ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds

Heng Lin[1,2], Xiaowei Zhu[1,5], Bowen Yu[1], Xiongchao Tang[1,5], Wei Xue[1], Wenguang Chen[1], Lufei Zhang[3], Torsten Hoefler[4], Xiaosong Ma[5], Xin Liu[6], Weimin Zheng[1], and Jingfang Xu[7]

***Abstract*—Graphs are an important abstraction used in many scientific fields. With the magnitude of graph-structured data constantly increasing, effective data analytics requires efficient and scalable graph processing systems. Although HPC systems have long been used for scientific computing, people have only recently started to assess their potential for graph processing, a workload with inherent load imbalance, lack of locality, and access irregularity. We propose ShenTu[8], the first general-purpose graph processing framework that can efficiently utilize an entire Petascale system to process multi-trillion edge graphs in seconds. ShenTu embodies four key innovations: hardware specialization, supernode routing, on-chip sorting, and degree-aware messaging, which together enable its unprecedented performance and scalability. It can traverse a record-size 70-trillion-edge graph in seconds. Furthermore, ShenTu enables the processing of a spam detection problem on a 12-trillion edge Internet graph, making it possible to identify trustworthy and spam webpages directly at the fine-grained page level.**

***Index Terms*—Application programming interfaces; Big data applications; Data analysis; Graph theory; Supercomputers**

I. Justification for ACM Gordon Bell Prize

ShenTu enables highly efficient general-purpose graph processing with novel use of heterogeneous cores and extremely large networks, scales to the full TaihuLight, and enables graph analytics on 70-trillion-edge graphs. It computes PageRank and TrustRank distributions for an unprecedented 12-trillion-edge real-world web graph in 8.5 seconds per iteration.

II. Performance Attributes

Performance Attributes	Content
Category of achievement	*Scalability, Time-to-solution*
Type of method used	*Framework for graph algorithms*
Results reported based on	*Whole application including I/O*
Precision reported	*Mixed precision (Int and Double)*
System scale	*Measured on full-scale system*
Measurement mechanism	*Timers & Code Instrumentation*

[1]Tsinghua University. Email: {*xuewei,cwg,zwm-dcs*}*@tsinghua.edu.cn*, {*linheng11,zhuxw16,yubw15,txc13*}*@mails.tsinghua.edu.cn*
[2]Fma Technology
[3]State Key Laboratory of Mathematical Engineering and Advanced Computing. Email: *zhang.lufei@meac-skl.cn*
[4]ETH Zurich. Email: *htor@inf.ethz.ch*
[5]Qatar Computing Research Institute. Email: *xma@qf.org.qa*
[6]National Research Centre of Parallel Computer Engineering and Technology. Email: *yyylx@263.net*
[7]Beijing Sogou Technology Development Co., Ltd. Email: *xujingfang@sogou-inc.com*
[8]ShenTu means "magic graph" in Chinese.

III. Overview of the Problem

Graphs are one of the most important tools to model complex systems. Scientific graph structures range from multi-billion-edge graphs (e.g., in protein interactions, genomics, epidemics, and social networks) to trillion-edge ones (e.g., in connectomics and internet connectivity). Timely and efficient processing of such large graphs is not only required to advance scientific progress but also to solve important societal challenges such as detection of fake content or to enable complex data analytics tasks, such as personalized medicine.

Improved scientific data acquisition techniques fuel the rapid growth of large graphs. For example, cheap sequencing techniques lead to massive graphs representing millions of human individuals as annotated paths, enabling quick advances in medical data analytics [1]. For each individual, human genome researchers currently assemble de Bruijn graphs with over 5 billion vertices/edges [2]. Similarly, connectomics models the human brain, with over 100 billion neurons and an average of 7,000 synaptic connections each [3].

Meanwhile, researchers face unprecedented challenges in the study of human interaction graphs. Malicious activities such as the distribution of phishing emails or fake content, as well as massive scraping of private data, are posing threats to human society. It is necessary to scale graph analytics with the growing online community to detect and react to such threats. In the past 23 years, the number of Internet users increased by 164× to 4.2 billion, while the number of domains grew by nearly 70,000× to 1.7 billion. Sogou, one of the leading Internet companies in China, crawled more than 271.9 billion Chinese pages with over 12.3 trillion inter-page links in early 2017 and expects a 4× size increase with whole-web crawling.

Graph processing and analytics differ widely from traditional scientific computing: they exhibit significantly more *load imbalance*, *lack of locality*, and *access irregularity* [4]. Modern HPC systems are designed for workloads that have some degree of locality and access regularity. It is thus natural to map regular computations, such as stencil-based and structured grid-based, to complex architectures achieving high performance, as recent Gordon Bell Prize winners demonstrated. However, as witnessed by the set of recent finalists, very few projects tackle the challenge of completely irregular computations at largest scale. We argue that the science drivers outlined above as well as the convergence of data science, big data, and HPC necessitate serious efforts to enable graph computations on leading supercomputing architectures.

SC18, November 11-16, 2018, Dallas, Texas, USA
978-1-5386-8384-2/18/$31.00

TABLE I
SUMMARY OF STATE-OF-THE-ART GRAPH FRAMEWORKS AND THEIR REPORTED PAGERANK RESULTS

Year	System	In-memory or out-of-core	Synthetic or Real-world	Platform	Max. number of edges	Time for one PageRank iteration	Performance (GPEPS)
2010	Pregel	In-memory	Synthetic	300 servers	127 billion	n/a	n/a
2015	Giraph	In-memory	Real-world	200 servers	1 trillion	less than 3 minutes	>5.6
2015	GraM	In-memory	Synthetic	64 servers	1.2 trillion	140 seconds	8.6
2015	Chaos	Out-of-core	Synthetic	32 servers (480GB SSD each)	1 trillion	4 hours	0.07
2016	G-Store	Out-of-core	Synthetic	1 server with 8x512GB SSDs	1 trillion	4214 seconds	0.23
2017	Graphene	Out-of-core	Synthetic	1 server with 16x500GB SSDs	1 trillion	about 1200 seconds	0.83
2017	Mosaic	Out-of-core	Synthetic	1 server with 6 NVMe SSDs	1 trillion	1247 seconds	0.82

In order to execute trillion-edge graph processing on leading supercomputers, one needs to map the imbalanced, irregular, and non-local graph structure to a complex and rather regular machine. In particular, naive domain partitioning would suffer from extreme load imbalance due to the power-law degree distribution of vertices in real-world graphs [5]. For example, the in-degrees in the Sogou web graph vary from zero to three billion. This load imbalance can be alleviated by either vertex randomization or edge partitioning, yet, both techniques increase inter-process communication in many graph algorithms. The challenge is further compounded by system features not considered by widely used distributed graph processing frameworks, including the high core counts, complex memory hierarchy, and heterogeneous processor architecture within each node on modern supercomputers.

Data scientists tweak algorithms in agile development frameworks and require quick turnaround times to evolve their experiments with ever growing data volumes. We squarely address all these challenges with ShenTu, the first general-purpose graph processing framework to utilize machines with ten millions of cores. It shares a vertex-centric programming model with established small-scale graph processing frameworks [6], [7], allowing data scientists to express arbitrary graph algorithms via familiar APIs. In particular, as a general-purpose graph framework, ShenTu is within a factor of two of the breadth first search (BFS) performance of the current 2nd entry in the Graph500 list. When disabling optimizations that *only* apply to the BFS benchmark, ShenTu's performance is equal to this highly-optimized record-holding code.

ShenTu adopts well-known optimizations of parallel graph frameworks, such as message coalescing, combining, and routing, randomized vertex assignment, and automatic push/pull optimizations to large-scale systems. Furthermore, ShenTu introduces the following key innovations advancing the state of the art of adopting highly irregular graph computations to hierarchical Petascale systems such as Sunway TaihuLight: *hardware specialization* selects the best (heterogeneous) compute unit and memory for each task; *supernode routing* adapts the irregular global communication to the machine's (supernode) topology; *on-chip sorting* maps the irregular local communication to manycore processors; and *degree-aware messaging* selects the best communication scheme according to vertex properties, such as its degree. All those optimizations enable ShenTu to execute various graph analytics algorithms for Sogou's webpage graph with the unprecedented 12-trillion edges in 8.5 seconds running one iteration of PageRank utilizing all nodes of Sunway TaihuLight.

IV. CURRENT STATE OF THE ART

Driven by the increasing significance of data analytics, numerous graph processing frameworks were introduced in recent years. Here, we focus on related frameworks that either aim at high performance or large-scale graphs. We adopt a simple metric, *processed edges per second (PEPS)* (see Section VII for detail), as a pendant to the well-known FLOPS metric to gauge the performance in graph processing.

Large-scale graph processing: Table I summarizes published large-scale frameworks and results, based on PageRank, one of the most important graph algorithms [8]. It has been studied and optimized in hundreds of research papers and it is the gold standard for comparing graph processing frameworks. Among such frameworks, Google Pregel [9] introduced a vertex-centric abstraction in 2010, computing the PageRank of a synthetic graph with 127 billion edges on 300 machines. In 2015, Facebook scaled Giraph [10] to complete one PageRank iteration on a trillion-edge graph in "less than 3 minutes". In the same year, GraM [11] uses an RDMA-based communication stack tuned for multi-core machines to process a similar graph on an InfiniBand cluster, outperforming the Giraph result with less than one third of the nodes. The achieved 8.6 GPEPS remains the fastest trillion-edge PageRank throughput reported to date.[1] Recently, out-of-core systems, such as Chaos [13], G-Store [14], Graphene [15], and Mosaic [16] enable trillion-edge graph processing on small numbers of nodes, however with more than an order of magnitude lower performance.

Our work advances the problem scale by one order of magnitude and performance by 2-3 orders of magnitude. ShenTu enables in-memory processing of synthetic graphs with up to 70 trillion edges. We also demonstrate performance of up to 1,984.8 GPEPS on 38,656 compute nodes. [2] In addition, we show the first large-scale spam-score study of 271.9 billion webpages and 12.3 trillion links between them.

Other graph processing systems: For processing moderately sized graphs (billion-edge level), popular shared-memory frameworks include Galois [17], Ligra [6], Polymer [18], and

[1] Gemini [7] and STAPL [12] achieved 27.4 GPEPS and 10 GPEPS, respectively, for much smaller graphs ($\leq$64 billion edges). Processing larger graphs increases communication and thus reduces performance.

[2] ShenTu is able to scale to the whole 40,960 nodes in Sunway TaihuLight, limited by 38,656 available nodes during test.

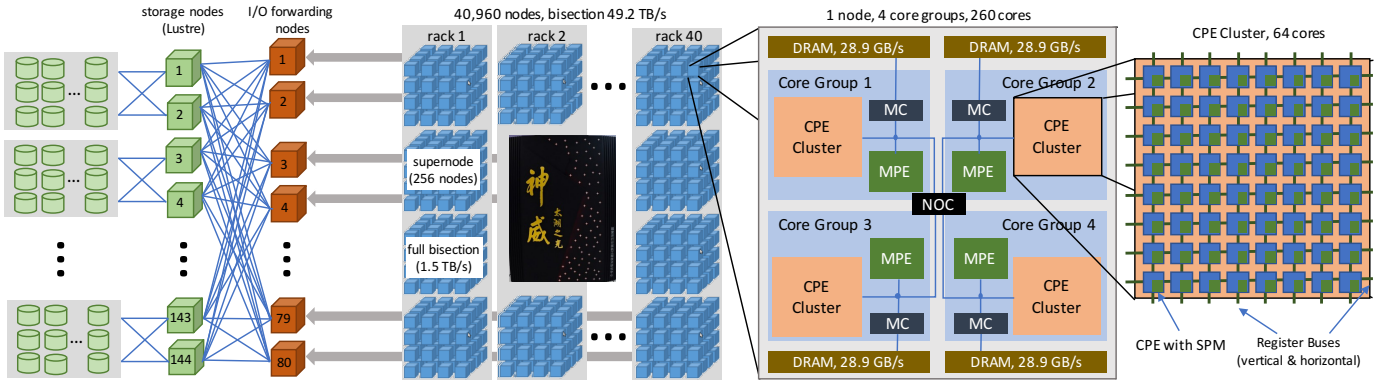

Fig. 1. TaihuLight System and CPU architecture. The left part shows the I/O system and the right part shows details of the heterogeneous compute elements.

GraphMat [19]. To provide additional capacity or throughput, researchers typically use (1) out-of-core frameworks, such as GraphChi [20], X-Stream [21], FlashGraph [22], and GridGraph [23], or (2) distributed frameworks, such as GraphLab [24], PowerGraph [25], PowerLyra [26], and PGX.D [27]. In addition to the inferior performance of out-of-core systems similar to that shown in Table I, existing distributed graph systems also suffer from poor performance or scalability [7]. Even HPC-style frameworks, such as the Parallel Boost Graph library, fail to scale beyond hundreds of nodes [28]. ShenTu, on the other hand, efficiently processes large graphs on 38,656 compute nodes using millions of cores.

Graph benchmarks on supercomputers: Recognizing the growing importance of high-performance graph processing, the Graph500 breadth first search (BFS) benchmark ranking was started in 2010. This competition facilitated the development of highly-specialized BFS implementations on cutting-edge supercomputers [29], [30], [31], often leveraging BFS-specific characteristics such as idempotent updates and message compression that cannot be generalized to general graph algorithms (such as PageRank or WCC). In contrast, ShenTu allows users to define arbitrary processing tasks and handles real-world graphs with significantly higher skewness than synthetic graphs used in Graph500 tests (cf. Section VII-A).

We conclude that achieving highest performance for graph processing on large-scale supercomputers remains challenging and no scalable general-purpose framework has been demonstrated on millions of cores of a Petascale machine.

Scientific computing programming frameworks: Multiple recent Gordon Bell finalists present common programming frameworks that improve user productivity on modern supercomputers, for problems such as adaptive mesh refinement (AMR) [32], computational fluid dynamics (CFD) [33], N-Body [34], and stencil-based computation [35]. ShenTu is in line with such spirit, requiring $<$20 lines of application code to implement common tasks such as BFS/PageRank/WCC. Behind its intuitive and expressive APIs, 22,000 lines of ShenTu code implement efficient and adaptive optimizations to obtain massive processing throughput and parallel efficiency.

V. System Architecture Challenges

Before describing ShenTu's innovations, we introduce the Sunway TaihuLight system [36], [37], which is optimized for traditional scientific computing applications and ever ranked No.1 in Top500 list during June 2016 to June 2018. TaihuLight's complex architecture is shown in Figure 1; it poses several challenges to the construction of a scalable, general-purpose parallel graph processing framework. We design microbenchmarks to measure ideal peak bandwidths under laboratory conditions (empty system etc.) and show the results compared to hardware peak numbers in Table II. We will later use these ideal measured numbers to show that ShenTu achieves close-to-peak performance.

TABLE II
TaihuLight specifications and bandwidth measured

Component	Configurations	BW measured (% peak)
MPE	1.45 GHz, 32/256KB L1/L2	DRAM 8.0 GB/s (80%)
CPE	1.45 GHz, 64KB SPM	Reg. 630 GB/s (85%)
CG	1 MPE + 64 CPEs	DRAM 28.98 GB/s (85%)
Node	1 CPU (4 CGs), 4×8GB RAM	Net: 6.04 GB/s (89%)
Super Node	256 nodes, FDR 56 Gbps IB	Bisection 1.5 TB/s (72%)
TaihuLight	160 supernodes	Bisection 49.2 TB/s (68.6%)
Agg. memory	1.3 PB	4.6 PB/s (85%)
Storage	5.2 PB (Online2)	70 GB/s (54.35%)

Processors and memory hierarchy Each TaihuLight compute node has a single heterogeneous SW26010 CPU (see right part of Figure 1), composed by four core groups (CGs) connected via a low-latency on-chip network (NoC). Each CG consists of a management processing element (MPE), a 64-core computing processing element (CPE) cluster, and a memory controller (MC); a total of 260 cores per CPU (node).

The MPE has a 64-bit RISC general-purpose core capable of superscalar processing, memory management, and interrupt handling. The CPEs are simplified 64-bit RISC accelerators for high compute performance at low energy consumption. Both have single-cycle access latency to their register set. CPEs do not have caches, but each comes with a 64KB scratch pad memory (SPM) that requires explicit programmer control, similar to shared memory in GPUs. Those require a manual orchestration of all data movement, a challenging task for irregular random accesses. The CPEs also support direct DRAM load/store, but the cache-less design makes it rather inefficient, as every operation accesses a full 256-Byte DRAM row to move data to/from a single 64-bit register.

On-chip communication SPMs are private, thus CPEs can only synchronize through inefficient atomic DRAM accesses, often leading to severe contention. However, the 64 CPEs in the same cluster are connected with an 8×8 mesh topology

through register buses (cf. Figure 1). CPEs in the same row or column can exchange data using fast register communication. Each can send one 256-bit message each cycle and messages arrive after 10 cycles at the destination. Switching from row-direction to column-direction or vice-versa requires software routing. While this mechanism is extremely fast, programmers must avoid deadlocks in software.

Interconnection network The TaihuLight compute nodes are connected via a 2-level InfiniBand network. A single-switch with full bisection bandwidth connects all 256 nodes within a *supernode*, while a fat-tree with 1/4 of the full bisection bandwidth connects all supernodes (cf. Figure 1). Table II shows measurements of bisection communication bandwidth at different levels of the system. To fully utilize this hierarchical network, our graph framework must adapt to the structure of the fast intra-supernode network and perform aggressive message batching for better small-message and software routing performance.

I/O system The TaihuLight compute nodes connect to its Lustre file system via a flexible I/O forwarding layer, shown in the left part of Figure 1. I/O requests from four *super nodes* (1,024 compute nodes) are currently handled by two I/O forwarding nodes. ShenTu runs use the *Online2* Lustre partition dedicated to large-scale, I/O-intensive jobs. It is composed by 80 I/O forwarding nodes, 144 storage nodes (Lustre OSSs) and 72 Sugon DS800 disk arrays, each containing 60 1.2 TB SAS HDD drives. Limitations of Lustre only allow to reach 70 GB/s even with ideal access patterns. ShenTu is currently the most I/O bandwidth intensive application on TaihuLight, reading in real-world large graphs (total volume up to 100s of TBs) and demanding efficient and scalable parallel file I/O as well.

VI. Innovations Realized

ShenTu is the *first general-purpose graph processing framework* targeting the efficient utilization of entire Petascale systems. On top of HPC technologies such as MPI and athreads, the Sunway lightweight threading library, we extend methodologies from distributed graph processing to attack unique problems. More specifically, to conquer efficiency and scalability challenges brought by the combination of problem and machine scale, irregularity in computation, and highly heterogeneous compute node architecture, ShenTu realizes four key innovations: *hardware specialization*, *supernode routing*, *on-chip sorting*, and *degree-aware messaging*. Overall, ShenTu focuses on efficient and adaptive data routing, in different forms at different levels of the system organization (from CPE cores to the entire supercomputer), to enable fast parallel execution of highly irregular graph applications.

ShenTu offers a simple and intuitive programming interface similar to Ligra [6] and Gemini [7] with its two main functions: `mapVertices` and `mapEdges`. Both routines take input with a subset of *active vertices* and a user-defined function for processing vertices or edges, respectively. The computation advances in bulk synchronous steps until the active vertex set is empty. This enables scientists to quickly implement algorithms in tens of lines of code. For example, a 20-line BFS implementation in ShenTu performs within a factor of two of a 9,000-line highly-specialized not generalizable implementation [30]. The framework itself has approximately 22,000 lines of code to enable all the optimizations.

A. Key innovations for petascale systems

We first present our four key innovations to high efficiency for large graph processing on millions of cores. This discussion assumes a partitioned input graph to all compute nodes, while we give more details on our partitioning methodology and optimizations later in Section VI-B.

Innovation 1. Hardware specialization Leading supercomputers like TaihuLight increasingly adopt heterogeneous hardware. Within each SW26010 CPU, at the coarsest level we assign the four core groups into distinct functions as shown in Figure 2: (A) Generation, (B) Relay, (C1) Coarse sort, and (C2) Update. The functions, each carried out by one of the four CPE clusters, are strategically designed to achieve balanced core group utilization and peak-bandwidth processing of irregular data. This spatially pipelined architecture allows us to process batched data in a streaming way, obtaining lower I/O complexity to main memory and higher utilization of the massive on-chip bandwidth.

At the second level of specialization, we consider the specific hardware capabilities within each core group. Here the MPE is well suited for task management, plus network and disk I/O, while the CPEs are tightly connected through the register communication feature. This naturally leads us to execute communication tasks on the MPE and data sorting and processing tasks on the CPEs. The left part of Figure 2 illustrates the overall specialization and pipelining, with MPEs 1-4 and their corresponding CPE clusters 1-4, respectively. The aforementioned per-CPE-cluster functions are tuned to stream graph updates through this pipeline as follows.

***Generation*:** Linearly scans vertex partition assigned to this node, identifies active vertices, and generates messages to the destination nodes for the next stage.

***Relay*:** Routes coalesced messages for high-throughput inter-node communication (cf. Innovation 2).

***Coarse sort*:** Performs the first-stage, approximate sorting, to organize messages into buckets, each fitting into the fast SPM to be forwarded to the next stage (cf. Innovation 3).

***Update*:** Performs the last processing stage, the final sorting within each bucket and updates (now consecutive) destination vertices with the user-specified function.

In summary, ShenTu specializes not only to the different capabilities of the heterogeneous processors, but also implements an innovative spatial pipelining strategy that utilizes heterogeneous many-core chips for streaming graph processing.

Innovation 2. Supernode routing This technique targets efficient inter-node communication, extending our heterogeneous processing pipeline to the full system.

Intuitively, distributed graph applications are plagued by large numbers of small messages, sent following the graph topology, often with an all-to-all connectivity among the partitions. Sending small messages to 10,000s of destinations

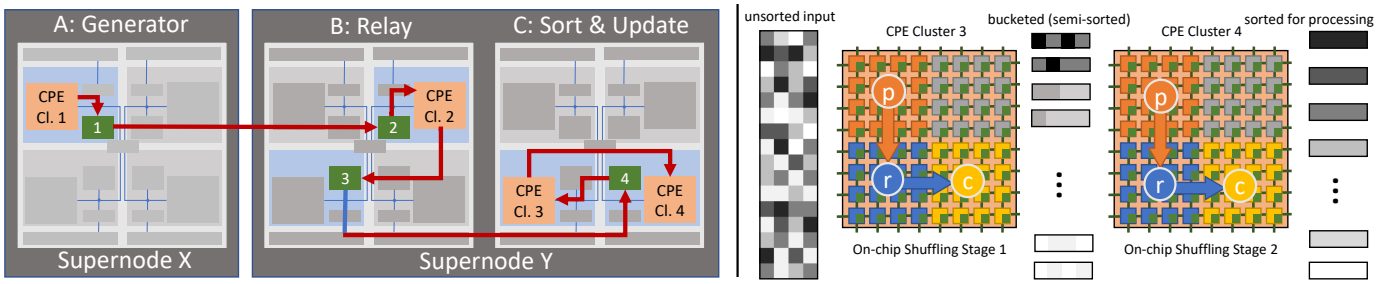

Fig. 2. Supernode routing (left) and on-chip sorting (right).

is inefficient due to static per-message overheads (routing information, connection state, etc.) ShenTu mitigates this by factoring all compute nodes into groups, by their supernode affiliation. Each node coalesces all messages to nodes within the same target group into a single message, sent to a designated node within that group. This so-called *relay node* unpacks the received messages and distributes them to appropriate peers.

Our relaying design needs to meet several requirements. To maximize utilization of the higher intra-supernode bandwidth, we form target groups using supernode boundaries. To effectively perform message aggregation, each source node minimizes the number of relay nodes it sends to within a target group. To achieve load balance, each node in a target group acts as a relay node.

Due to regular maintenance and failures, some supernodes contain less than 256 operational nodes. A naive mapping strategy could lead to relay nodes that receive twice the "normal" load. Instead, we use a simple stochastic assignment algorithm that distributes the load evenly among all active nodes in each supernode. Supernode routing greatly reduces the number of messages and source-destination pairs. Messages arriving at the corresponding supernode can quickly be distributed to the target host due to the full bandwidth in each supernode.

Figure 2 illustrates this: Node A sends a message to Node C, relayed via Node B (Nodes B and C are in the same supernode). CPE cluster 1 on Node A generates messages and notifies MPE 1 to send. MPE 2 on Node B receives messages and CPE cluster 2 performs destination-based on-chip sorting and combining on message. MPE 3 then sends the resulting coalesced message to Node C. The messages, targeted at vertices in Node C are received by MPE 4, which notifies CPE clusters 3 and 4 to sort and update the destination vertices, respectively. This way, ShenTu's routing seamlessly couples with its on-node processing pipeline, orchestrating an efficient global communication scheme that fully utilizes all compute elements and their bandwidth at different domains of the whole system.

Innovation 3. On-chip sorting Most graph computations are bounded by memory bandwidth and similar to inter-node communication, small random memory accesses hurt performance. For each CPE cluster, peak memory bandwidth can only be achieved if 16 or more CPEs issue at least 256-Byte DMA transfers between SPM and DRAM. For example, accessing only 128 Bytes or using only 8 CPEs leads to a bandwidth drop below 16 GB/s from the peak 28.9 GB/s.

Messages arriving at Node C contain vertex updates in a random order. ShenTu uses on-chip sorting to assign disjoint partitions of destination vertices to different CPE cores to obviate expensive DRAM synchronizations. It also aggregates messages updating the same vertex, increasing data reuse and reducing both DRAM traffic and random accesses. To this end, ShenTu carries its specialization approach further into each CG: It partitions the homogeneous CPE cores into three types, shown in the right part of Figure 2: (p) *producers* for reading data, (r) *routers* for sorting data, and (c) *consumers* for applying vertex updates or passing data to the MPE for sending. As shown in the figure, ShenTu assigns square blocks of 16 CPEs as producers, routers, and consumers, respectively. This ideal configuration, leaving 16 CPEs for other tasks, achieves the maximum CG sorting bandwidth of 9.78 GB/s.

The *producer CPEs* read messages in consecutive 256-Byte blocks received by the MPE from DRAM. The *router CPEs* leverage the fast register communication, receiving unsorted vertex updates from producers, and sorting them using register communication. Sorted buckets are sent to the appropriate *consumer CPEs*, who in the sort stage write the buckets to DRAM. The ones in the final (update) stage apply the user-defined function to their disjoint sets of 8-Byte vertices cached in their SPMs. Thus the aggregate processing capacity per CG will be $16 \times 64\,kB/8\,B = 2^{17}$ vertices.

We use two-stage sorting as a CG cannot sort its whole 32GB DRAM in a single pass due to the SPM size (64kB). The first stage sorts into buckets and the second sorts each bucket. One bucket buffer in SPM is 256B, to enable fast DRAM transfer from CPE cluster 3 to cluster 4. This results in $16 \times 64\,kB/256\,B = 2^{12}$ buckets, where CPEs in cluster 3 read data (p), coarse sort (r), and write buckets to DRAM (c). Our two-pass sorting supports $2^{12} \times 2^{17} = 2^{29}$ vertices per compute node, enough to fully sort the 32GB DRAM.

Innovation 4. Degree-aware messaging The in and out-degrees of typical real-world graphs follow a power-law distribution. Thus, in distributed graph processing, high-degree vertices send or receive a large number of messages to or from many neighbors, often covering all compute nodes. For example, the Sogou graph has 24.4 million vertices (out of 271.9 billion in total, less than 0.001%) that have an in-degree over 40,960, and these edges cover 60.48% of all the edges in the graph. This means, when processing it on 40,960 nodes, those vertices may reach the whole system.

Recognizing this, ShenTu's design singles out these high-degree vertices by *replicating* them across all compute nodes,

creating *mirrors* vs. the *primary copy* owned by the compute node originally assigned to process such a vertex. ShenTu further differentiates between vertices with high out- and high in-degrees, denoted as *hout* and *hin* vertices, respectively.

With high-degree vertices mirrored at every compute node, ShenTu handles messages to and from these vertices differently. Instead of individual hout vertices sending messages to update their many neighbors, all compute nodes cooperate in a single `MPI_Bcast` call to update the mirrors at all nodes. The mirrors then act as per-node delegates, updating local neighbors of each hout vertex. For hin vertices the process is similar, except that the mirrors first collect updates to an hin vertex mirror, using a user-specified `combineEdgeMsg` function, similar to local combiners in Pregel or MapReduce. Then they aggregate mirrors using an `MPI_Gather` or `MPI_Reduce` collective call to update the primary copy.

The respective degree thresholds for the hout or hin lists are determined by considering the graph and job scale (details in Section VIII-C). A vertex can be both hin and hout, only hin, only hout, or *regular*. Regular vertices have their edges stored at the source, with messages sent to compute nodes owning their neighbors using the described intra- and inter-node pipelines. Unlike with hin or hout vertices, ShenTu does not attempt message combination with regular ones, as there is little benefit.

Our degree-aware message passing has several benefits: (1) it leverages the powerful supercomputer interconnect, highly optimized for collective operations; (2) it involves *all* compute nodes in collective operations for the high-degree vertices, which further reduces the number of messages; (3) it balances load because local mirrors distribute or collect updates to high-degree vertices, which essentially shares the workload of those vertices among all compute nodes, and (4) it simplifies the design with its "all-or-none" replication policy (high-degree vertices are replicated to all compute nodes), which avoids the space-costly replica lists used in many distributed graph frameworks [25], [38], [26]. We evaluate the effectiveness of the load balancing schemes in Section VIII-B.

B. Auxiliary Tasks and Optimizations

In addition to the above key innovations enabling ShenTu's efficiency and scalability, it adopts existing techniques, that were proven at small scales, but *require* extensions and optimizations to scale them orders of magnitude further.

Direction optimization (push vs. pull) Graph processing frameworks can choose to propagate updates along edges in two modes: *push* (where source vertices send messages to update their neighbors) or *pull* (where destination vertices solicit updates from their neighbors) [39], [40], [6], [7]. Similar to frameworks such as Gemini, ShenTu implements both modes and automatically switches between them at runtime. More specifically, ShenTu constantly monitors the fraction of all edges that are *active*, i.e., to be visited in the current iteration. A low fraction of active vertices indicates a "sparse" iteration, to be processed efficiently with push, while a high fraction is best processed with pull for high-degree vertices and configurable push/pull for regular ones according to application characteristics. Such dual-mode execution leads to significantly fewer processed edges and update conflicts. A typical threshold is 1/20 [6], [7], while ShenTu adopts 1/35 as pull performs better with TaihuLight's architecture, especially at large scales. In addition, ShenTu's push-pull mode is integrated with the aforementioned degree-aware optimization: each mode with its specific hin/hout/reg handling.

Graph loading We store all real-world graphs as unsorted edge lists on disk. For most graph algorithms, in-memory processing is much faster than loading the terabyte-scale graphs from disk. To maximize the Lustre I/O bandwidth utilization, we split the input data into 100,000s of 1 GB files and coordinate parallel I/O in a grouped manner to reduce I/O resource contention and I/O synchronization. We further arrange those files in a balanced directory structure, designed to ease the load on the Lustre metadata server.

Randomized vertex partitioning Sophisticated partitioning schemes such as (Par)Metis [41] are more expensive than many graph analytics because balanced partitioning is itself an NP-hard graph problem. Complementary to the global replication of high-degree vertices, ShenTu uses a lightweight randomized partitioning method for all primary vertices. To avoid the space and time overheads of vertex-to-node maps, we store the destination of each edge as the tuple (destination node id, local vertex id on that node). All-in-all, our partitioning scheme delivers well-balanced load in all our experiments[3].

VII. How Performance Was Measured

The performance of many scientific computing applications can be characterized by the achieved floating point rate (such as PFLOPS). This makes it simple to compare the achieved performance with the theoretical maximum (peak) performance of a particular computer. In the comparatively young field of graph analytics, no such simple metric can be applied because the computation (be integer or floating point) is often insignificant with respect to the cost of handling the irregular data. The Graph500 benchmark defines the "traversed edges per second" (TEPS) metric, which is computed for BFS by dividing the total number of graph edges by the computation time. This metric has been debated as for direction-optimizing BFS, not all edges are traversed [42]. Furthermore, the metric only applies to label-setting graph traversal algorithms like BFS. PageRank, e.g., does not "traverse" edges. Thus, we measure our performance by counting the *actual processed edges per second (PEPS)*. E.g., in a single PageRank iteration, each edge is processed exactly once, while a label-correcting SSSP may process the same edge multiple times and direction-optimizing BFS may skip many edges.

For both TEPS and PEPS, it remains challenging to define a peak machine performance due to the fundamentally irregular data movement. A trivial upper bound is the cumulative memory bandwidth divided by the storage size of an edge. In ShenTu's case, vertex IDs are 8-byte values, resulting in

[3]Without this technique, nodes ran out of memory during preprocessing.

approximately (4.6 PB/s)/(8 bytes) = 575 TPEPS. However, this bound is not useful because real-world graphs are strongly connected and hard to cut for parallel computation [4]. Thus, we use the *bisection bandwidth divided by the graph cut-width (into the number of distributed memory partitions at the target machine)* as a better upper bound. This upper bound provides a strong indicator for how well a graph application uses a machine, with a certain partitioning strategy. To measure the edge cut, we instrumented ShenTu's communication subsystem to collect transmitted data volume.

A. Data sets

We evaluate ShenTu on several real-world and synthetic large-scale graphs (Table III). We store all real-world graphs with a compact edge list format on disk, using 48 bits per vertex. For RMAT and Kronecker generation, ShenTu follows the Graph500 reference code (version 2.1.4 with parameters A=0.57, B=0.19, C=0.19, D=0.05) and uses 16 as edge factor. The Erdős-Rényi generator uses the $G(n, m)$ rather than $G(n, p)$ model [43]. As per Graph500 convention, we use "scale x" to refer to a Graph500 synthetic graph with 2^x vertices. our main study object is the 136.9TB Sogou Chinese web graph with 12.3 trillion edges. As Sogou expects a 4× graph size increase with full-web crawling, we also demonstrate a 70-trillion-edge Kronecker graph.

TABLE III
SPECIFICATION OF SHENTU TEST DATASETS

Graph	Vertices	Edges	Size on Disk
Twitter [44]	41.7m	1.47b	16GB
UK-2007 [45]	105.9m	3.74b	41GB
Weibo [46]	349.7m	44.27b	474GB
UK-2014 [45]	747.8m	47.61b	532GB
Clueweb [47]	978.4m	42.57b	476GB
Hyperlink [48]	3.56b	128.7b	1.5TB
Sogou	271.9b	12,254.0b	136.9TB
Kronecker1 [49]	17.2b	274.9b	-
RMat [50]	17.2b	274.9b	-
Erdős [43]	17.2b	274.9b	-
Kronecker2 [49]	4,398.0b	70,368.7b	-

Previous work at large scale [30], [31] was designed and evaluated with only synthetic graphs. Processing extremely large-scale real-world graphs poses multiple challenges that we had to overcome in our work. First, the sheer size of the graph requires a highly scalable processing system. Second, the Sogou graph is highly irregular and differs significantly from synthetic graphs. Third, reading the whole graph from disk requires a fast and stable I/O orchestration. We briefly analyze these key challenges in the following.

Extremely large scale: Sogou's graph has 271,868,983,613 vertices and 12,253,957,290,295 edges. Designing an in-memory graph processing framework on such large scale requires all of our key innovations. In addition, to load such a graph into memory creates a significant challenge by itself. It requires a large amount of I/O resources, putting an unprecedented pressure on the distributed file system of Sunway TaihuLight to load the 136.9 TBs of input data.

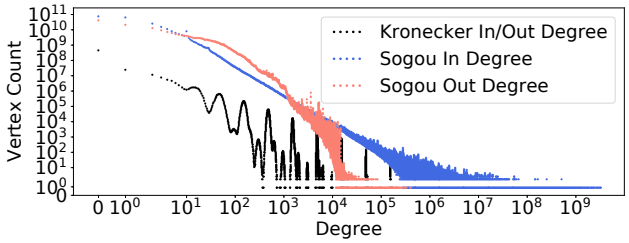

Fig. 3. In-degree and out-degree vertices comparing a Kronecker graph with the Sogou graph.

Asymmetric distribution of in- and out-degree and skewed in-degree distribution: Like many real-world graphs, but unlike most synthetic graphs, Sogou's graph has very different out-degree and in-degree distributions. The out-degree distribution of the vertices is relatively narrow and balanced while the in-degree distribution is highly skewed, as shown in Figure 3. For example, one of the largest in-degree vertices is http://www.ganji.com/, which has more than 2.7 billion in-edges, while the vertex with the largest out-degree has five orders of magnitude fewer out-edges (381 thousand). Furthermore, 90% of the vertices have an in-degree of under 15, while 90% have an out-degree of under 104.

Small number of in-edge magnets: Another feature of the Sogou graph (and web graphs in general) not represented by Graph500 synthetic graphs is that a small number of vertices are targeted by a large fraction of the edges. For example, the 16,222 ($6 \times 10^{-6}\%$ of total) vertices with an in-degree higher than 2^{25} account for 1,868,291,370,018 (15.25% of total) in-edges. Such asymmetry, imbalance, and in-edge magnets observed in real-world graphs pose significant challenges in processing load and space consumption, *motivating* the special handling of hin and hout vertices as proposed in Section VI-A.

B. Applications and algorithms

We demonstrate four important fundamental graph analytics applications: PageRank, BFS (Breadth-First Search), WCC (Weakly Connected Components), BC (Betweenness Centrality) and K-Core on the full scale of Sunway TaihuLight. In our main result, we test a real application that was, according to Sogou, highly desired but impossible before: we compute the fine-grained TrustRank [51] to differentiate trustworthy and spam webpages. TrustRank first selects a small set of manually identified *seed pages*, assigning them the trust score of 1, while all other vertices are given the initial score of 0. TrustRank then runs an algorithm similar to PageRank to convergence. The result of TrustRank is compared with the result of PageRank, to identify potential manipulation of page ranking using methods such as link spam. Section IX validates the results by comparing known non-spam and spam webpages with respect to their position in TrustRank and PageRank.

VIII. PERFORMANCE RESULTS

We first present benchmark results showing the impact of ShenTu optimizations, followed by scalability results to the whole TaihuLight system. We start with the smaller-scale real-world graphs widely used in evaluating graph frameworks

(shown in Table IV). We provide these results and the percentage of peak bandwidth (computed as outlined in Section VII) for each graph to enable comparison with other small-scale systems (analysis omitted due to space limit). These results demonstrate that the achieved performance heavily depends on the graph structure and algorithm. ShenTu delivers reasonable performance across the board.

Meanwhile, our key result is PageRank computation on the 12-trillion-edge Sogou graph in 8.5 seconds per iteration. Such performance demonstrates the capability of a cutting-edge supercomputer for whole-graph processing, potentially enabling complex analytics not affordable today.

TABLE IV
PERFORMANCE RESULTS (I/O IN GB/S AND GPEPS, WITH % OF PEAK IN BRACKETS (CF. SECTION VII) FOR PR). MISSING ENTRIES WERE OMITTED DUE TO BUDGETARY RESTRICTIONS AT FULL SCALE.

Graph (Nodes)	IO	PR	BFS	WCC	BC	K-Core
Twitter (16)	0.6	0.7 (28)	0.4	0.5	0.3	0.008
UK-2007 (64)	1.4	4.0 (28)	3.4	4.0	0.9	0.125
Weibo (100)	3.9	8.2 (41)	2.6	7.1	4.8	0.718
UK-2014 (100)	4.0	4.7 (47)	2.1	3.8	0.4	0.318
Clueweb (100)	4.0	5.0 (47)	2.5	4.8	11.2	0.407
Hyperlink (256)	5.9	17.2 (60)	3.7	14.5	0.3	0.264
Sogou (38656)	64.3	1443 (19)	30.8	224.4	-	-
Kro.1 (1024)	-	72.8 (40)	10.4	62.3	43.1	0.82
RMat (1024)	-	82.9 (29)	34.5	78.1	44.7	0.25
Erdős (1024)	-	52.5 (63)	47.9	51.3	40.9	18.6
Kro.2 (38656)	-	1969 (40)	774	1956	-	-

A. Performance of on-chip sorting

ShenTu heavily relies on efficient on-chip sorting. To this end, we implemented multiple algorithms and compare three variants here: (1) a simple implementation on the MPE, (2) an implementation using spin-locks on all CPEs, and (3) our optimized CPE routing implementation described in Section VI-A. Figure 4 compares the three implementations.

The simple MPE implementation achieves only 1.09 GB/s due to the limited memory bandwidth of the single MPE in each CG. The second implementation, called *CPE-atomic*, uses all the 64 CPEs for parallel sort, and synchronizes through an atomic-increment instruction. Although the CPE cluster itself can fully utilize its memory bandwidth, the overall sorting bandwidth is only 0.42 GB/s, due to the DRAM contention for locks and the random access pattern. Our optimized *CPE-network* implementation, on the other hand, performs on-chip sorting via register communication, eliminates explicit synchronization among CPEs, and allows memory access coalescing. As a result, the CPE cluster achieves a 8.96× performance than the baseline MPE-only implementation.

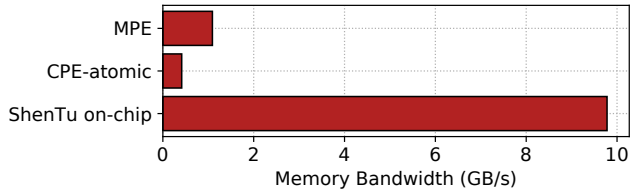

Fig. 4. Memory bandwidth when sorting with the three designs: MPE only, CPE-atomic, and ShenTu's on-chip sorting

B. Load balancing

The load imbalance of graph computation comes from the skewed degree distribution of the input graph and materializes in two aspects: (1) storage and (2) communication and processing. For example, the number of edges to be stored at each node will be highly imbalanced if we evenly partition the vertices in the Sogou graph. On 38,656 nodes, the average number of in- and out-edges is approximately 617 million per node. however, one node would have 36 billion edges, producing a 58× load. Without more intelligent load balancing, this would both exhaust the single-node memory and fail the graph assembly. Our optimizations also greatly reduce the communication volume. For example, after applying ShenTu's degree-aware messaging, the number of edges crossing partitions is reduced by 40% from 12.3 trillion to 4.8 trillion. Furthermore, after separating out hin and hout edges, the remaining regular edges are balanced among all nodes: the heaviest node has only 25.7% more edges than average.

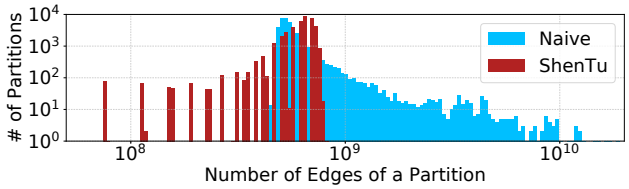

Fig. 5. Distribution of numbers of outgoing plus incoming edges per partition comparing naive vertex-balanced and ShenTu's partitioning.

Figure 5 shows a histogram of edge count per partition, which reflects the communication volume per node. Note that the y axis is in log scale. ShenTu's partitioning does leave a small fraction of nodes with significantly fewer outgoing edges, as its optimization targets high-degree vertices. However, it eliminates the heavily overloaded "tail", generating a much higher impact on parallel processing efficiency.

C. Selecting high-degree thresholds

Next, we discuss the configuration of our degree-aware messaging scheme, by selecting the two threshold parameters, K_{in} and K_{out}, used to identify the hin and hout vertices, respectively. Intuitively, if this bar is set too low, too many vertices will qualify, diminishing the benefits of high-degree specific optimizations and wasting the rather heavy-weight collective operations. If it is too high, we cannot achieve desired balancing results.

We empirically evaluate different combinations of the two parameters on a Kronecker graph of scale 34 on 4,096 nodes. Table V summarizes the results. We see that growing both K_{in} and K_{out} improves performance but quickly saturates when both exceeds 2048. This is due to the characteristic of the Kronecker scale-34 generated graph. About 90% of the vertices' in-degree and out-degree are lower than 5, thus the graph is dominated by low-degree vertices.

We also observe that performance is relatively robust with respect to small changes in K_{in} or K_{out}. Thus, we use a simple threshold-selection heuristic: To determine K_{in}, each node counts the number of vertices whose in-degree is larger

TABLE V
ITERATION TIMES (S) FOR PAGERANK ON KRONECKER GRAPH A ($|V| = 2^{34}$, $|E| = 2^{38}$) WITH 4,096 NODES COMPARING K_{in} AND K_{out}.

K_{out} \ K_{in}	0	1024	2048	4096	8192
0	6.132	4.713	4.619	4.081	4.069
1024	3.779	2.703	1.966	2.023	2.334
2048	3.910	2.020	1.921	1.957	2.221
4096	4.099	1.984	1.934	1.952	2.306
8192	4.468	2.038	1.942	2.016	2.308

than the total number of nodes as its local hin vertex count, and K_{in} is the median of all nodes' local hin vertex count. K_{out} is determined in the same way. Our dynamic threshold selection algorithm selects $K_{in} = K_{out} = 1,722$ for this case, the performance deviation of which is within 3% of the optimal selection.

D. Comparison with specialized Graph500 performance

We now proceed to compare ShenTu's performance to a highly-tuned implementation of the Graph500 benchmark on the same system [30]. This implementation takes advantage of BFS-specific properties to apply much more aggressive optimizations to secure the second spot on the Nov. 2017 list. For example, it removes isolated vertices and changes the vertex numbering to follow the degree distribution. ShenTu would need to translate user to internal labels, which would lead to unacceptable memory overheads.

At scale-38 on 9,984 nodes, the optimized Graph500 implementation traverses only 298.57 billion of the graph's 4,398.02 billion edges, obtaining 6,664.47 GTEPS and 294.73 GPEPS. With its degree-aware optimization, ShenTu processes around 70% fewer edges as Graph500 does, resulting in 577.85 GTEPS. When considering the actual traversed edges per second, ShenTu achieves 180.6 GPEPS, only 38.7% slower than the specialized Graph500 implementation.

To enable a fair comparison, we disable direction optimizations in both the Graph500 benchmark and ShenTu. Processing a scale-38 Kronecker graph on 9,984 nodes, Graph500 achieves 69.01 GPEPS while ShenTu achieves 88.02 GPEPS, demonstrating that ShenTu can even exceed the performance of a high-performance BFS implementation.

E. Scaling to the full system

We examine both weak and strong scalability of ShenTu using our four test applications. Because the full Sogou graph needs at least 10,000 nodes, we performed strong scaling tests on a synthetic scale-36 Kronecker graph. Figure 6 shows

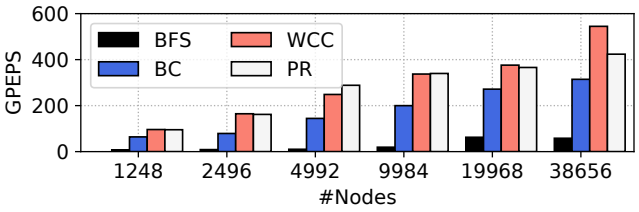

Fig. 6. Strong scaling results with a Scale-36 Kronecker graph.

the performance of three key applications using a growing number of nodes, relative to their respective 2500-node baseline. We note that the graph cut grows superlinearly when doubling the input scale. While ShenTu can process the fixed-size graph at full machine scale, the performance is limited by the increasing volume of the graph cut and the limited bisection bandwidth due to static routing in InfiniBand [52]. Our microbenchmarks validate these conjectures.

For weak scalability, we use Kronecker graphs with various scales ranging from 36 on 1,248 nodes to 40 on 38,656 nodes shown in Figure 7. ShenTu shows a good weak scalability

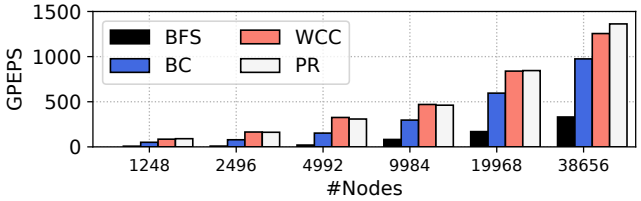

Fig. 7. Weak scaling results for Kronecker graphs at various scales.

for all algorithms to the full machine scale. The flattening of the scalability at the last step is also explained by the limited full-scale bisection bandwidth due to static routing.

The weak scalability test shows that ShenTu can run on full scale on a heterogeneous extremely large scale super computer. We also ran BFS and PageRank on the entire system, which is Scale-42 on 38,656 nodes. It costs 6.5s and 25.3s to finish one iteration of PageRank and BFS, achieving 1,984.8 GPEPS and 809.9 GPEPS, respectively.

F. Analyzing TrustRank of the full Sogou graph

Thanks to ShenTu's unprecedented processing capacity, we are able to run a variety of graph analytics algorithms at a scale not attempted before. As one use case, we compare TrustRank and PageRank in differentiating between non-spam and spam pages in the Sogou graph. We were able to run both algorithms to convergence,[4] requiring 47 iterations in 4 minutes for TrustRank, and 53 iterations in 8 minutes for PageRank. To evaluate their effectiveness, we divide the vertices (webpages) into 20 buckets according to their PageRank values, using a rank-based partitioning method [51]. It sorts vertices by their PageRank and divide them into buckets with (nearly) uniform PageRank sums. We reuse such PageRank bucket size distribution for TrustRank, to obtain 20 buckets with the same sizes, but with pages sorted by TrustRank. We then randomly sampled 100 webpages from each bucket (of both sets of 20) and requested manual assessment by Sogou engineers. The result is a per-page quality score, assigned considering factors such as link validity and presence of spam content.

Figure 8 visualizes the quality score distribution of both algorithms, showing that TrustRank gives more quality-relevant results than PageRank. Buckets with descending TrustRank values possess a more consistent trend of declining quality, with more pages "sinking" to the lower quality bands. Its difference with PageRank is especially evident in the middle buckets (9-14), where the PageRank buckets admit a larger fraction of low-quality pages.

[4]We stopped when the sum of rank differences between iterations was under 10^{-6}.

This is the first experiment comparing results of *exact* PageRank and TrustRank computations on a real-world Internet graph with billions of webpages and trillions of links. We gathered terabytes of output data which are invaluable for future analysis by network scientists.

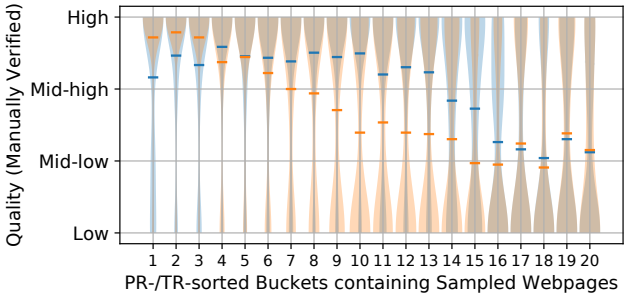

Fig. 8. The quality score distribution of sampled webpages grouped by corresponding buckets given by TrustRank (blue) and PageRank (orange) values. The average scores of each bucket are marked by blue and orange horizontal lines respectively.

IX. Implications

The implications of our work are manyfold. First, we show a hero-scale computation of the Sogou web graph, which was previously impossible in reasonable time and cost. Second, and even more powerful, we demonstrate a graph framework that can execute arbitrary algorithms at full scale of the fastest supercomputer in operation today. This will have a huge impact on the many scientific fields, such as genomics, connectomics, or social sciences, where graph processing plays a major role. Third, we establish principles for transforming irregular computations to hierarchical streaming regular computations on heterogeneous compute architectures. These techniques not only act as blueprint for future graph processing frameworks on Peta- and Exascale machine but also may influence the design of other irregular processing approaches such as sparse linear algebra or unstructured or adaptive grid computations.

The state of the art in large web graph processing uses rather inefficient methods, either MapReduce [53] or specialized out-of-core systems, both reading the 100 TBs of graph data for each iteration leading to excessive time and energy requirements. Thus, researchers and industry usually resort to simplified graphs or other approximation methods [54]. For example, in the production environment of Sogou, PageRank is currently computed for a reduced graph, merging pages by URL prefixes. This assumes all webpages under the same upper level URL share the same PageRank score, which sacrifices resolution and thus quality of research insights and search results. It also opens avenues for spammers to place pages under useful pages, for example in Internet boards.

Processing even the *reduced* graph takes more than 60 hours on 100 Hadoop server machines at Sogou. ShenTu runs PageRank on Sogou's *full* web graph to convergence in minutes on Sunway TaihuLight, more than three orders of magnitude faster than the state of the art for the reduced graph! Even on a per-server basis, ShenTu achieves more than an order of magnitude speedup. This allows us to compute additional features, such as distance, in order to counteract the deceiving efforts of search engine optimization (SEO), a billion dollar industry [55]. Detecting and ignoring such ranking manipulation is one of the key issues in providing high quality search results and can even be used to optimize web crawling itself. The intuitive idea of such anti-SEO algorithm is to use page rank to find certain high-profile webpages (seeds) and then calculate the distances of other webpages to these seeds. The pages which are too far away from these seeds are considered less important. The rationale here is that the SEO industry can boost PageRank of webpages by adding links from its controlled webpages, but cannot boost them high enough to become seeds.

In a broader view, ShenTu gives a concrete example on using the large memories and extreme parallelism of supercomputers to solve real-world data analytics problems on the ever increasing size of data. The scalable key innovations proposed and discussed in this paper, such as supernode routing, degree-aware messaging, hardware specialization, and on-chip sorting can easily be generalized to other supercomputers. Supernode routing can be generalized to a general routing scheme for hierarchical network topologies where islands of higher bandwidth send messages to other such islands to be distributed inside the high-bandwidth domains. Our degree-aware messaging using HPC techniques such as collective operations that can be tuned to the specific network architecture shows how those fundamental algorithms can be applied, even for irregular and unbalanced problems. In fact, our load balancing strategy that combines this with randomized vertex assignment and efficient storage enables handling very large real-world graphs in any parallel computing setting. Hardware specialization is a necessity to achieve lowest energy consumption in future systems. While we can only guess the specifics of those systems, we believe that the division of communication and processing functions will remain. Furthermore, the on-chip sorting shows how TaihuLight's innovative register communication can be exploited in irregular applications. This will not only influence designers of such applications but also hardware manufacturers to introduce advanced communication features into their many-core chips. ShenTu is first to integrate these and many more innovations into a scalable high-performance system that maps irregular and unbalanced graph computations to a heterogeneous Petascale system. With it, we hope to fuel a new trend to use HPC systems not only for traditional scientific computing workloads but also for emerging data analytics.

Acknowledgment

We thank Rujun Sun, Ziyu Hao from SKL-MEAC, Huanqi Cao, Miao Wang, Wentao Han, Xu Ji, YuanChao Xu, Guangwen Yang from Tsinghua, Xiyang Wang, Zhao Liu, Bin Yang, Wanliang Li from NSCC-Wuxi, and Yi Li from Beijing Sogou. This work was partially supported by National Key Research & Development Plan of China under grant #2017YFA0604500 and #2016YFA0602100, NSF of China under grant #61525202 and #91530323. The corresponding authors are Wenguang Chen and Wei Xue.

REFERENCES

[1] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler, "Genbank," *Nucleic acids research*, 2005.

[2] H. Mustafa, I. Schilken, M. Karasikov, C. Eickhoff, G. Ratsch, and A. Kahles, "Dynamic compression schemes for graph coloring," *bioRxiv*, 2018.

[3] B. Pakkenberg and H. Gundersen, "Total number of neurons and glial cells in human brain nuclei estimated by the disector and the fractionator," *Journal of microscopy*, 1988.

[4] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, "Challenges in parallel graph processing," *Parallel Processing Letters*, 2007.

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *SIGCOMM*, ACM, 1999.

[6] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *ACM SIGPLAN Notices*, ACM, 2013.

[7] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *OSDI*, USENIX, 2016.

[8] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and information systems*, 2008.

[9] 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*, ACM, 2010.

[10] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: Graph processing at facebook-scale," *VLDB*, 2015.

[11] M. Wu, F. Yang, J. Xue, W. Xiao, Y. Miao, L. Wei, H. Lin, Y. Dai, and L. Zhou, "Gram: scaling graph computation to the trillions," in *SoCC*, ACM, 2015.

[12] Harshvardhan, A. Fidel, N. M. Amato, and L. Rauchwerger, "An algorithmic approach to communication reduction in parallel graph algorithms," in *PACT*, IEEE, 2015.

[13] A. Roy, L. Bindschaedler, J. Malicevic, and W. Zwaenepoel, "Chaos: Scale-out graph processing from secondary storage," in *SOSP*, 2015.

[14] P. Kumar and H. H. Huang, "G-store: high-performance graph store for trillion-edge processing," in *SC*, IEEE Press, 2016.

[15] H. Liu and H. H. Huang, "Graphene: Fine-grained io management for graph computing," in *FAST*, USENIX Association, 2017.

[16] S. Maass, C. Min, S. Kashyap, W. Kang, M. Kumar, and T. Kim, "Mosaic: Processing a trillion-edge graph on a single machine," in *Proceedings of the Twelfth European Conference on Computer Systems*, pp. 527–543, ACM, 2017.

[17] D. Nguyen, A. Lenharth, and K. Pingali, "A lightweight infrastructure for graph analytics," in *SOSP*, SOSP, ACM, 2013.

[18] K. Zhang, R. Chen, and H. Chen, "Numa-aware graph-structured analytics," in *ACM SIGPLAN Notices*, ACM, 2015.

[19] 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," *VLDB*, 2015.

[20] A. Kyrola, G. E. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc.," in *OSDI*, 2012.

[21] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: edge-centric graph processing using streaming partitions," in *SOSP*, ACM, 2013.

[22] D. Zheng, D. Mhembere, R. Burns, J. Vogelstein, C. E. Priebe, and A. S. Szalay, "Flashgraph: Processing billion-node graphs on an array of commodity ssds," in *FAST*, 2015.

[23] X. Zhu, W. Han, and W. Chen, "Gridgraph: Large scale graph processing on a single machine using 2-level hierarchical partitioning," in *USENIX ATC*, 2015.

[24] 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," *VLDB*, 2012.

[25] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs.," in *OSDI*, 2012.

[26] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," in *Proceedings of the Tenth European Conference on Computer Systems*, ACM, 2015.

[27] S. Hong, S. Depner, T. Manhardt, J. Van Der Lugt, M. Verstraaten, and H. Chafi, "Pgx.d: A fast distributed graph processing engine," in *SC*, ACM, 2015.

[28] D. Gregor and A. Lumsdaine, "The parallel bgl: A generic library for distributed graph computations," *POOSC*, 2005.

[29] F. Checconi and F. Petrini, "Traversing trillions of edges in real time: Graph exploration on large-scale parallel machines," *IPDPS*, 2014.

[30] H. Lin, X. Tang, B. Yu, Y. Zhuo, W. Chen, J. Zhai, W. Yin, and W. Zheng, "Scalable graph traversal on sunway taihulight with ten million cores," in *IPDPS*, IEEE, 2017.

[31] K. Ueno, T. Suzumura, N. Maruyama, K. Fujisawa, and S. Matsuoka, "Extreme scale breadth-first search on supercomputers," in *Big Data*, IEEE, 2016.

[32] C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadler, T. Warburton, and L. Wilcox, "Extreme-scale amr," in *SC*, IEEE, 2010.

[33] M. Bernaschi, M. Bisson, T. Endo, S. Matsuoka, and M. Fatica, "Petaflop biofluidics simulations on a two million-core system," in *SC*, 2011.

[34] J. Chhugani, C. Kim, H. Shukla, J. Park, P. Dubey, J. Shalf, and H. D. Simon, "Billion-particle simd-friendly two-point correlation on large-scale hpc cluster systems," in *SC*, IEEE Computer Society Press, 2012.

[35] T. Muranushi, H. Hotta, J. Makino, S. Nishizawa, H. Tomita, K. Nitadori, M. Iwasawa, N. Hosono, Y. Maruyama, H. Inoue, H. Yashiro, and Y. Nakamura, "Simulations of below-ground dynamics of fungi: 1.184 pflops attained by automated generation and autotuning of temporal blocking codes," in *SC*, 2016.

[36] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang, W. Ge, J. Zhang, Y. Wang, C. Zhou, and G. Yang, "The Sunway TaihuLight supercomputer: system and applications," *Science China Information Sciences*, vol. 072001, 2016.

[37] W. Zhang, J. Lin, W. Xu, H. Fu, and G. Yang, "Scstore: managing scientific computing packages for hybrid system with containers," *Tsinghua Science and Technology*, vol. 22, no. 6, pp. 675–681, 2017.

[38] 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 '14*, 2014.

[39] S. Beamer, K. Asanovic, and D. Patterson, "Searching for a Parent Instead of Fighting Over Children: A Fast Breadth-First Search Implementation for Graph500," *Tech Report UCB/EECS-2011-117*, 2011.

[40] 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'17*, ACM, 2017.

[41] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM SISC*, 1998.

[42] S. Beamer, A. Buluc, K. Asanovic, and D. Patterson, "Distributed memory breadth-first search revisited: Enabling bottom-up search," *IPDPSW*, 2013.

[43] P. Erdős and A. Rényi, "On the existence of a factor of degree one of a connected random graph," *Acta Mathematica Hungarica*, 2005.

[44] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?," in *WWW*, ACM, 2010.

[45] P. Boldi and S. Vigna, "The webgraph framework i: compression techniques," in *WWW*, ACM, 2004.

[46] W. Han, X. Zhu, Z. Zhu, W. Chen, W. Zheng, and J. Lu, "Weibo, and a tale of two worlds," in *ASONAM 2015*, ACM, 2015.

[47] The lemur project: Clueweb12 web graph., "http://www. lemurproject.org/clueweb12/webgraph.php/."

[48] WDC - Hyperlink Graphs, "http://webdatacommons.org/hyperlinkgraph/," 2018.

[49] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," in *ECML-PKDD'05*, Springer, 2005.

[50] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *SIAM DM'04*, SIAM, 2004.

[51] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen, "Combating web spam with trustrank," in *VLDB*, VLDB Endowment, 2004.

[52] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks," in *Cluster'08*, IEEE, Oct. 2008.

[53] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, 2008.

[54] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova, "Monte carlo methods in pagerank computation: When one iteration is sufficient," *SIAM NA*, vol. 45, no. 2, pp. 890–904, 2007.

[55] Search engine optimization marketing spending, "https://www.statista.com/statistics/269410/advertising-expenditure-for-seo-marketing/," 2018.