - | 406.17 |
| | | Stream-traid | - | 423.51 |
| | Latency (ns) | | - | 173.1 |
| | Number of Channels | | - | 8 |
| DRAM | Bandwidth (GB/s) | All Read | 40.44 | 78.67 |
| | | 1:1 R-W | 34.02 | 77.63 |
| | | Stream-traid | 36.16 | 76.13 |
| | Latency (ns) | | 77.2 | 141.0 |
| | Number of Channels | | 4 | 8 |

"-" There is no L3 cache on KNL or MCDRAM on Xeon-E5.
Stream-triad is similar to the operation: a[i]=b[i]+s*c[i].

that each connect to a 2 GB MCDRAM. Furthermore, KNL provides three memory models and five configuration modes, making a total of fifteen configurations [4].

We conduct our experiments on two machines. The xeon-server is configured with two Xeon-E5 processors and 128 GB of memory that runs RedHat with a 2.6.32 kernel. The knl-server is configured with a Knight Landing processor, 16GB MCDRAM, and 96 GB of memory that runs CentOS with a 3.10.0 kernel. We configure the knl-server as an all-to-all cluster using flat memory mode which is the most common pattern.

We use ccbench [5] to measure the latency of cache access and Intel Memory Latency Checker [6] to check the bandwidth and latency of memory for both the Xeon-E5 and KNL processors. Table 1 shows the benchmark results in detail. KNL has a total of 128 cores, which is almost ten times more than that of the Xeon-E5 processor. However, the frequency of KNL is much lower.

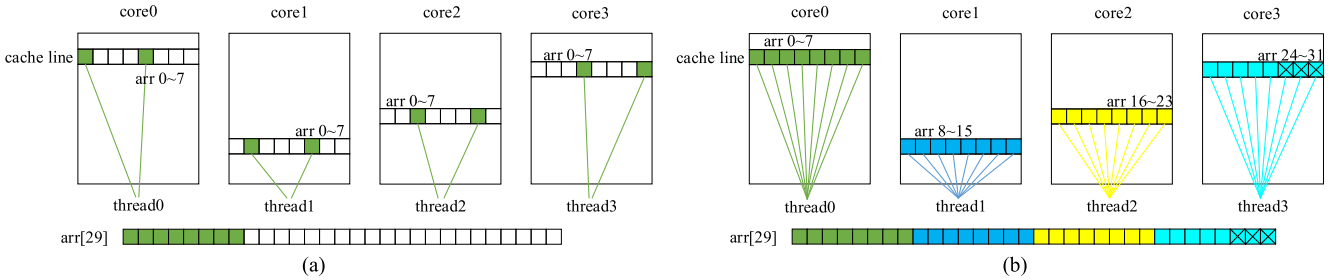


**FIGURE 3.** The Procedure of an Array Update.

One of the most important differences between Xeon-E5 and KNL is the cache hierarchy and connection. Due to the larger number of cores, the tiles in KNL are connected into a 2D mesh to provide cache coherence between the L2 caches. Comparatively, all cores in the Xeon-E5 processor have a private path to the L3 cache. Additionally, Xeon-E5 has a bidirectional ring interconnect that connects the 12 cores, the L3 cache, the QPI agent, and the integrated memory controller. Both of the processors have a 64 KB L1 cache. However, the latency of KNL is nearly twice that of Xeon-E5. The last level cache for KNL is the L2 cache, which is 64 MB spread over 64 tiles, whereas each core within the Xeon-E5 processor has a 2.5 MB L3 cache, which amounts to a 30 MB last level cache. Each core on Xeon-E5 can directly load/store any data in the whole last level cache through the ring bus, but the core on KNL must transfer data stored in other L2 caches to its local L2 cache for operations. This may not only lead to large amount of messaging, but also reduce the valid capacity of last level cache. In the worst situation, each L2 cache on all tiles contains the same data and the practical last level cache turns into 1.5 MB. Additionally, the latency of last level cache for both processors is very close.

KNL presents a heterogeneous memory hierarchy with MCDRAM and DRAM [10]. The bandwidth of MCDRAM can reach almost 400 GB/s, which is much higher than DRAM; MCDRAM also has a little longer latency. The KNL processor has two more DDR channels than Xeon-E5 and can achieve a much higher bandwidth. However, the memory access latency for KNL is nearly twice that of Xeon-E5. It seems that the more memory channels, the higher bandwidth, and the longer latency. Most importantly, KNL has a larger number of cores, lower frequency, larger cache capacity, a more complex interconnect, and a higher memory bandwidth and latency than Xeon-E5.

## III. MOTIVATION EXAMPLE

Data processing engines usually support parallel computing for fully exploiting the hardware potential. Cache false-sharing is a well-study issue in the architecture world [25], [26]. And, there have been several efforts over the years to address the problem: profiling and manual tuning the application, compiler techniques, and data layout transformations [27]. However, the cache false-sharing issue is often overlooked by developers in system development and it can't be addressed by the compiler automatically in many situations. Thus, it may become a bottleneck for big data analytics on multi-core or many-core memory systems.

For example, suppose a processor with four cores initializes four threads to parallel update an array of 29 lengths, and each element takes up 8 bytes of memory space. Figure 3(a) shows one 64-byte cache line in cores that can contain 8 array elements and the four threads parallel update the continuous array space. In this situation, the updates caused by each thread will lead to cache line failures in other cores, and extra last level cache (LLC) load/store instructions will be needed for cache coherence (details is discussed in section V). This outcome may trigger not only excessive load/store instructions and memory accesses but also congestion on interconnects and the memory controller. The more threads there are, the more serious the situation becomes.

A better update strategy is shown in Figure 3(b). In this strategy, there are no shared data in the cache lines of the different cores, and we limit the effect of element updates in the core where the thread was initialized. The optimization methods will achieve good performance improvement for highly parallel applications on both multi-core memory systems and on many-core memory systems. This phenomenon is prevalent in the applications of big data analytics, and it is often ignored by the developer. But, the he parallel computing is one of the key steps, in general. We can avoid or reduce the overhead of cache false-sharing through appropriate task partitioning and scheduling mechanism that may lead to significant performance improvements.

## IV. EXPERIMENTS SETUP

To verify the overhead of cache false-sharing in big data analytics, we select Ligra which is a popular graph analytic framework [13], and compare the performance behavior after the optimized for cache false-sharing on both multi-core and many-core memory systems.

Graph structures provide a basic model of entities with connections between them that can represent almost anything [9]. Graph analytics has been widely adopted in various big data applications such as social computation, web search, and recommendation systems. Ligra is a lightweight graph

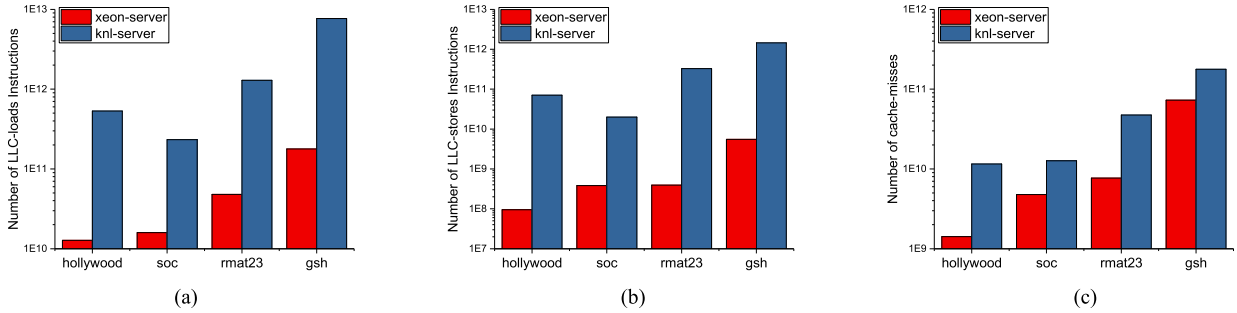


**FIGURE 4.** PR on xeon-server and knl-server. (a) LLC-loads. (b) LLC-stores. (c) cache-misses.

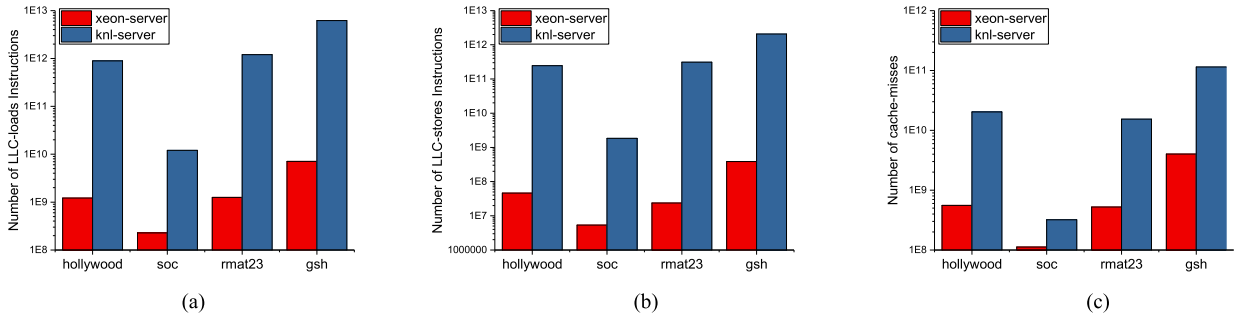


**FIGURE 5.** TC on xeon-server and knl-server. (a) LLC-loads. (b) LLC-stores. (c) cache-misses.

processing framework that is specific to the shared-memory multi-core memory system. We can efficiently implement many graph applications with the interfaces provided by Ligra, such as PageRank [14], Triangle Counting [15], BFS [16], Betweenness Centrality [17], etc. The computation is done by iteratively calling the VertexMap and EdgeMap functions: VertexMap applies the application-define function to all vertices in the active set; EdgeMap applies the application-define function to all edges whose source vertex belong to the active vertex set. For most real-world graphs, the number of edges is several times larger than vertices. Thus, the EdgeMap function takes up the main computation time in graph analytics. One key step in EdgeMap function is parallel updating the vertex values by all threads that is very similar to the situation in Figure 3(a).

Data layout transformation is one of the useful methods to reduce the overhead of cache false-sharing and has been widely adopted by developers. Ligra implements multi-thread parallel processing through Clik/OpenMP just like any other platforms [21], [22]. For OpenMP, we can get rid of cache false-sharing using its inherent scheduling polices. For instance, it takes up 8 bytes for each vertex value needs to be updated, while one cache line is 64 bytes that can contain 8 vertex values, and it is better that one thread updates 8 sequential vertex values. This can be realized by setting a parallel granularity to 8 using the interface of OpenMP. As for Cilk/Cilk++, it adopts the bisection method to automatically set the parallel granularity for performance. But, we can control the granularity by adding extra invalid elements. Suppose we initialize four threads on four cores and parallel update an array of 52 elements. In general, we set the parallel granularity to 8, the task partition will be <7, 6, 7, 6, 7, 6, 7, 6> for the four work threads. The update operations of one thread will cause the cache line failures of other threads. If we increase the number of array elements to 64 by adding invalid elements, the task partition will be <8, 8, 8, 8, 8, 8, 8, 8> that can reduce the overhead of cache false-sharing. And, we just need to skip the artificially added elements without any operations in the processing. We optimize Ligra using this method and compare with the native Ligra on multi-core memory system (xeon-server) and many-core memory system (knl-server), respectively.

## V. EVALUATION

The experiments are run individually on the xeon-server and knl-server. The configuration of machines is described in Section II. We use both synthetic and real-world graph datasets for performance measurement: Hollywood and soc are social graphs, gsh is a web graph, and rmat23 is generated using the RMAT generator with an average degree of 16 (as recommended by the Graph500 benchmark) [23]. RMAT graphs have a scale-free property that is a feature of many real-world graphs [24]. We focus in this section on the performance impact of cache false-sharing and the different behaviors between multi-core and many-core memory systems.

### A. OVERALL PERFORMANCE

We select two typical graph applications realized by Ligra. PageRank (PR) was first proposed and used by Google. PR pulls values from the vertex's neighbors to update the vertex PageRank value in parallel. Triangle Counting (TC)

**TABLE 2.** A collection of graphs.

| Datasets | Vertices | Edges | Description |
|---|---|---|---|
| hollywood[8] | 2.18 M | 0.23 B | hollywood-2011 social graph |
| soc[7] | 4.85 M | 68.99 M | livejournal social graph |
| rmat23 | 8.34 M | 0.13 B | generated graph |
| gsh[8] | 68.66 M | 1.80 B | gsh web graph |

**TABLE 3.** Runtimes (in seconds).

| Application \ Machine | | xeon-server | | | knl-server | | |
|---|---|---|---|---|---|---|---|
| | | Na | Op | **Sp** | Na | Op | **Sp** |
| PR | hollywood | 7.1 | 5.16 | **1.37** | 28.6 | 8.65 | **3.31** |
| | soc | 8.99 | 9.2 | **0.98** | 13.5 | 8.76 | **1.54** |
| | rmat23 | 25 | 22.5 | **1.11** | 103 | 51.6 | **2** |
| | gsh | 155 | 141 | **1.10** | 483 | 135 | **3.58** |
| TC | hollywood | 30.5 | 12.2 | **2.5** | 104 | 12.7 | **8.19** |
| | soc | 0.546 | 0.412 | **1.33** | 1.12 | 0.373 | **3** |
| | rmat23 | 26.2 | 11.6 | **2.26** | 115 | 48.6 | **2.37** |
| | gsh | 129 | 59.6 | **2.16** | 619 | 64.9 | **9.54** |

'Na' is the native Ligra performance, 'Op' is the optimized Ligra performance and 'Sp' is short for 'speed-up ratio'.

computes a vertex's edge list and the neighbors of the vertex on its edge list for triangles. Both of them need parallel update vertex PageRank/Triangle value during the computation, and the update takes up most of the processing time. The results are shown in Table 3, and we found that:

1) The native Ligra (Na-Ligra) runs faster on xeon-server than knl-server while it didn't appear to be much difference for optimized Ligra (Op-Ligra).
2) Overall, the Op-Ligra behaves better than Na-Ligra on both xeon-server and knl-server. The optimization effect is more obvious on knl-server than on xeon-server for both PR and TC, and TC has achieved greater performance improvement relative to PR on the two servers.

### B. RESULTS ANALYSIS

It is well known that both the CPU capacity and memory access have a significant impact on performance. The xeon-server has two Xeon-E5 sockets, a peak performance of 211.2 Gflops for each socket. While the knl-server has a Xeon Phi 7250 processor, providing a peak performance of 5.3 Tflops. Though the knl-server has more computing power, the Na-Ligra spends more time on knl-server than on xeon-server for both PR and TC computation. Therefore, we count the number of last level cache load/store instructions (LLC-loads and LLC-stores) and the number of last level cache failures (cache-misses) generated during the calculation for further analysis. Figure 4 and 5 show the records about the three factors for PR and TC, respectively. In general, the number of LLC-loads, LLC-stores and cache-misses increase with the size of graphs. The calculation on knl-server produces several times larger number of load/store instructions relative to xeon-server, and more cache failures appear accordingly. The extra instructions may be caused by the

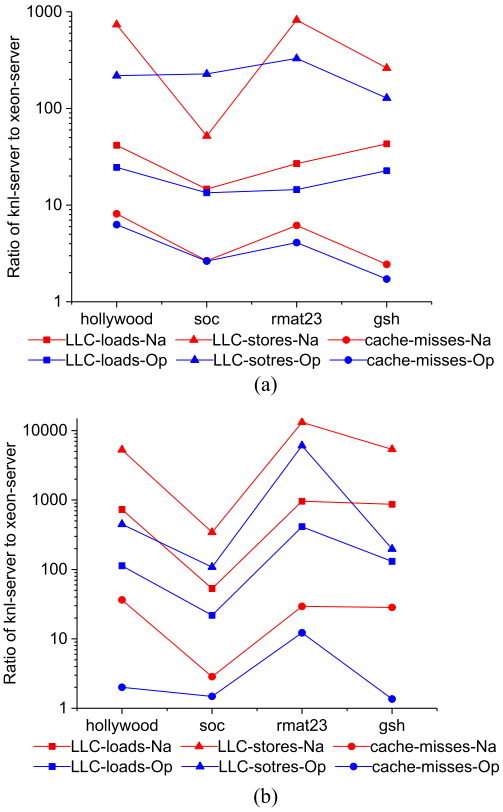


**FIGURE 6.** The gap between xeon-server and knl-server. LLC-loads-Na is for Na-Ligra while LLC-loads-Op is for Op-Ligra, and so are the others. (a) PR. (b) TC.

different cache hierarchy: Xeon-E5 has three levels of cache while KNL only has two levels of cache, and the L2 cache has made the great contribution to the last level cache accesses reduction. As for the cache failures, it increases as the number of load and store instructions increases. In addition, KNL has a total of 64 MB last level cache (64 tiles and 1 MB LLC for each tile), each LLC is more private to its own tile which means one thread must transform the needed data from other LLC or memory to its local LLC. In extreme cases, each LLC in 64 tiles contains the same data which is equivalent to just 1 MB LLC. Correspondingly, threads on Xeon-E5 can access the whole 30 MB L3 cache directly that can be very helpful to reduce the number of cache failures. The result of cache failures will lead to extra memory accesses. It is more time consuming since memory latency is always several times longer than caches on the processor (see Table 1). Above all, the differences in cache hierarchy and interconnect result in the differences in the number of LLC-loads/stores and cache-misses, which in turn affect the performance on xeon-server and knl-server.

Figure 6 shows the gap between xeon-server and knl-server for both native and optimized Ligra. The Y-axis is the ratio of knl-server to xeon-server about the numbers of LLC-loads, LLC-stores, cache-misses. For example, the number of LLC-loads instructions generated on the knl-server is 4 while on the xeon-server is 2, then the ratio

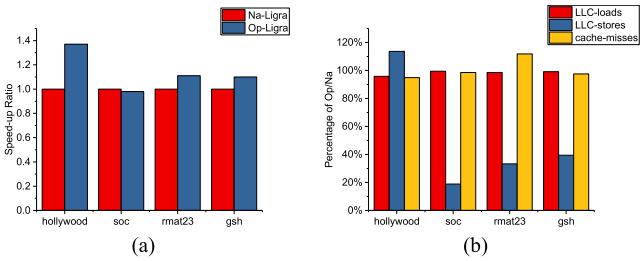


**FIGURE 7.** PR on xeon-server. (a) Speed-up Ratio of Op/Na-Ligra. (b) Percentage of Op/Na-Ligra.

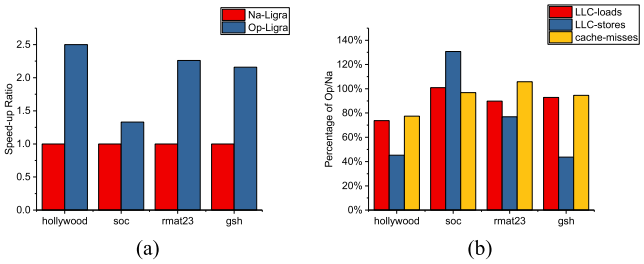


**FIGURE 8.** TC on xeon-server. (a) Speed-up Ratio of Op/Na-Ligra. (b) Percentage of Op/Na-Ligra.

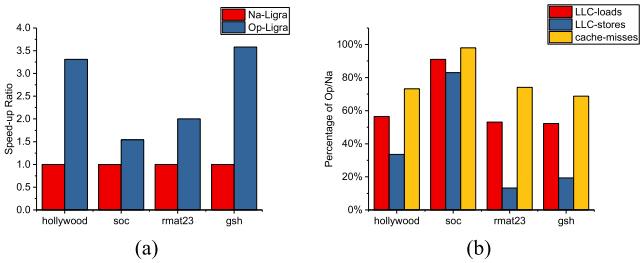


**FIGURE 9.** PR on knl-server. (a) Speed-up Ratio of Op/Na-Ligra. (b) Percentage of Op/Na-Ligra.

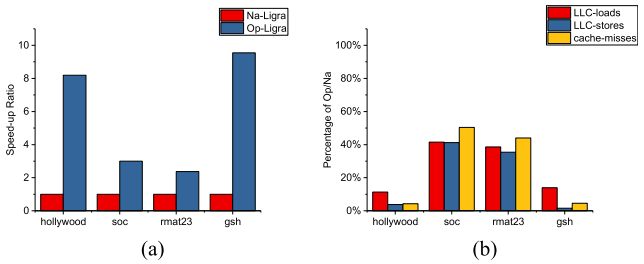


**FIGURE 10.** TC on knl-server. (a) Speed-up Ratio of Op/Na-Ligra. (b) Percentage of Op/Na-Ligra.

is $4/2 = 2$. The larger the ratio, the larger the performance gap between xeon-server and knl-server. Apparently, the gap is narrowing after our optimization for the above three parameters (except LLC-loads-Op on soc graph). Due to the greater computing power of the KNL, Op-Ligra has not much difference in performance between xeon-server and knl-server.

We also compare the characteristics of Op-Ligra with Na-Ligra on both xeon-server and knl-server. The results are present in Figure 7, 8, 9 and 10. The subgraph(a) shows the performance improvement ratio for Op-Ligra relative to Na-Ligra, and the subgraph(b) shows the percentage of Op-Ligra to Na-Ligra about the number of LLC-loads/stores instructions and cache-misses: the smaller the percentage, the larger the decrease. Overall, the Op-Ligra generated fewer LLC-loads and LLC-stores instructions during the entire computation, and the number of cache-misses was also greatly reduced. This can not only ease the congestion on interconnects and controllers but also reduce memory access. Thus, the Op-Ligra achieves great performance improvements.

Through the comparison between the four figures, we found that the smaller the percentage, the larger the speed-up ratio. TC on knl-server achieves the maximum performance improvement while PR on xeon-server is the minimum. Thanks to the three levels of cache and the bi-directional ring interconnect on Xeon-E5, the L2 cache can effectively reduce the LLC-load/store instructions. And the ''big'' shared LLC also plays an important role in increasing the cache-hit rate. Thus, our optimization for cache false-sharing has less effect on xeon-server than on knl-server. Besides, TC needs more updates on each vertex value every iteration so that it gets more benefits from the optimization. Furthermore, the acceleration effect seems to be more relevant to the LLC-loads rather than LLC-stores in Figure 7. Give that the number of LLC-load instructions is always much larger than LLC-store instructions, reducing the number of LLC-load instructions has a significant impact on improving performance.

### C. DISCUSSION

The architecture of multi-core and many-core is much different, it is very important for developers to understand their characteristics in order to design efficient programs. We have made a comprehensive comparison about them by taking an example of xeon-E5 and KNL. And we put forward the cache false-sharing issue which is easy to overlook by many programmers, and we also make detailed analytics about the performance on graph processing.

The many-core memory systems can provide a larger amount of lower frequency cores relative to multi-core memory systems, and they seem to prefer higher memory bandwidth but longer memory latency.

The cache false-sharing exits in many big data analytics, and addressing this issue contributes to fully develop parallelism and performance improvement.

The many-core memory systems are more sensitive to data dependency due to their complex interconnection among cores and two-level cache hierarchy. Developers should pay more attention to the issues that may cause data consistency, such as parallel granularity, task scheduling, context data structures and so on.

## VI. RELATED WORKS

Previous research is primarily focused on either multi-core or many-core memory systems. Ramos and Hoefler [10]

developed an intuitive performance model for cache coherent many-core architectures and provided several optimal and optimized algorithms for complex parallel data exchanges. Saini *et al.* [11] presented a performance evaluation of Pleiades based on the Intel Xeon E5-2670 processor and conducted detailed experiments using several low-level benchmarks, and four full-scale scientific and engineering applications. Ramos and Hoefler [12] also derived systematic benchmarking methods to select relevant parameters for capability models of memory subsystems. The built models can rigorously analyze the performance of many applications. However, none of the built models have attempted to complete a detailed comparison of the multi-core and many-core memory systems and provide optimization techniques based on the respective architectures.

## VII. CONCLUSION

Multi-core and many-core processors have been widely deployed in data centers and supercomputing centers. Through our experiments, we derive and parametrize capability models for multi-core and many-core memory systems and compare their architecture characteristics. Additionally, we propose an optimization mechanism for reducing the overhead of cache false-sharing caused by the parallel updates, and we utilize Ligra to verify the effectiveness of our approach and analyze the different influence of cache false-sharing on multi-core and many-core memory systems. Furthermore, our strategy can be widely adopted by other parallel frameworks for big data analytics.

## ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments.

## REFERENCES

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. MSST*, Incline Village, NV, USA, 2010, pp. 1–10.
[2] M. Zaharia, M. Chowdhury, M. J. Franklin, and S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. HotCloud*, Boston, MA, USA, 2010, pp. 1-10.
[3] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.
[4] V. Codreanu, J. Rodriguez, and O. W. Saastad, "Best practice guide—Knights landing," PRACE, Brussels, Belgium, Jan. 2017.
[5] C. Celio. (Feb. 2014). *ccbench: Memory System Microbenchmarks*. [Online]. Available: http://github.com/ucb-bar/ccbench
[6] V. Viswanathan, K. Kumar, and T. Willhalm. (Nov. 2013). *Intel Memory Latency Checker*. [Online]. Available: https://software.intel.com/en-us/articles/intelr-memory-latency-checker
[7] (2014). *Stanford Large Network Dataset Collection*. [Online]. Available: https://snap.stanford.edu/data/index.html
[8] (2004). *Laboratory for Web Algorithmics*. [Online]. Available: http://law.di.unimi.it/datasets.php
[9] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, and N. G. Leiserand Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. SIGMOD*, Indianapolis, IN, USA, 2010, pp. 135–146.
[10] S. Ramos and T. Hoefler, "Capability models for manycore memory systems: A case-study with Xeon Phi KNL," in *Proc. IPDPS*, Orlando, FL, USA, May 2017, pp. 297–306.
[11] S. Saini, J. Chang, and H. Jin, "Performance evaluation of the Intel sandy bridge based NASA pleiades using scientific and engineering applications," NASA, Moffett Field, CA, USA, Tech. Rep. NAS-2015-05, 2005.
[12] S. Ramos and T. Hoefler, "Modeling communication in cache-coherent SMP systems: A case-study with Xeon Phi," in *Proc. HPDC*, New York, NY, USA, 2013, pp. 97–108.
[13] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," *ACM SIGPLAN Notices*, vol. 48, no. 8, pp. 135–146, 2013.
[14] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Comput. Netw. ISDN Syst.*, vol. 30, pp. 107–117, Apr. 1998.
[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, p. 440, Jun. 1998.
[16] S. Beamer, K. Asanović, and D. Patterson, "Direction-optimizing breadth-first search," *Sci. Program.*, vol. 21, nos. 3–4, pp. 137–148, 2013.
[17] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.
[18] M. M. Michael, A. K. Nanda, B.-H. Lim, and M. L. Scott, "Coherence controller architectures for SMP-based CC-NUMA multiprocessors," in *Proc. 24th Int. Symp. Comput. Archit.*, Jun. 1997, pp. 219–228.
[19] X. Liao, L. Xiao, C. Yang, and Y. Lu, "MilkyWay-2 supercomputer: System and application," *Frontiers Comput. Sci.*, vol. 8, no. 3, pp. 345–356, 2014.
[20] (2017). *Trinity: Advanced Technology System*. [Online]. Available: https://www.lanl.gov/projects/trinity/index.php
[21] (Nov. 2015). *OpenMP Application Programming Interface, Version 4.5*. [Online]. Available: http://www.openmp.org/
[22] (2012). *Intel Cilk Plus*. [Online]. Available: https://www.cilkplus.org/
[23] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *Proc. SIAM*, 2004, pp. 442–446.
[24] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
[25] J. Torrellas, H. S. Lam, and J. L. Hennessy, "False sharing and spatial locality in multiprocessor caches," *IEEE Trans. Comput.*, vol. 43, no. 6, pp. 651–663, Jun. 1994.
[26] S. J. Baylor and Y. Hsu, "Cache coherence protocol for reducing the effects of false sharing in non-bus-based shared-memory multiprocessors," U.S. Patent 5 822 763, Oct. 13, 1998.
[27] T. E. Jeremiassen and S. J. Eggers, "Reducing false sharing on shared memory multiprocessors through compile time data transformations," in *Proc. PPoPP*, Jul. 1995, pp. 186–197.

**YUXUAN XING** received the B.S. degree in software engineering from Xidian University, Xi'an, China, in 2012, and the M.S. degree in computer science from the National University of Defense Technology, Changsha, China, in 2015. He is currently pursuing the Ph.D. degree in computer science with the National University of Defense Technology.

His research interests include high performance computing, graph processing, storage reliability, and deduplication.

Dr. Xing's awards and honors include the First-Class Scholarship (Xidian University), the Outstanding Students (National University and Defense Technology), and the International Conference on Progress in Informatics and Computing Best Paper Award in 2014.

**FANG LIU** received the B.S., M.S., and Ph.D. degree in computer science from the National University of Defense Technology, in 2005.

Her research interests include nonvolatile storage, edge computing, and deduplication.

**NONG XIAO** was born in Nanchang, China, in 1969. He received the B.S., M.S., and Ph.D. degrees in computer science from the National University of Defense Technology.

He is currently the Professor with the School of Computer Science, National University of Defense Technology. His research interests include grid computing, cloud computing, and big-data processing.

Dr. Xiao's awards and honors include the Cheung Kong Scholars Chair Professor, the National Outstanding Young Science Fund, and the first prize of the National Science and Technology Progress.

**ZHIGUANG CHEN** received the B.S. degree in computer science from the Harbin Institute of Technology, in 2007, and the M.S. and Ph.D. degrees in computer science from the National University of Defense Technology, in 2013.

His research interests include parallel operating system, global file system, and nonvolatile storage.

**YUTONG LU** was born in Harbin, China, in 1969. She received the B.S., M.S., and Ph.D. degrees in computer science from the National University of Defense Technology. She became a fellow of the ISC High Performance in 2017.

She is the Director of the National Supercomputing Center, Guangzhou, and was the Deputy Chief Designer for Tianhe-2 systems. She is also a Professor at the School of Computer Science, Sun Yat-sen University. Her extensive research and development experience has spanned several generations of domestic supercomputers in China and her continuing research interests include parallel operating system, high-speed communication, global file system, and advanced programming environment.

Dr. Lu was a recipient of the National Science and Technology Progress Special Award and the New Century Excellent Talent Support Plan in China.