Exploring HPC and Big Data Convergence
A Graph Processing Study on Intel Knights Landing

Uta, Alexandru; Varbanescu, Ana Lucia; Musaafir, Ahmed; Lemaire, Chris; Iosup, Alexandru

DOI
10.1109/CLUSTER.2018.00019

Publication date
2018

Document Version
Accepted author manuscript

Published in
2018 IEEE International Conference on Cluster Computing (CLUSTER)

Exploring HPC and Big Data Convergence: A Graph Processing Study on Intel Knights Landing

Alexandru Uta*, Ana Lucia Varbanescu‡, Ahmed Musaafir*, Chris Lemaire§, Alexandru Iosup*§
*Vrije Universiteit Amsterdam, §TU Delft, ‡University of Amsterdam, the Netherlands
{a.uta, a.a.m.musaafir, a.iosup}@vu.nl, a.l.varbanescu@uva.nl, c.lemaire@student.tudelft.nl

***Abstract*—The question "Can big data and HPC infrastructure converge?" has important implications for many operators and clients of modern computing. However, answering it is challenging. The hardware is currently different, and fast evolving: big data uses machines with modest numbers of fat cores per socket, large caches, and much memory, whereas HPC uses machines with larger numbers of (thinner) cores, non-trivial NUMA architectures, and fast interconnects. In this work, we investigate the convergence of big data and HPC infrastructure for one of the most challenging application domains, the highly irregular graph processing. We contrast through a systematic, experimental study of over 300,000 core-hours the performance of a modern multicore, Intel Knights Landing (KNL) and of traditional big data hardware, in processing representative graph workloads using state-of-the-art graph analytics platforms. The experimental results indicate KNL is convergence-ready, performance-wise, but only after extensive and expert-level tuning of software and hardware parameters.**

I. Introduction

Currently, the HPC and big data communities are not convergent [1]: they operate different types of infrastructure and run complementary workloads. As computation demand and volumes of data to be analyzed are constantly increasing, the lack of convergence is likely to become unsustainable: the costs of energy, computational, and human resources far exceed what most organizations can afford. The community has started to address the *software convergence problem*, discussing joint HPC-big data software-stacks [1], [2], [3], proposing techniques for software-integration [4], and augmenting big-data software-frameworks with efficient HPC libraries [1], [5], [6]. We envision a form of *deep convergence*, across both the software level and the infrastructure level. However, the case for deep convergence needs pragmatic arguments. Addressing the lack of performance studies for the deep convergence problem, in this work we conduct the first performance study of processing a representative workload on big-data software-platforms running on modern HPC infrastructure.

In our vision, the deep convergence of HPC and big data entails defining common software *and* hardware ecosystems, catering to the needs of both worlds. This could enable: (1) efficient operation of software and hardware, which is currently available only to the HPC community, for a wider-audience including the big data community, (2) sustainable development of new solutions, addressing the human-resource scarcities affecting both communities, and (3) innovation, especially for more efficient software-and-hardware solutions, exploiting the full capability of modern hardware, and avoiding the consequences of *lack of convergence* already observed by practitioners [5], [6], [7].

To address the convergence problem, the community focuses already on porting big-data software to HPC-infrastructure [4], or, conversely, porting and integrating HPC-software into big-data infrastructure [1], [5], [6]. These alternatives pose research and engineering challenges related to portability: not only code has to be re-written for new hardware, but also intricate software parameter tuning and customization must be performed. (The alternative explored in this work only requires the latter.) We see an emerging alternative: deploying big-data software directly to HPC-hardware. To this end, we see KNL [8], the de-facto processor for three of the top ten Top500 [9] systems, as currently an important representative of an emerging class of processors. That is, with massively parallel multicore and high-bandwidth on-chip memory, but also with full x86-compatibility, and self-booting and reconfiguration capabilities that allow supporting efficiently different workloads.

Moreover, KNL, and HPC-friendly processors (e.g., Sunway, BlueGene PowerPC) are highly representative for the top ten of the Top500, and significantly different from typical processors in big data (i.e., the Intel Xeon family). KNL is designed for HPC-like workloads, but is also functionally capable to run big-data software, out-of-the-box. What remains unknown is if relevant big-data workloads can run with *good performance* on a KNL-based infrastructure. As discussed in Section VI, previous performance studies of first-generation Intel MICs explore much of the performance-space, but KNL has a significantly different architecture and its performance-space remains largely unexplored beyond HPC.

To answer the question *Can big data and HPC infrastructure converge?*, we propose in this work to take a first step toward understanding the performance of KNL-based

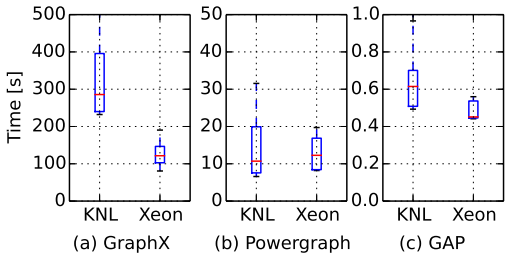


Fig. 1. Diverse performance results in the hardware-software parameter space of (a) GraphX, (b) Powergraph, and (c) GAP, running on KNL or Xeon. (Dataset: `DFB79`. Workload: `PageRank`. Vertical axes differ.)

infrastructure, focusing on *processing power*, for an important class of applications that lie at the intersection of big-data and HPC: *graph analytics*. Although both the HPC and the big data fields are vast, with many more workload categories, our study takes a needed step in an otherwise under-explored direction that could apply to many existing distributed ecosystems [10]. Therefore, for the scope of this article we restrict our study to graph processing. Graphs already represent data in a variety of application domains (see Section III-B), and are currently processed with various generic and x86-compatible big-data solutions (e.g., GraphX [11], Powergraph [12], Apache Giraph [13]), and with performance-centric HPC solutions (e.g., CuSHA [14] and Gunrock [15] on GPUs, Polymer [16] on multi-core NUMA architectures).

In this work, we explore the performance-interaction between KNL and graph-processing platforms (wide range of interactions, exemplified by Figure 1). This is a highly complex endeavor: the community has firmly established that performance depends non-trivially on the interaction of software-platform, algorithm, and dataset ("*the PAD triangle*") [17], [18], [19], [20]. Additionally, we identify that running graph-processing workloads on KNL-based infrastructure adds a hardware dimension of complexity, to form the *hardware-platform-algorithm-dataset* (HPAD) performance-space.

To this end, we contrast the performance achieved by KNL and by typical big data processors (e.g., the Intel Xeon-family). The impact of the H-dimension (the hardware parameter-space) is suggested by the much wider range of performance results for KNL-based than for Xeon-based deployments, which Figure 1 summarizes and Section V-A explains. Overall, the figure shows evidence that ill-chosen configurations for the *hardware* and software parameters can generate a wide range of workload response times; this indicates difficulties for performance portability.

Toward qualitatively and quantitatively assessing the practicality of HPC and big data convergence for graph processing, we make in this work a five-fold contribution:

1) We propose a method for analyzing the performance of modern graph-processing platforms on KNL-based infrastructure (Section III). Our experimental method uses sensitivity analysis to explore the impact of the entire HPAD performance-space. To account for advances in graph-processing, the method explores three classes of platforms.
2) We design, around the method at point 1, a practical, yet comprehensive process to conduct a comparative performance study of graph processing using KNL- and Xeon-based infrastructure (Section IV). We use state-of-the-art workloads, datasets, and software for graph analytics.
3) We use the practical process at point 2 to quantify and explore the large hardware parameter-space of the Intel KNL, and to showcase its interaction with the PAD triangle (Sections V-A for un-tuned and V-F for tuned platforms). Our analysis navigates the complex parameter space of hardware-software interaction, using about 300,000 compute-core hours (significantly less than the exponential design).
4) We present guidelines for tuning representatives of the three classes of graph-processing platforms considered in this work (Sections V-B, V-C, and V-D). We identify, for each representative, critical parameters and their values for achieving good performance on the KNL, when running state-of-the-art graph-analytics platforms.
5) We study and compare the strong scaling of the three classes of graph-processing platforms (Section V-E). We find strong scaling where it exists, and identify an important challenge for performance engineering.

II. Hardware for Graph Processing

In this section, we present a comparative analysis of hardware typically used for graph-processing platforms across big data and HPC. We further present Intel KNL, the HPC hardware that could bridge the big data-HPC gap.

A. Big Data Hardware for Graph Processing

We investigate what is the typical software-hardware for graph processing, in big data ecosystems. To find this, we conduct a systematic survey of distributed graph-processing platforms. We choose the most highly-cited, community- and industry-driven, platforms of the previous 8 years, published in top-tier venues (data source: Google Scholar).

Table I summarizes our findings, in chronological order. The entries marked "—" are either not specified by the authors, or cannot be inferred due to virtualization. We find that the most cited community-driven graph processing systems are Powergraph, GraphX, Giraph (highly tailored towards addressing Facebook's needs [21]), and GraphMat. Further, we identify two industry-driven platforms: PGX.D (Oracle) and SystemG (IBM). With the exception of SystemG, all selected platforms were published in top-tier venues, such as *VLDB*, *SC*, and *OSDI*. We find that the typical big-data hardware is represented by general purpose, x86-based commodity CPUs, with modest numbers of fat cores per socket, large L3 caches, and large amounts of memory per core. Memory is the resource with the fastest increasing trend.

B. HPC Hardware for Graph Processing

To determine HPC software-hardware commonly used for graph processing, we conduct a similar survey: we search for the most highly-cited publications of graph-processing platforms in the HPC domain.

Our study reveals that the highest-cited HPC approaches for graph processing are: Medusa [25], Gunrock [15], CuSHA [14], Totem [26], Polymer [16], and Mosaic [27]. We find that the most commonly used hardware for HPC graph analytics is massively parallel, using high-speed interconnects, and/or with intricate NUMA hierarchies. Such hardware is primarily designed to achieve high peak performance (hundreds of GFLOPS or TFLOPS) for compute-intensive applications. However, due to the constant increase in memory bandwidth, these platforms also provide good performance for highly-dynamic, memory-intensive big-data applications.

TABLE I
GRAPH-PROCESSING PLATFORMS AND HARDWARE USED IN PEER-REVIEWED WORK. (THE SYMBOL '—' DENOTES UNAVAILABLE INFORMATION.)

System	Citations (March '18)	Year	Developer	Sockets	Cores/ socket	Cache	Memory	Architecture
Powergraph [12]	869	2012	Community	2	4	12 MB (L3)	30 GB	Intel Xeon
GraphX [11]	482	2014	Community	—	8	—	68 GB	Virtualized, Amazon EC2
Giraph [21]	121	2015	led by Facebook	—	16	—	—	—
GraphMat [22]	69	2015	led by Intel	2	12	30 MB (L3)	60 GB	Intel Xeon
PGX.D [23]	35	2015	Oracle	2	8	20 MB (L3)	256 GB	Intel Xeon
SystemG [24]	0	2018	IBM	4	24	8/16 MB (L3/L4)	2 TB	IBM S824L, PowerPC

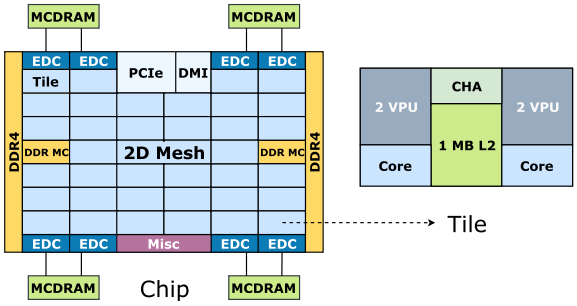


Fig. 2. KNL architecture. (Adapted from [8, p. 143].)

C. Convergence and the Intel KNL

We identify two options to achieve HPC and big data convergence for *both* hardware and software: (1) porting HPC software to big data hardware and (2) porting big data software to HPC hardware, yet none of these approaches is feasible without expert knowledge and engineering effort, mainly due to divergent infrastructure. Until recently, HPC infrastructure was not prepared to execute general-purpose big data software out-of-the-box; when porting is attempted, significant research and engineering efforts are required, not only for code re-writing, but also for parameter tuning (e.g., porting Spark on GPUs [28]). Similarly, highly-specialized, highly-tuned HPC software, typically written for different kinds of accelerators, is incompatible with regular big-data hardware without significant effort invested in porting and tuning.

Due to emerging hardware, such as Intel's Knights Landing (KNL) architecture, it is now possible to directly run big data software on HPC hardware. KNL is a hybrid many-core, with up to 72 thin cores on a single socket, 4 hyperthreads per core, a large, fast on-chip memory (16GB) and wide (512B) vector registers. Despite its HPC features, KNL can execute general-purpose x86 code, and is capable of self-booting and running an operating system. As such, big-data graph analytics platforms are directly portable to KNL, cutting down the software re-writing efforts, but still leaving large hardware and software parameter space to be explored. This portability makes Intel KNL a prime candidate for exploring big data and HPC convergence. Therefore, in this work, we conduct the first comprehensive performance study on the implications of running representative graph-processing workloads on big-data software-platforms, on Intel KNL.

KNL Architecture. Figure 2 presents the Intel KNL architecture. The processor is composed of up to 36 2-core tiles (38 physically printed on die, 2 disabled), interconnected by a *2D mesh*. Each core is able to run 4 hardware threads and contains 2 vector processing units (VPUs), able to perform 512-bit SIMD operations. Pairs of cores on a tile share a 16-way associative 1MB L2 cache and a caching home-agent (CHA), which is a distributed tag directory for cache coherence.

High-bandwidth MCDRAM. KNL is equipped with a 16GB high-bandwidth multi-channel dynamic random access memory (MCDRAM), connected to the chip via the EDC controller. The MCDRAM is capable of achieving bandwidths of over 400GB/s. However, as reported by Ramos and Hoefler [29], its latency is higher than system DRAM, due to the multiple hops generated by the distributed tag directory.

Clustering and Memory Modes. At boot time, the KNL user can select from a number of *clustering* and *memory modes* [8]. The former refer to how memory lines are mapped in the distributed tag directory of the L2 cache, while the latter refer to how the on-chip MCDRAM is exposed to the user. The clustering modes operate as follows:

1) `All-to-all (A)`: addresses are hashed uniformly over all CHAs. This scheme delivers uniform performance for all memory accesses, irrespective of thread placement.
2) `Quadrant (Q)`: the 2D mesh is split into four regions which are spatially local to groups of memory controllers. Memory accesses are guaranteed to be served by a tag directory local to the quadrant, thus reducing latency.
3) `Hemisphere (H)`: similar to Quadrant, but the mesh is split spatially into two regions.
4) `SNC-4 (S4)`: mode is similar in behavior to `Quadrant`, but it also exposes the 4 partitions as NUMA nodes to the operating system, allowing NUMA-aware optimizations.
5) `SNC-2 (S2)`: similar to SNC-4, but with 2 regions.

The memory modes operate as follows:

1) `Flat (F)`: the MCDRAM is exposed as a separate NUMA node. In this setup, the user decides (through the use of software libraries or `numactl`) where the program memory is allocated: in the system memory or in the MCDRAM.
2) `Cache (C)`: the MCDRAM acts as a last-level cache between the KNL and the system DRAM. In this setup, the entire address space of the DRAM is cached transparently into the MCDRAM, without the intervention of the programmer.
3) `Hybrid (H)`: the MCDRAM is partitioned into a *cache* region and *flat* region, exhibiting the properties of the `C` and of the `F` modes, respectively.

III. Method for Assessing Performance in Modern Graph Processing

In this section, we present our method for analyzing the performance of graph-processing platforms on Intel KNL. Our method is experimental in nature and aims to quantify the entire *hardware-platform-algorithm-dataset* (HPAD) performance-space (Section III-A). To this end, our method is based on *sensitivity analysis*: comprehensive parameter tuning and experimental exploration, and state-of-the-art datasets (Section III-B) and workloads (Section III-C). We also propose a conceptual framework to describe modern graph-processing platforms, with three classes (Section III-D). Overall, our method focuses on *quantifying the impact of KNL's hardware complexity on the performance of different classes of graph-processing platforms*.

A. Exploring the *HPAD* Performance-Space

Several studies [18], [19], [20] show that characterizing the performance of graph-analytics performance is complex: performance depends, non-trivially, on the platform-algorithm-dataset interaction (the PAD triangle). Running graph processing workloads on KNL adds the hardware (H) dimension to the complexity of the problem. Thus, in contrast to previous studies, in this work we focus on empirically quantifying the complexity of the entire HPAD space.

Our main method adds to the state-of-the-art PAD analysis a *sensitivity-analysis of the hardware-related parameters*, extracted from KNL's hardware features (Section II-C). By design, this sensitivity analysis will help uncover other, non-trivial KNL parameters that impact the performance of graph-analytics platforms, further allowing us to characterize the HPAD interaction. We highlight our main findings regarding these parameters, which implicitly indicate the interactions of the H and PAD dimensions, in our evaluation (Section V).

B. PA*D*: Graph-Processing Datasets

Graphs are powerful abstractions that enable rich data analysis. Such analyses range from traditional graph algorithms (e.g., graph search, shortest paths, transitive closures), to modern machine learning [30] and to deep learning on graph data [31]. Such large variety of datasets and workloads leverage a high societal impact. Graph processing has been used for combating human trafficking [32], monitor wildfires [33], study the human brain [34] and drug discovery [35].

A recent study [36] from Broido and Clauset presents evidence that *scale-free networks are rare*. This finding is in contrast with typical graph datasets used in practice to assess the performance of graph analytics systems, which are generally scale-free, powerlaw datasets, such as the Graph500 [37], or R-MAT [38]. To align with the finding of Broido and Clauset, we curate our input datasets and consider both scale-free and non-scale-free graphs. For the former, we use Graph500 datasets, while for the latter we use Datagen [39], [40] graphs, which closely follow real-world social networks structure, through modeling the communities that emerge in social networks.

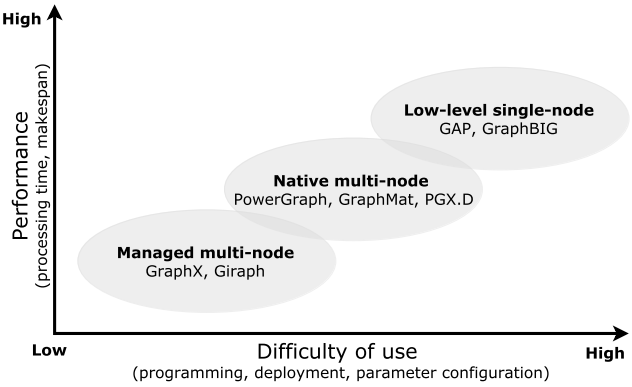


Fig. 3. Difficulty of use vs. Performance for graph processing platforms: the closer to the hardware, the more difficult to use, and especially to port.

C. PAD: Graph-Processing Workloads

Although graph analytics workloads are extremely diverse, they are highly challenging from a computational perspective. First, a vast majority of these algorithms perform little computation, rendering them heavily memory-bound. Second, they require traversing links between graph entities, which leads to irregular memory access patterns. This irregularity is challenging in massively parallel or distributed setups, as it leads to low bandwidth utilization [41], large amounts of cache misses [18], and network communication [42]. The exact impact of such behavior on the observed performance depends heavily on the algorithm and its internal data structures and representation (e.g., edge lists, adjacency lists, sparse matrices and their storage formats), on the programming model (e.g., Pregel, GAS [12], sparse matrix operations [43]), on the dataset [44], and even on the partitioning scheme [45].

In this work, we build upon comprehensive performance analysis work such as GAP [46], [41], and LDBC Graphalytics [19], which uncover through diverse tests the intricacies of understanding the performance of graph-analytics platforms. They both consider a varied portfolio of workloads, and their common subset is used in this study.

D. *P*AD: Graph-Processing Platforms

To address the growing diversity of graph datasets and graph analysis problems, developers and system integrators have created a large variety of graph-processing platforms, tailored to specific needs and use-cases. We have analyzed representative graph-processing platforms, popular in both industry and academia, to understand their main features, performance behavior, and difficulty of use. This analysis lead us to identify three main classes of graph-processing platforms: (1) managed multi-node, (2) native multi-node, and (3) low-level single-node implementations; their behavior is presented in Figure 3 and detailed in the following paragraphs.

(1) Managed multi-node: These platforms run in a multi-node managed environment, such as the *Java virtual machine*, or *Microsoft .Net*, and are built to be easily portable, running on most types of big data infrastructure. Common examples include GraphX [11] and Giraph [13]. At such a high level of abstraction, platforms cannot exploit underlying hardware

TABLE II
KNL VS. XEON MACHINES USED IN OUR EXPERIMENT DESIGN.

	Xeon E5-2630v3	Xeon Phi 7230
Cores	16 (32 hyperthreads)	64 (256 hyperthreads)
Frequency (GHz)	2.4	1.3
Network	56Gbit FDR InfiniBand	56Gbit FDR InfiniBand
Memory	64GB DDR4	96GB DDR4
OS	Linux 3.10.0	Linux 3.10.0

properties and therefore *cannot be tuned* to match the underlying hardware. Their tuning options focus on the high-level mapping of model's abstractions to a high-level, simplistic model of the hardware platform, and are typically limited to choosing the number of threads per worker, tweaking the data partitioning per worker, or the file system block size.

(2) Native multi-node platforms run in a multi-node native environment. They are typically programmed in C/C++, closer to the hardware and inherently much harder to tune for a given architecture than the *managed platforms*. Common examples include Powergraph [12], GraphMat [22], and PGX.D [23]. Despite being closer to the hardware, such platforms are still general enough to permit portability. Moreover, they promise to achieve superior performance to *managed platforms*. Possible tuning options include choosing the appropriate numbers of threads per worker, and pinning threads to cores.

(3) Low-level single node: These are single-node native platforms which are performance-oriented and include architecture-dependent optimization [47]. As a consequence, such implementations are able to exploit better the hardware capability, leveraging good performance, but the programming and tuning efforts are high and, more importantly, portability is difficult to achieve. Common examples include: GraphBIG [18], CRONO [48], and GAP [46]. The performance of such platforms is expected to be higher than that of the (single-node) *native* platforms. Possible tuning options include choosing the appropriate number of running threads, changing pinning patterns, the choice of appropriate compiler and enabling/disabling vectorization.

Our HPAD analysis covers all three classes of graph processing platforms, by selecting one representative from each class. For each representative platform, we compare KNL's performance against the reference, Xeon-based versions of these platforms. Further, we correlate the platform tuning parameters to KNL's hardware-related parameters, and leverage our sensitivity-analysis to determine the most impactful ones.

IV. EXPERIMENT SETUP

In this section we describe our experiment setup, from design to hardware, software, datasets, workloads, and metrics.

1) Experiment Design: To assess the practicality of big data and HPC convergence, we design a comprehensive experiment to evaluate and tune KNL's performance (our HPC-hardware platform) for representative graph-processing software and workloads (our big-data software). Our empirical study features real-world workloads, as proposed by the LDBC Graphalytics [19] and GAP [46] benchmark suites, a total of 8 datasets (3 scale-free, 4 synthetic graphs that closely model real-world network structures [36], and 1 very large real-world network [49]), and three representative graph-processing platforms (GraphX, Powergraph, and GAP). We choose GraphX and Powergraph due to their popularity and extensive use (see: Table I). However, through the LDBC Graphalytics drivers, one could replace these two platforms with other candidates. Regarding *Low-level platforms*, we selected the reference implementation of the GAP benchmark which, according to our tests, provides more stable behavior and superior performance to alternatives such as GraphBIG [18] and CRONO [48]. Table V summarizes our experiment design. A full performance-space exploration would be too expensive: our selected experiments, in Section V, spanned over 300,000 compute-core hours.

TABLE III
SELECTED SOFTWARE PLATFORMS, ACCORDING TO SECTION III-D TAXONOMY.

Platform	Type	Model	Language	Version
GraphX	Managed multi-node	Spark	Scala	1.6.0
Powergraph	Native multi-node	GAS	C++	2.2
GAP	Low-level single-node	OpenMP	C++	1.0

2) Workloads: The workloads we use are the intersection of LDBC Graphalytics [19] and GAP [46] algorithms:

- Breadth-first search (BFS): for a given source, measures the minimum number of "hops" to reach all vertices.
- Weakly-connected components (WCC): determines the weakly-connected component each vertex belongs to.
- Single-source shortest paths (SSSP): for a given source, computes the path of minimum cost to reach all vertices.
- PageRank [50] (PR): determines the rank of each vertex.

3) Metrics: In our experiments we report only processing time (T_p), as it is generally the only parallel part, i.e., Powergraph and GAP perform single-threaded graph loading and building.

4) Systems Software: Graph-processing software-platforms depend on underlying systems software (e.g., compilers, virtual machines, runtimes, libraries) to run efficiently. For native platforms we used two compilers, `icc` (18.0.2, 15.0.4) and `gcc` (6.4.0, 5.2.0), and Intel MPI 18.0.2 for message passing. We found that performance varies between compilers, but the impact of different compiler versions is minimal. Thus, for GAP and Powergraph, the results presented in Section V are the *best results* achieved with the `icc` compiler (unless otherwise stated). For GraphX, we used Oracle JDK 1.8.0_162.

5) Hardware: We use two hardware plaforms (Table II): KNL is our HPC hardware of choice for assessing the convergence feasibility, while Xeon is representative for typical big data hardware. The KNL clustering and memory modes used in our experiments are depicted in Table V. We do not consider the SNC-2 and Hemisphere modes, because their behavior is similar to SNC-4 and Quadrant, respectively; we also do not use the Hybrid memory mode since it is a mere combination of Flat and Cache. All experiments except *strong horizontal scalability* use single-node deployments.

TABLE IV
REAL-WORLD (FRI) AND SYNTHETIC DATASETS USED IN OUR EXPERIMENT DESIGN.

Dataset	Type	Scale	$\|V\|$	$\|E\|$	Degree distribution
DFB79	Datagen	Small	1.4M	85.7M	Facebook
DZF82	Datagen	Medium	43.7M	106.4M	Zipfian
DFB86	Datagen	Large	5.7M	422.0M	Facebook
DZF87	Datagen	Large	145.1M	340.2M	Zipfian
G24	Graph500	Medium	8.9M	260.4M	Power law
G25	Graph500	Large	17.1M	523.6M	Power law
G26	Graph500	Extra large	32.8M	1.1B	Power law
FRI	Friendster	Extra large	65.6M	1.8B	Real-world

6) Platform Tuning: We iteratively tuned the software platforms' parameters to get the best performance for both KNL and Xeon. We employed a "guided parameter search" with the human expert in the loop, as otherwise the cost and time to run all possible configurations would be too high. For all the experiments presented in this paper, including tuning, we used over 300,000 compute-core hours. More than 80% of this time was spent on the KNL, as the Xeon was much easier to tune.

7) Relevance: We report performance results using the guidelines of Hoefler and Belli [51]. For all experiments, we have performed at least five repetitions. As we found that the results are deterministic, with minimal variability for the same parameters (below 5%), all our plots report averages and do not include error bars.

V. GRAPH ANALYTICS ON KNL AND XEON

In this section we present the results of KNL and Xeon experiments, based on the setup designed in Section IV. Table V summarizes the goal and configuration of each experiment. Our main findings are the following:

MF1 *Convergence:* for graph processing, the KNL-based infrastructure can deliver better performance than Xeon-based infrastructure, per machine. This makes KNL a candidate-part of converged HPC-big data infrastructure.

MF2 *The H in HPAD:* relatively to Xeon, the KNL introduces the hardware-dimension for performance engineering: the KNL clustering and memory modes impact significantly and non-trivially the performance achieved for graph processing. In particular, the platform-hardware interactions are different across the three classes of platforms.

MF3 *The H-P interaction:* the platforms closer to the hardware deliver better performance, in general, but require significantly more and/or more complex tuning.

MF4 *Tuning:* to reach better performance than Xeon-based infrastructure, the KNL-based infrastructure requires detailed control over many system-parameters and extensive effort (and, for native platforms, expertise). The tuning results we present and guidelines we formulate should inform practitioners.

MF5 *Scaling:* We find the following technical constraints and affordances of KNL: (i) KNL exhibits strong vertical scalability; (ii) KNL does not scale well horizontally; (iii) CPU-intensive algorithms, such as PageRank, favor KNL over typical big data hardware when scaling.

A. Quantifying the impact of the hardware parameter space

We start with the results from Figure 1 (Section I). To understand the range of performance results for the three classes of graph-analytics platforms, we measure the performance of each platform, across all hardware and software parameters studied in this work (see Table V and Sections V-B–V-D), when running on each of the hardware platforms. We only use results obtained for single-node deployments.

Figure 1 summarizes the performance results, per software platform and per hardware deployment, as "box-and-whiskers" (that is, IQR box, median bar, min-max whiskers). The results coalesce detailed performance results obtained for all configurable parameters (both software and hardware in the KNL case, and only software in the Xeon case) of each platform, which are first-class candidates for tuning the performance of graph-analytics platforms. The ranges depicted in the figure are larger for KNL-based deployments than for Xeon-based deployments.

The results in Figure 1 indicate that, for KNL-based deployments, an untrained practitioner or a naïve deployment can lead to either good or bad performance, anywhere in the broad range of empirical results; whereas, for Xeon-based deployments, the expected performance is much more stable regardless of expertise. This gives evidence that the *performance range for KNL-based deployments* (HPC-friendly) *is much broader than for Xeon-based deployments* (typical big data hardware). We conclude that the results of this analysis support the main finding **MF2**.

B. Tuning GraphX

We tune in this section GraphX, a platform representative for the managed multi-node class. Overall, we find that performance is sensitive of all parameters we explore, which supports **MF3** and the KNL-related parts of **MF4**.

Workers and Threads. We vary the number of Spark workers (w) per node, and of threads per worker (t), such that $w \times t = 128$. The value 128 is the best possible for a conservative setup: using all 256 hardware threads of the KNL results intermittently in either poor performance or crashes. This is expected, and Spark best-practices [52] suggest not to use all compute resources. Instead, some threads should be left for the OS, the JVM garbage collector, or internal management (e.g., of Spark, Hadoop, HDFS, and other components of the big data ecosystem).

Figure 4a depicts the results. The best-performing setups are, depending on the algorithm, 1×128 (for SSSP and BFS) and 2×64 (for WCC and PR). Because PR is especially compute-intensive, we use the 2×64 configuration for all further GraphX experiments (workload-dependent *guideline*).

Data Partitions. Spark data-structures (i.e., RDDs [53]) are partitioned across workers, and the number of partitions gives the maximum amount of parallelism a job can exhibit. A small

TABLE V
SUMMARY OF EXPERIMENTS CONDUCTED IN SECTION V.

Section	Description	Hardware	Platform	Algorithm	Dataset	Platform params.	No. of Nodes
§V-A	Exploring performance range	All	All	All	All	All	1
§V-B	Tuning GraphX	KNL, mode Q	GraphX	All	DFB79, G24	Various (see §V-B)	1
§V-C	Tuning Powergraph	KNL, mode Q	Powergraph	All	All	Various (see §V-C)	1
§V-D	Tuning GAP	KNL, mode Q	GAP	All	All	Various (see §V-D)	1
§V-E	Exploring strong scaling	All	All	All	DFB79, G25	Best found through tuning	1,2,4,8
§V-F	Exploring KNL modes	KNL, All modes	All	All	DFB79	Best found through tuning	1

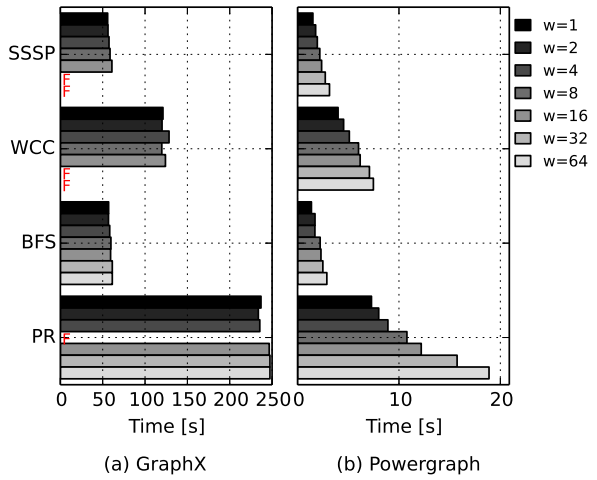


Fig. 4. Processing time (T_p) of KNL vs. algorithm: (*left*) $r = 128$, (*right*) $r = 256$, where $r = w \times t$ is the number of running threads, w is the number of workers, and t is the number of threads/worker. (Dataset: DFB79.)

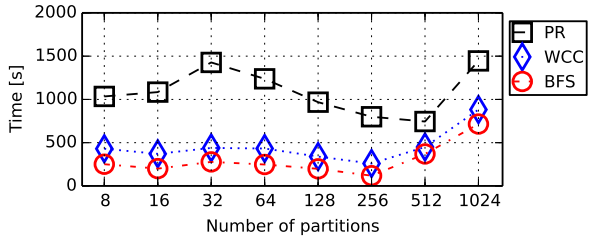


Fig. 5. The impact of the number of partitions on the GraphX runtime. (Dataset: G24.)

number of partitions can lead to underutilization, whereas a very large number of partitions can lead to overhead. Figure 5 shows the impact of the number of partitions on performance, for various graph-processing workloads. The results indicate that BFS and WCC favor 256 partitions (2 per KNL core), but PR favors 512 partitions (4 per KNL core).

We derive from these results a *guideline*: for graph processing, set the number of partitions higher than typically for big data, with a ratio closer to 4/core than to 1/core.

Block Size. In a big data ecosystem, Spark typically inputs data from a distributed datasource, such as HDFS, where the input is split into *blocks*. HDFS best-practices [54, p. 315] suggest block-sizes of 64 or 128 MB. To understand the impact of block-size on performance, we stored the input-graph in HDFS, and varied the block size of the input-graph files (i.e., containing vertices and edges), from 8 MB to 256 MB.

Figure 6 compares the performance achieved by GraphX, when running on KNL and on Xeon, with varying block-

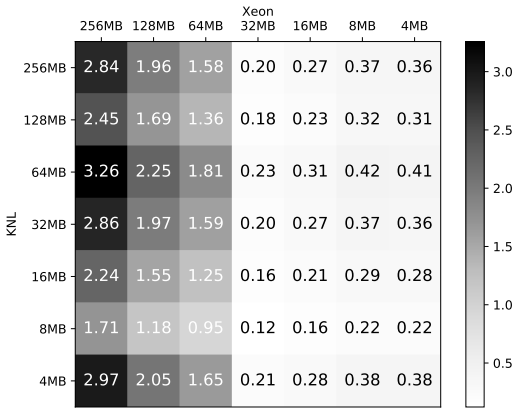


Fig. 6. GraphX block-size analysis, KNL vs. Xeon. Values in matrix represent the ratio of runtimes, KNL:Xeon. Algorithm: WCC. Dataset: G24. (Higher values are worse. For values above 1, KNL is slower than Xeon.)

sizes. The cells in the comparative matrix represent ratios between the performance of KNL and of Xeon, for the block-size configuration corresponding to the row and the column of the cell, respectively. From the set of values in any row, which take as numerator the same KNL performance-result and as denominator the performance obtained for each configuration on Xeon, we conclude that the best Xeon configuration is 256 MB, which falls outside recommended best-practices. Similarly, from the values in any column, we conclude that KNL performs best on non-intuitive block sizes, i.e., lower than 32 MB. This can be attributed to the fine-grained parallelism of the KNL: in contrast to Xeon, KNL is composed of many thin cores. This leads to the general *guideline* of reconsidering best-practices for setting block-sizes, for graph processing.

The results show that, for block-sizes of at least 64 MB, Xeon performance is superior to KNL. Similarly, for block-sizes of at most 32 MB, KNL performance is superior to Xeon. As a *guideline*, the performance of KNL with 8 MB blocks and of Xeon with 64–128 MB blocks are comparable.

C. Tuning Powergraph

We tune in this section Powergraph, a platform representative for the native multi-node class. Overall, as in Section V-B, we find performance sensitivity, and thus support for **MF3** and **MF4**. Additionally, we find in this section that uncommon expertise in performance engineering is required to achieve good performance.

Workers and Threads. We vary the amount of (MPI) workers, and the number of threads per worker, up to a

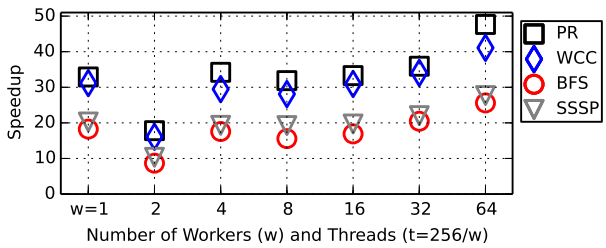


Fig. 7. The speedup achieved by Powergraph, when pinning threads to cores, relative to not pinning (best configuration, see Figure 6). (Dataset: DFB79.)

total of $r = w \times t = 256$ running threads. Figure 4b plots the results. As *guideline*, Powergraph on the KNL strongly favors $w = 1$ MPI-process that spawns $t = 256$ worker-threads. When all threads run within a single MPI process, they communicate through shared memory. Otherwise, with multiple MPI processes, communication is achieved via the MPI communication library, which adds overhead.

Thread pinning. Powergraph implements a mechanism of pinning *pthreads* to *hardware threads*. However, this option is not enabled by default or enabled for the user. We needed to do source-code modifications and a full recompilation to enable this option. Because native-platforms contain significant amounts of intricate performance-related code, this operation requires both general expertise in performance engineering and in-depth knowledge of the platform (e.g., we have used Powergraph for several years).

Figure 7 presents the speedup achieved when pinning threads, relative to no-pinning (the denominator of the speedup-ratio), for different configurations of workers and threads ($r = w \times t = 256$ running threads). The results exceed expert intuition about the KNL: speedups vary between 10 and approximately 50X. In contrast, on Xeon, pinning attains a speedup of at most 10%. These results also show that pinning attains larger speedups for graph workloads that perform more computation, WCC and PR, but also that pinning brings important performance gains for BFS and SSSP. This leads us to a *guideline*: use thread-pinning for Powergraph on KNL.

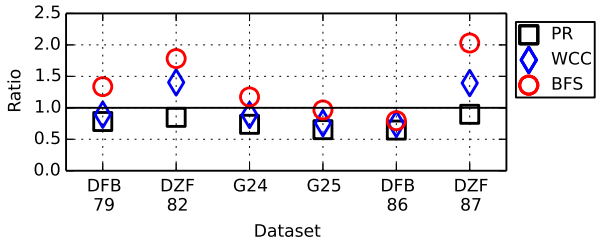


Fig. 8. Ratio of KNL:Xeon runtimes for Powergraph, when processing different graph datasets. Graph sizes increase towards right. Lower ratios are better for KNL-based deployments. Ratio values below 1 mean KNL-based deployments are faster than Intel-based deployments.

Powergraph KNL vs. Xeon. We compare the best-performing configurations of Powergraph on KNL and on Xeon, when processing increasingly larger graphs. Figure 8 plots the results. For the Datagen graphs (names start with "D"), KNL performs better on those with a *facebook* degree-distribution (names include "FB"), whereas Xeon performs better on graphs with *zipfian* degre-distribution (names include "ZF"). This can be attributed to the *facebook* distribution graphs being much denser than the *zipfian* graphs, and hence requiring more computation per vertex.

We conclude that, generally, KNL obtains better performance for larger graphs and for workloads that perform more work. For PR, Figure 8 gives evidence that KNL-based deployments can result in better performance than Xeon-based deployments, an argument for convergence (**MF1**).

D. Tuning GAP

We tune in this section GAP, a platform representative for the native low-level class. Overall, as in Section V-B, we find support for **MF3** and **MF4**. Additionally, we find in this section that the licensed compiler of Intel can provide a significant advantage over the free gcc.

TABLE VI
TUNING GAP: IMPROVEMENT DUE TO DIFFERENT CONFIGURATIONS FOR COMPILER, THREAD-PINNING, AND VECTORIZATION.

Dataset	icc vs. gcc	spread vs. close	AVX512 vs. novec
DFB86	0.86	0.92	0.90
DZF87	0.59	**1.03**	**1.10**
G25	0.79	0.97	0.90
G26	0.83	0.99	0.94

Ratios above 1 (in bold) favor the denominator.

Compiler, pinning, vectorization. Many performance engineering techniques can be used on low-level platforms. We run GAP with code generated in turn by one of several *compilers* (e.g., Intel icc and GNU gcc), with various OpenMP *pinning strategies* (e.g., spread and close are the only strategies fine-grained enough, working per core), and with AVX512 *vectorization* enabled or disabled. We used multiple datasets and compared the performance at full parallelism (i.e., $r = 256$ running threads). Table VI summarizes the results. The icc compiler achieves significantly better performance, on all datasets. Compared with Powergraph, pinning strategies offer little variance in performance, likely due to using OpenMP. Moreover, enabling AVX512 on the Intel icc compiler shows a maximum improvement of only 10% over no vectorization. This can be attributed to GAP not being implemented to fully use the KNL wide-vector registers. For best performance, our *guideline* is to perform comprehensive (auto)tuning for all these parameters.

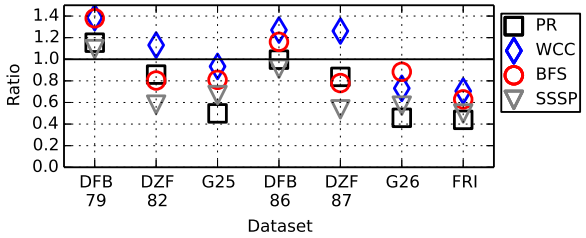


Fig. 9. Ratio of KNL:Xeon runtimes for GAP, when processing different graph datasets. Graph sizes increase towards right. Lower ratios are better for KNL-based deployments. Ratio values below 1 mean KNL-based deployments are faster than Intel-based deployments.

GAP KNL vs. Xeon. Similarly to the last experiment in Section V-C, we compare here the best-performing configu-

rations of GAP on KNL and on Xeon, on increasingly larger graphs. Figure 9 plots the results.

For DZF82 and DZF87, KNL attains speedups of up to 2X (ratio down to 0.5). For the larger graphs, G26 and FRI, KNL obtains even better performance, relatively to Xeon, for all workloads. This is consistent to the many extra cores KNL offers relatively to Xeon, and supports **MF1**.

From the Datagen graphs (names start with "D"), KNL performs better on those with a *zipfian* degree distribution, while Xeon performs better on the graphs with *facebook* degree distribution. This is in contrast with the Powergraph results, which leads to our conclusion that the HPAD relationship is non-trivial and significant (**MF2**).

E. Strong Scaling

Strong scaling keeps the workload fixed while adding resources. Here, we consider, in turn, strong vertical scaling (adding cores inside the same machine) and strong horizontal scaling (adding cores across multiple machines). Overall, we find evidence of good scalability, but also some limitations to it, across the platforms–support for **MF5** and **MF2**.

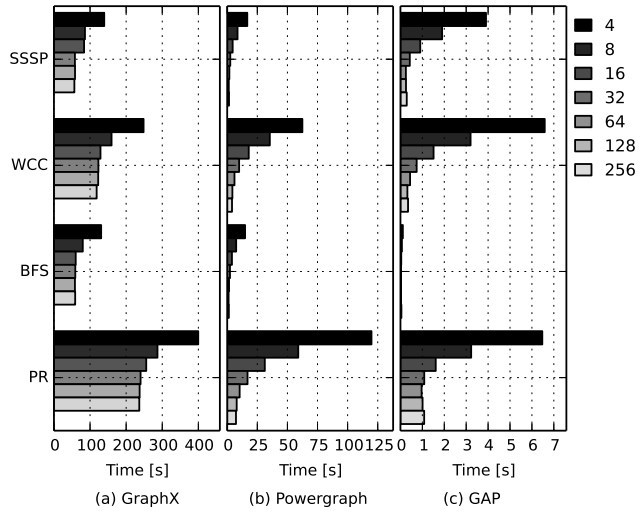
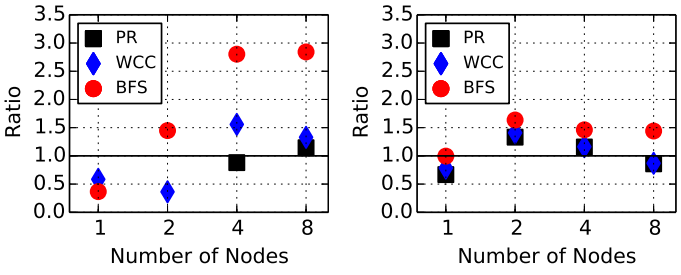


Fig. 10. Strong vertical scalability on KNL, exponentially increasing the number of threads. Dataset: DFB79. (Horizontal axes differ across subplots.)

Strong vertical scaling. We experiment with all three platforms, using the DFB79 dataset. We increase the number of running threads, in powers-of-two starting from 4, up to 128 for GraphX, and up to 256 for Powergraph and GAP. Figure 10 depicts the results. We find that, whereas Powergraph and GAP show excellent scaling behavior, GraphX attains limited vertical scalability.

Strong horizontal scaling. We assess the strong horizontal scaling behavior of GraphX and Powergraph, on KNL and Xeon, for an experiment doubling the number of machines gradually, from 1 up to 8. (GAP operates only single-node. Experimenting at larger scales was not possible with the environment to which we had access.)

Figure 11 plots the results. (Some workloads do not complete their operation in one hour and, according to benchmark specifications, are considered failed.) On GraphX, the KNL is competitive on 1 machine for all workloads that complete,



(a) GraphX. (b) Powergraph.

Fig. 11. Horizontal scalability analysis on KNL vs. Xeon running algorithms WCC, BFS, PR. Points depict the ratio of KNL:Xeon runtime. Lower is better for KNL. Dataset: G25. There are missing symbols: PR does not complete within 2hrs, on 1 and 2 nodes, for KNL and Xeon.

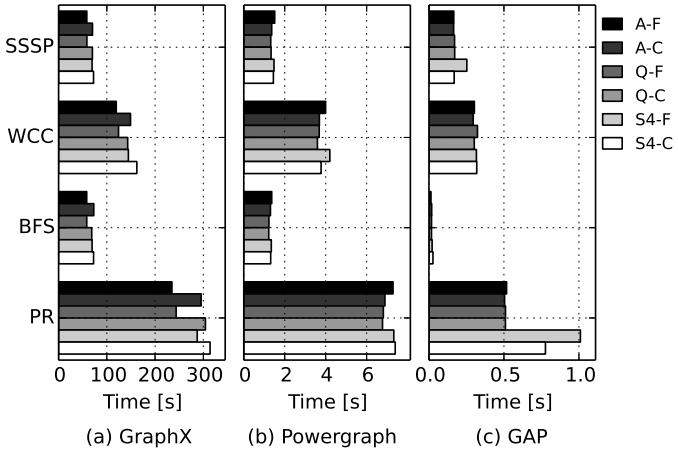


Fig. 12. KNL clustering and memory modes analysis when running workloads on GraphX, Powergraph, and GAP. Dataset: DFB79. (Horizontal axes differ.)

on 2 machines only for WCC, on 4 machines only for PR. In contrast, for Powergraph, KNL is competitive on 1 machine for all workloads, and on 8 machines only for WCC and PR.

Overall, the results indicate the *guideline*: expect poor horizontal scaling. We attribute the poor performance to poor use of the network. GraphX and Powergraph are not optimized to exploit the KNL parallelism for network communication, as they assume typical, fat-core, big data hardware. However, the thin cores of KNL do not perform well on single-threaded network communication. In our setup, using the InfiniBand network, single-threaded point-to-point communication between KNL nodes suffers from an up to 3X reduction in bandwidth and an up to 8X increase in latency, compared to Xeon. We conclude that expert-level performance engineering is required to compensate for this situation (**MF2**).

F. KNL Modes Analysis

Using all the expert performance-engineering (best practices and parameter tuning discussed in the previous sections), we perform a parameter sweep over six KNL modes for GraphX, Powergraph, and GAP. Figure 12 summarizes our findings. The platforms closer to the hardware, Powergraph and especially GAP, achieve the best performance. GraphX favors the flat memory-mode over all clustering modes. This behavior suggests that GraphX accesses memory with very irregular patterns and triggers large amounts of high-latency

accesses to the on-chip memory. In contrast, Powergraph achieves better performance using the `cache` memory mode, over all clustering modes. For GAP, the behavior is similar to Powergraph. Over all platforms, the `Quadrant` clustering-mode achieves best performance. The `SNC-4` clustering mode is not competitive, because none of the platforms are optimized to take into account the NUMA topology. We conclude that platform-performance relates differently and non-trivially to the KNL modes (**MF2**).

VI. Related Work

We survey in this section two bodies of related work: studies on enabling big-data software-platforms to run on HPC hardware, and performance engineering studies for KNL. Overall, ours is the first performance study of graph-processing platforms running on KNL-based infrastructure.

1) Big data software on HPC hardware: Few studies on the efficiency of running big-data software on modern hardware have emerged so far. For example, [55] focuses on the JVM, and by extension on JVM-based platforms, showing they are not efficient in using accelerators; [56], [6] show how Spark should be carefully tuned and augmented when running on HPC hardware. Moreover, [57], [58], [59] demonstrate how to augment existing big-data platforms with additional communication primitives to efficiently cope with HPC-like infrastructure; [5] even presents a specific plugin to enhance the performance of MapReduce on KNL-based clusters. All these studies identify shortcomings of big-data software running on HPC infrastructure, but are limited in scope by the targeted software and/or workload. Our study is significantly broader, covering the whole domain of graph processing through representative datasets and software platforms.

2) Performance engineering for KNL: Due to its promising design and software flexibility, KNL's performance has been investigated in several studies. For example, [60], [61] analyze the performance- and power-consumption for scientific kernels, [62], [63], [29] focus on the performance impact of MCDRAM and the memory modes, and [64] investigates networking performance. These studies focus on HPC mini-apps and benchmarks with different characteristics from graph processing. Our performance study fills this gap.

Multiple studies showcase KNL-specific performance-engineering techniques, successfully applied to scientific applications, from molecular dynamics simulations [65] to seismic simulations [66]. Applying similar techniques to graph-processing kernels leads to significantly lower performance gain [67], due to inherent features of graph processing workloads (see Section III). Our optimization strategies are less intrusive, as we approach the performance analysis and tuning problem from the perspective of a regular, non-expert user.

A single performance study presents a *somewhat* detailed characterization of KNL's performance for graph processing workloads [68]. Our study is significantly more extensive in scope, and much more comprehensive in depth: we use orders of magnitude larger graphs, we analyze representatives from all three classes of graph analytics platforms (instead of one), and we add the complex, multi-node scalability analysis.

VII. Conclusion

"Can big data and HPC infrastructure converge?" is a pressing and challenging problem of the HPC and big data communities. The current approaches seem build-first, and raise the daunting challenge of solving portability for a broad domain. In contrast, in this work we have advocated an understand-first approach, and conducted the first performance study of graph processing (a representative big-data workload) on big-data software platforms running on HPC infrastructure based on KNL (which can run graph processing workloads out-of-the-box). In other words, we investigate KNL as a hardware platform to achieve HPC-big data convergence.

We have proposed a method for conducting performance studies of graph processing running on KNL-based infrastructure. The method focuses on the complex HPAD performance space, and on three classes of modern software platforms for graph processing. Around this method, we have designed a practical experiment-design, balancing the needs of exploration with the pragmatic cost (hundreds of thousands of core-hours). We have put the method and the experiment design in practice, by conducting a comprehensive performance-study of GraphX (as a representative of managed multi-node platforms), Powergraph (native multi-node), and GAP (low-level single-node) running on KNL-based infrastructure.

Our results show that:

MF1 *Convergence:* performance-wise, KNL is promising for the HPC-big data convergence: compared to typical big-data hardware, KNL achieves good performance, especially on larger input sizes.

MF2 *The H in HPAD:* we found that the HPAD concept captures very well the performance-space of graph processing on KNL.

MF3 *The H-P interaction:* we found that the closer the platform is to the hardware, and the less general is its programming model, the better its performance is. We showed strong evidence that none of the tested platforms can fully utilize the KNL hardware, requiring performance engineering beyond mere tuning.

MF4 *Tuning:* we showed that each type of platform interacts differently with the KNL hardware, and requires different types of tuning.

MF5 *Scaling:* we found KNL shows poor horizontal strong-scaling, in particular for the less CPU-intensive algorithms.

We envision that, with some adaptations (e.g., parallelizing the communication engine), graph processing could fully utilize HPC hardware similar to KNL. A graph-processing platform, if KNL-enabled, will have to better utilize wide-vectors, make better use of the high-bandwidth on-chip memory, and improve cross-machine communication. We plan to further assess how other classes of big-data applications perform on KNL processors with HPC-grade I/O-hardware [6], [69].

ACKNOWLEDGMENTS

Work supported by Dutch projects Vidi MagnaData, COMMIT/. The infrastructure was kindly provided by Intel as a gift, by the Dutch Supercomputing Center through the HPGraph NWO grant, and by the Dutch DAS5 Supercomputing infrastructure co-sponsored by NWO.

REFERENCES

[1] G. Antoniu *et al.*, "Big Data and Extreme-scale Computing: Pathways to Convergence," http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/whitepapers/bdec2017pathways.pdf, 2018, [Online; accessed 17-July-2018].

[2] G. Fox, J. Qiu, S. Jha, S. Ekanayake, and S. Kamburugamuve, "Big data, simulations and hpc convergence," in *WBDB*. Springer, 2015, pp. 3–17.

[3] G. C. Fox and S. Jha, "A tale of two convergences: Applications and computing platforms."

[4] S. Jha, J. Qiu, A. Luckow, P. Mantha, and G. C. Fox, "A tale of two data-intensive paradigms: Applications, abstractions, and architectures," in *Big Data (BigData Congress), 2014 IEEE International Congress on*. IEEE, 2014, pp. 645–652.

[5] L. Chen, B. Peng, B. Zhang, T. Liu, Y. Zou, L. Jiang, R. Henschel, C. A. Stewart, Z. Zhang, E. McCallum, Z. Tom, J. Omer, and J. Qiu, "Benchmarking harp-daal: High performance hadoop on KNL clusters," in *CLOUD*, 2017, pp. 82–89.

[6] M. Anderson, S. Smith, N. Sundaram, M. Capotă, Z. Zhao, S. Dulloor, N. Satish, and T. L. Willke, "Bridging the gap between HPC and big data frameworks," *Proceedings of the VLDB Endowment*, vol. 10, no. 8, pp. 901–912, 2017.

[7] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs scale-out for hadoop: Time to rethink?" in *Proceedings of the 4th annual SoCC*. ACM, 2013, p. 20.

[8] J. Jeffers, J. Reinders, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann, 2016.

[9] "Top 500," "http://www.top500.org", [Online, accessed: May, 2018].

[10] A. Iosup, A. Uta, L. Versluis, G. Andreadis, E. van Eyk, T. Hegeman, S. Talluri, V. van Beek, and L. Toader, "Massivizing computer systems: a vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems," in *IEEE ICDCS*, 2018.

[11] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph processing in a distributed dataflow framework." in *OSDI*, vol. 14, 2014, pp. 599–613.

[12] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: distributed graph-parallel computation on natural graphs." in *OSDI*, vol. 12, no. 1, 2012, p. 2.

[13] C. Martella, R. Shaposhnik, D. Logothetis, and S. Harenberg, *Practical graph analytics with Apache Giraph*. Springer, 2015.

[14] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, "CuSha: vertex-centric graph processing on GPUs," in *HPDC*, 2014, pp. 239–252.

[15] Y. Wang, A. A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: a high-performance graph processing library on the GPU," in *PPoPP*, 2016, pp. 11:1–11:12.

[16] K. Zhang, R. Chen, and H. Chen, "NUMA-aware graph-structured analytics," in *PPoPP*, 2015, pp. 183–193.

[17] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke, "How well do graph-processing platforms perform? An empirical performance evaluation and analysis," in *IPDPS*, 2014, pp. 395–404.

[18] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "GraphBIG: understanding graph computing in the context of industrial solutions," in *SC*, 2015, pp. 1–12.

[19] A. Iosup, T. Hegeman, W. L. Ngai, S. Heldens, A. Prat-Pérez, T. Manhardto, H. Chafio, M. Capotă, N. Sundaram, M. Anderson *et al.*, "LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1317–1328, 2016.

[20] M. Besta, M. Podstawski, L. Groner, E. Solomonik, and T. Hoefler, "To push or to pull: On reducing communication and synchronization in graph computations," in *HPDC*, 2017, pp. 93–104.

[21] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: Graph processing at facebook-scale," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1804–1815, 2015.

[22] N. Sundaram, N. Satish, M. M. A. Patwary, S. R. Dulloor, M. J. Anderson, S. G. Vadlamudi, D. Das, and P. Dubey, "Graphmat: High performance graph analytics made productive," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1214–1225, 2015.

[23] S. Hong, S. Depner, T. Manhardt, J. Van Der Lugt, M. Verstraaten, and H. Chafi, "PGX.D: a fast distributed graph processing engine," in *Supercomputing, SC*, 2015, p. 58.

[24] G. Tanase, T. Suzumura, J. Lee, J. Crawford, H. Kanezashi, S. Zhang, W. D. Vijitbenjaronk *et al.*, "System G distributed graph database," *arXiv preprint arXiv:1802.03057*, 2018.

[25] J. Zhong and B. He, "Medusa: Simplified graph processing on gpus," *TPDS*, vol. 25, no. 6, pp. 1543–1552, 2014.

[26] A. Gharaibeh, L. B. Costa, E. Santos-Neto, and M. Ripeanu, "A yoke of oxen and a thousand chickens for heavy lifting graph processing," in *PACT*, 2012, pp. 345–354.

[27] S. Maass, C. Min, S. Kashyap, W. Kang, M. Kumar, and T. Kim, "Mosaic: Processing a trillion-edge graph on a single machine," in *EuroSys*, 2017, pp. 527–543.

[28] Y. Yuan, M. F. Salmi, Y. Huai, K. Wang, R. Lee, and X. Zhang, "Spark-GPU: An accelerated in-memory data processing engine on clusters," in *IEEE Big Data*, 2016, pp. 273–283.

[29] S. Ramos and T. Hoefler, "Capability models for manycore memory systems: A case-study with xeon phi KNL," in *IPDPS*, 2017, pp. 297–306.

[30] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed GraphLab: A framework for machine learning in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[31] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *CVPR*, 2016, pp. 5308–5317.

[32] P. A. Szekely *et al.*, "Building and using a knowledge graph to combat human trafficking," in *ISWC*, 2015, pp. 205–221.

[33] M. Koubarakis, C. Kontoes, and S. Manegold, "Real-time wildfire monitoring using scientific database and linked data technologies," in *EDBT/ICDT*, 2013, pp. 649–660.

[34] A. Fornito, A. Zalesky, and M. Breakspear, "Graph analysis of the human connectome: Promise, progress, and pitfalls," *NeuroImage*, vol. 80, pp. 426–444, 2013.

[35] A. L. Hopkins, "Network pharmacology: the next paradigm in drug discovery," *Nature chemical biology*, vol. 4, no. 11, p. 682, 2008.

[36] A. D. Broido and A. Clauset, "Scale-free networks are rare," *arXiv preprint arXiv:1801.03400*, 2018.

[37] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *CUG*, vol. 19, pp. 45–74, 2010.

[38] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *ICDM*, 2004, pp. 442–446.

[39] A. Prat-Pérez and D. Dominguez-Sal, "How community-like is the structure of synthetically generated graphs?" in *GRADES*, 2014, pp. 1–9.

[40] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz, "The LDBC Social Network Benchmark: Interactive workload," in *SIGMOD*, 2015, pp. 619–630.

[41] S. Beamer, K. Asanovic, and D. A. Patterson, "Locality exists in graph processing: Workload characterization on an ivy bridge server," in *IISWC*, 2015, pp. 56–65.

[42] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *SIGMOD*, 2010, pp. 135–146.

[43] S. Venkataraman, E. Bodzsar, I. Roy, A. AuYoung, and R. S. Schreiber, "Presto: distributed machine learning and graph processing with sparse matrices," in *EuroSys*, 2013, pp. 197–210.

[44] A. Iosup, T. Hegeman, W. L. Ngai, S. Heldens, A. Prat-Pérez, T. Manhardt, H. Chafi, M. Capota, N. Sundaram, M. J. Anderson, I. G. Tanase, Y. Xia, L. Nai, and P. A. Boncz, "LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1317–1328, 2016.

[45] Y. Guo, S. Hong, H. Chafi, A. Iosup, and D. Epema, "Modeling, analysis, and experimental comparison of streaming graph-partitioning policies," *J. of Par. Distrib. Computing*, vol. 108, pp. 106–121, 2017.

[46] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

[47] M. Paredes, G. Riley, and M. Luján, "Breadth first search vectorization on the Intel Xeon Phi," in *CF*, 2016, pp. 1–10.

[48] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *IISWC*, 2015, pp. 44–55.

[49] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, 2014.

[50] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[51] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results," in *Supercomputing, SC*, 2015, pp. 73:1–73:12.

[52] Cloudera, "Cloudera Best Practices," http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/, 2015, [Online, accessed: May, 2018].

[53] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012, pp. 2–2.

[54] T. White, *Hadoop: The definitive guide*. "O'Reilly Media, Inc.", 2012.

[55] J. Peltenburg, A. Hesam, and Z. Al-Ars, "Pushing big data into accelerators: Can the JVM saturate our hardware?" in *High Performance Computing - ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^3MA, VHPC, Visualization at Scale, WOPSSS, Revised Selected Papers*, 2017, pp. 220–236.

[56] E. Totoni, S. R. Dulloor, and A. Roy, "A case against tiny tasks in iterative analytics," in *Proceedings of the 16th HotOS*, 2017, pp. 144–149.

[57] H. Lin, Z. Su, X. Meng, X. Jin, Z. Wang, W. Han, H. An, M. Chi, and Z. Wu, "Combining hadoop with mpi to solve metagenomics problems that are both data-and compute-intensive," *International Journal of Parallel Programming*, pp. 1–14, 2017.

[58] A. J. Awan, "Performance characterization and optimization of in-memory data analytics on a scale-up server," Ph.D. dissertation, KTH Royal Institute of Technology, 2017.

[59] R. A. de Carvalho, A. Goldman, and G. G. H. Cavalheiro, "Is intel high performance analytics toolkit a good alternative to apache spark?" in *16th NCA*, 2017, pp. 171–178.

[60] A. Haidar, H. Jagode, A. YarKhan, P. Vaccaro, S. Tomov, and J. J. Dongarra, "Power-aware computing: Measurement, control, and performance analysis for intel xeon phi," in *HPEC*, 2017, pp. 1–7.

[61] 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 - 8th International Workshop, PMBS 2017, Proceedings*, 2017, pp. 236–249.

[62] C. Rosales, J. Cazes, K. Milfeld, A. Gómez-Iglesias, L. Koesterke, L. Huang, and J. Vienne, "A comparative study of application performance and scalability on the intel knights landing processor," in *High Performance Computing - ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P^3MA, VHPC, WOPSSS, Revised Selected Papers*, 2016, pp. 307–318.

[63] A. Li, W. Liu, M. R. B. Kristensen, B. Vinter, H. Wang, K. Hou, A. Marquez, and S. L. Song, "Exploring and analyzing the real impact of modern on-package memory on HPC scientific kernels," in *Supercomputing, SC*, 2017, pp. 26:1–26:14.

[64] D. Doerfler, B. Austin, B. Cook, J. Deslippe, K. Kandalla, and P. Mendygral, "Evaluating the networking characteristics of the cray XC-40 intel knights landing-based cori supercomputer at NERSC," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 1, 2018.

[65] M. Jacquelin, W. A. de Jong, and E. J. Bylaska, "Towards highly scalable ab initio molecular dynamics (AIMD) simulations on the intel knights landing manycore processor," in *IPDPS*, 2017, pp. 234–243.

[66] J. Tobin, A. Breuer, A. Heinecke, C. Yount, and Y. Cui, "Accelerating seismic simulations using the intel xeon phi knights landing processor," in *High Performance Computing - 32nd International Conference, ISC High Performance, Proceedings*, 2017, pp. 139–157.

[67] M. Paredes, G. D. Riley, and M. Luján, "Vectorization of hybrid breadth first search on the intel xeon phi," in *Proceedings of the Computing Frontiers Conference*, 2017, pp. 127–135.

[68] X. Liu, L. Chen, J. S. Firoz, J. Qiu, and L. Jiang, "Performance characterization of multi-threaded graph processing applications on intel many-integrated-core architecture," *arXiv preprint arXiv:1708.04701*, 2017.

[69] A. Trivedi, P. Stuedi, J. Pfefferle, A. Schüpbach, and B. Metzler, "Albis: High-performance file format for big data systems," in *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018.*, 2018, pp. 615–630. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/trivedi