Order or Shuffle: Empirically Evaluating Vertex Order Impact on Parallel Graph Computations

George M. Slota
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY
slotag@rpi.edu

Sivasankaran Rajamanickam
Scalable Algorithms Department
Sandia National Laboratories
Albuquerque, NM
srajama@sandia.gov

Kamesh Madduri
Computer Science and Engineering
The Pennsylvania State University
University Park, PA
madduri@cse.psu.edu

***Abstract*—The in-memory graph layout affects performance of distributed-memory graph computations. Graph layout could refer to partitioning or replication of vertex and edge arrays, selective replication of data structures that hold meta-data, and reordering vertex and edge identifiers. In this work, we consider one-dimensional graph layouts, where disjoint sets of vertices and their adjacencies are partitioned among processors. Using the PuLP graph partitioning method and a breadth-first search (BFS)-based vertex ordering strategy, we empirically evaluate the impact of this graph layout on a collection of five distributed-memory graph computations. Our evaluation considers several objective metrics in addition to execution time, and we observe a considerable performance improvement over randomization.**

1. Introduction

The in-memory organization of a graph can have a considerable impact on the performance of a parallel graph computation. The integer identifiers assigned to vertices (i.e., vertex ordering) and the distribution of vertices and edges to compute units (graph partitioning) both impact performance. There are three key considerations when ordering vertices and partitioning a graph: memory locality, computational load balance, and inter-node communication time.

Memory locality can be further characterized in terms of spatial and temporal locality. As an example of the impact of vertex ordering on spatial locality, consider performing the list ranking computation on a linked list with randomly permuted node identifiers versus performing list ranking on an ordered list. Only a very small fraction of the cache line fetched is used in the random list case, and the performance impact can be an order of magnitude on current systems. For the importance of temporal locality, consider accessing vertex attributes in an unsorted edge list versus a sorted edge list. In a parallel setting, with shared caches and NUMA domains, the performance impact of ordering on locality is further complicated and challenging to analyze theoretically, but ordering certainly cannot be ignored.

Partitioning a graph across P compute units reduces the number of vertices (and edges) that compute units have to process by a factor of P. Since partitioning tools typically perform a balanced vertex partitioning, the number of edges per partition might vary. The edges assigned to a partition include local edges (connecting vertices in the same partition) and edges with an end point in another partition. In bulk-synchronous parallel computations, the local edge count, the vertex count per partition, and the edge count per partition may all be used as proxies for computational load balance. Even in asynchronous graph computations with dynamic load balancing, the overhead due to load balancing can be reduced if we begin with a well-balanced distribution of vertices and edges. Reducing inter-processor communication time is considered one of the key motivations for graph partitioning. Partitioning tools indirectly attempt to reduce communication time by attempting to minimize the number of inter-partition edges and improving balance of inter-partition edges.

While some partitioning-ordering combinations complement each other, others do not. Since partitioning tools create balanced vertex partitions, and it is helpful to contiguously number vertices in each partition, it makes sense to apply ordering after partitioning. However, vertex ordering formulations are generally defined in a sequential setting (e.g., degree-based ordering considers vertex degrees in the entire graph). Partitioning after ordering may undo the effect of ordering. Thus, using a graph partitioning tool for vertex partitioning, followed by an unspecified vertex ordering in each partition seems to be a commonly-used strategy. Another common strategy is to shuffle/permute vertex identifiers at the start to destroy any locality, and then create fairly well-balanced partitions. On shared-memory platforms, vertex ordering is shown to have a big performance impact on some breadth-first search (BFS) algorithms [1], and partitioning is only required to handle NUMA issues. For certain computations (e.g., triangle listing), an ordering-aware algorithm can be asymptotically faster (in terms of sequential work complexity) than an ordering-agnostic approach [2], [3]. In such cases, a distributed-memory graph partitioning and subsequent parallelization must be ordering-aware. For the Graph500 benchmark, early implementations used just randomization and no explicit partitioning (e.g., [4]), but over time, algorithms/implementations are being designed in concert with partitioning and ordering schemes [1], [5], [6],

and this has resulted in orders-of-magnitude performance improvement over earlier work.

If one considers the algorithms for BFS, triangle listing, PageRank, and other popular graph analytics as representative Graph Algorithm Building Blocks (GABBs), the impact of partitioning and ordering on GABBs is clearly significant (as recent results have shown). One of the goals of this paper is to investigate whether we can design an analytic-agnostic partitioning+ordering combination that consistently improves performance for GABBs and analytics based on GABBs. Randomization is a reasonable baseline, as it is analytic-agnostic and it typically offers good computational load balance (at the expense of memory locality and inter-node communication). We recently developed PuLP [7], [8], a fast parallel graph partitioning tool. This paper investigates ordering schemes that are compatible with PuLP.

The main contribution in this work is an empirical evaluation (see Section 5) of the impact of partitioning and ordering on five distributed graph analytics (PageRank, approximate subgraph counting [9], BFS, single-source shortest paths, and SPARQL query processing). PageRank, BFS, and SSSP can be considered as GABBs, and the other two computations involve a combination of data structures, optimizations, and parallelization schemes. We develop a new lightweight BFS-based ordering scheme that is compatible with PuLP, and additionally evaluate alternatives (see Section 2 for details). We also separately evaluate partitioning and ordering impact using machine-independent and analytic-independent performance measures.

2. Partitioning and Ordering Methods

We consider several partitioning schemes in this work. These include the standard *single balance constraint* and *single optimization criterion* case (i.e., balancing vertices per part and minimizing edge cut); the multiple balance constraints and single optimization criterion case (balancing vertices and edges per part and minimizing edge cut); and the multiple balance constraints and multiple optimization criteria case (balancing vertices and edges per part and minimizing edge cut and the max per-part edge cut). Other work [10] has demonstrated the importance of this secondary objective of max per-part cut (also termed as max messages). We select these schemes because PuLP can support all of them, and other partitioners can support some of them.

As the paper title suggests, we use a random partitioning as the baseline, where parts are assigned randomly to each vertex. Random partitioning leads to a very high edge cut, with the percentage of edges cut scaling approximately as $\frac{p-1}{p}\%$, where p is the number of computed parts. However, random partitioning typically gives balanced parts with close to equal numbers of vertices and edges per-part when $p \ll n, m$ (n is the number of vertices in a given graph and m is the number of edges). Randomization is also used as the baseline in other distributed graph processing frameworks (e.g., PowerGraph [11], Giraph [12]). Ghost vertices, vertices within the one-hop neighborhood of a part, require their values to be stored and accessed for many iterative computations. The number of ghost vertices generally increases proportionally to the edge cut. Computation time may also be higher due to decreased intra-part locality and increased total memory required. Communication time is higher due to a higher edge cut, with updated ghost vertex data being passed along these cut edges.

The other partitioning method we evaluate is the well-known METIS [13] partitioner. We treat it as a representation of the state-of-the-art quality for optimizing edge cut as a single objective with both single and multiple constraints. Very recently, partitioning methods in the KaHIP library [14] have demonstrated improved quality relative to METIS for small-world graphs. However, these methods only satisfy single balance constraint scenarios, and the improvements relative to METIS are likely not large enough to affect the general observations we note in this current study. Including KaHIP in our evaluations would make for promising future work. In our results, we use METIS to refer to METIS optimizing for edge cut and constraining only vertices per part; METIS-M is METIS optimizing for edge cut and constraining both vertices and edges per part; PuLP-MM is PuLP optimizing for edge cut and max per-part cut and constraining both vertices and edge; we also run PuLP-M, which optimizes for edge cut and constrains vertices and edges per part, for direct comparison to METIS-M.

2.1. Ordering

Reverse Cuthill-McKee (RCM) [15], [16] is a commonly-used vertex ordering strategy in sparse matrix and graph applications to minimize the metric of graph bandwidth for regular matrices and graphs. However, for graphs with highly skewed degree distributions, the bandwidth is usually going to be large, on the order of d_{max}, where d_{max} is the maximal degree of any vertex in the graph. In real-world graphs exhibiting a large degree skew, we observe $d_{max} \gg d_{avg}$, where d_{avg} is the average vertex degree. Optimizing for bandwidth will therefore not necessarily create any global improvements in compaction and therefore not improve cache access efficiency for these vertices.

As such, we look at other metrics to give an indication of the possible cache utilization in practice. Across the entire adjacency array, one can measure how often edges listed in order also have identifiers within a single integer value of each other. This indicates that these edges would be neighboring nonzeros in the same row of an adjacency matrix. Co-located edges improve cache utilization of per-vertex information accesses, such as checking visitation status for BFS or PageRank value lookups. To quantify how many co-located vertex identifiers for the edges are in the adjacency list, we consider two values. First, we measure a ratio of how co-located all edges are, where a value of zero indicates that no edges are co-located and a value of one indicates that all edges are co-located. Second, we report a running "cost" as the sum of the distances, or gaps, between vertex identifiers in the adjacency list. We scale the distances

by their $\log$, as a distance close to one indicates that vertices are closely co-located and would have minimal cache cost for their subsequent accesses. The cost difference between large and very large distances is minimal, since it is likely a new cache line would need to be loaded in both instances. Finding an ordering that minimizes this sum is referred to in the literature as the *Minimum Logarithmic Gap Arrangement* problem, which is NP-hard [17]; this metric has been mostly used for the related problem of graph compression. The true dependence of cache utilization on distance would be architecture-specific, but the approximation of this cost gives enough insight for comparative purposes when examining ordering quality.

We develop a simple BFS-based ordering method (described in an extended technical report of this paper [18]) that, in practice, performs comparable to RCM in terms of the previously-mentioned metrics. We call this ordering method BFS in subsequent sections, and refer to the combination of PuLP and this ordering method as DGL (distributed graph layout). We compare the performance of BFS and RCM to a random ordering, where vertices are shuffled into a random order. We omit comparisons to a "natural" ordering (i.e., how the graph vertices are ordered in the original dataset) due to highly variable performance (we observed a spread of both $2\times$ speedup and slowdown for a natural ordering vs. random ordering on our PageRank test; similarly, we observed similar variance in vertex block vs. random partitioning), and because a natural ordering is not well-defined.

In addition to RCM and BFS ordering, orderings based on nested dissection [13], space-filling curves [19], and spectral bisection [20] have been used in prior work. Our future work might involve a more thorough investigation of these and similar approaches for graph analytical applications.

3. Parallel Graph Computations

In this section, we give an overview of the five distributed graph analytics used in our evaluation. These graph analytics were selected to represent a range of execution characteristics.

3.1. Distributed PageRank

Our distributed PageRank uses an iterative bulk synchronous approach. For parallelism, we use MPI, OpenMP, and a $\frac{|V|}{p}$ partitioning, with each of p MPI tasks calculating the scores for roughly the same number of vertices. With one MPI task per node, we then use thread parallelism when updating the counts of owned vertices. With the exception of the single MPI communication call on each iteration, all per-task work can be done in parallel. Updates are passed among neighbors using an MPI all-to-all exchange. In practice, we observe this specific implementation to be very efficient and scalable, giving per-iteration costs of less than a few seconds for networks of over 100 billion edges while running on 256 compute nodes. The specific technical details of the implementation are omitted, but please see [21] for a more in-depth discussion.

3.2. Subgraph Counting

Our distributed subgraph counting procedure is an implementation of the Alon et al. color-coding technique [22]. The approach is very computationally intensive, with work scaling as $O(m2^k)$ and memory scaling as $n\binom{k}{\frac{k}{2}}$, where n and m are the number of vertices and edges in the graph and k is the number of vertices in the subgraph. Communication requirements scale the same as memory requirements and are proportional to edge cut. Our approach uses several optimizations, including fully partitioning and compressing the memory-intensive dynamic programming table to decrease memory requirements across all tasks, further compressing the table during communication to reduce the total transfer volume, and using all-to-all exchanges in lieu of broadcasts to reduce communication times. These optimizations demonstrate good scaling and enable us to count subgraphs of 10 and 11 vertices on billion-edge networks in minutes on a modest number of 16 nodes. Please refer to [9] for an in-depth discussion.

3.3. SSSP and BFS

We also assess the performance impact of layout on tuned implementations for parallel breadth-first search (BFS) and single-source shortest paths (SSSP) computation in this paper. Out of all the analytics we consider, BFS and SSSP are the least computationally intensive. Their communication requirements are also relatively small. Our parallel BFS approach can take advantage of both 1D and 2D graph distributions [4], [23], [24]. We use a 1D distribution in this work, as it is easier to correlate communication time with edge cut after partitioning with a 1D distribution. We use an optimized parallel implementation [25] of the Δ-stepping algorithm [26] for parallel SSSP in this paper. For an overview of the state-of-the-art in performance optimizations for these routines, we refer the reader to [1], [5], [27].

3.4. Distributed RDF Stores and SPARQL Query Processing

Resource Description Framework (RDF) is a popular data format for storing web data sets. Informally, the RDF format specifies typed relationships between entities, and the basic record in an RDF data set is a *triple*. We use a distributed MPI-based implementation of an open-source triple store called RDF-3X [28], called RDF3X-MPI [29]. An alternate approach to viewing an RDF data set is as a directed graph with edge types. To eliminate the need for communication when answering a query on the data store, this graph is partitioned with n-hop replication occurring for each part. Ideally, the graph would be partitioned such that the number of triples replicated between tasks is minimized. For this application, we study the impact of partitioning on the number of replicated triples. A smaller value of replication is desired in terms of memory usage, and smaller index sizes should further translate to faster query times.

4. Experimental Setup

We evaluate the effects of partitioning and ordering objective on the graph analytics workload using a collection of nine large-scale low diameter graphs, listed in Table 1. LiveJournal, Orkut, and Twitter (follower network) are crawls of online social networks obtained from the SNAP Database and the Max Planck Institute for Software Systems [30], [31]. uk-2005 and sk-2005 are crawls of the United Kingdom (.uk) and Slovakian (.sk) domains performed in 2005 using UbiCrawler and downloaded from the University of Florida Sparse Matrix Collection [32]–[34]. WebBase is similarly a crawl obtained in 2001 by the Stanford WebBase crawler. We created the BSBM and LUBM graphs from RDF data sets generated using the Berlin SPARQL benchmark [35] and Lehigh University Benchmark [36] generators. DBpedia was created from RDF triples extracted from Wikipedia [37].

The Orkut graph is undirected and the remaining graphs are directed. For the web and social graphs, we preprocessed the graphs before executing PageRank, BFS, SSSP, and subgraph counting. Specifically, we removed all degree-0 vertices, multi-edges, and extracted the largest (weakly) connected component. Further, edge directivity was ignored when partitioning the graphs using PuLP and METIS and reordering with RCM and BFS. Table 1 lists the sizes of these nine graphs after preprocessing.

TABLE 1. Test graph characteristics *after* preprocessing. Graphs belong to three categories, OSN: Online social networks, WWW: Web crawl, RDF: graphs constructed from RDF data. # Vertices (n), # Edges (m), average (d_{avg}) and max (d_{max}) vertex degrees, and approximate diameter ($\widetilde{D}$) are listed. $B = \times 10^9$, $M = \times 10^6$, $K = \times 10^3$.

Network	Category	n	m	d_{avg}	d_{max}	$\widetilde{D}$	Source
LiveJournal	OSN	4.8 M	42 M	18	39 K	21	[38]
Orkut	OSN	3.1 M	117 M	76	33 K	9	[39]
Twitter	OSN	44 M	2.0 B	37	750 K	36	[31]
uk-2005	WWW	39 M	781 M	40	1.8 M	21	[32]
WebBase	WWW	113 M	844 M	15	816 K	376	[32]
sk-2005	WWW	44 M	1.6 B	73	15 M	308	[32]
BSBM	RDF	16 M	67 M	8.6	3.6 M	7	[35]
LUBM	RDF	33 M	133 M	8.1	11 M	6	[36]
DBpedia	RDF	62 M	190 M	6.1	7.3 M	7	[37]

The scalability studies were primarily done on *Blue Waters*, a large petascale supercomputer at the National Center for Supercomputing Applications (NCSA). Each XE compute node of *Blue Waters* is a dual-socket system with 64 GB main memory and AMD 6276 Interlagos processors at 2.3 GHz. The system uses a Cray Gemini 3D torus interconnect. We built our programs with the GNU C++ compiler (version 4.8.2), using OpenMP for multithreading and O3 optimization. For the pre-processing phases of DGL (partitioning and reordering) and some scalability runs, we utilized *Compton*, a testbed cluster. *Compton* has a dual socket setup with Intel Xeon E5-2670 (Sandy Bridge) CPUs at 2.60 GHz and 64 GB main memory.

5. Results and Discussion

We partition all graphs for 16 and 64 parts using our different methods (random, METIS, METIS-M, PuLP-M, PuLP-MM) and run our analytics using these computed partitions. To evaluate partition quality directly, we report the time spent in the communication phase of each analytic as the speedup relative to a random partitioning. We report results with randomized ordering during the partitioning experiments for the sake of consistency. We additionally order all computed partitions using our methods (random, BFS, RCM). We evaluate ordering quality by the speedup of the computational phase of each analytic relative to a random ordering. With a random partitioning, improvements in order quality is difficult to achieve due to how disconnected each part is. Therefore, for the sake of consistency during ordering evaluation, we report our comparisons using the PuLP-MM partitions. We don't report timing comparison between partitioning and ordering strategies or discuss application-specific amortization of partitioning and ordering costs, as similar results have been presented and discussed in prior work [7], [8], [18].

5.1. Partitioning and Ordering Evaluation

The partitioning performance in terms of both vertex and edge balance constraints and edge cut and maximal per-part edge cut objectives for the different partitioners is shown in Table 2 as geometric averages. We also note that aggregate measures don't fully capture the wide spread of results among different tests, so include *min* and *max* improvements for edge cut and max per-part cut as well. For instance, the improvement METIS has relative to random partitioning varies from 1.5× for 64-way partitioning of Twitter to 107× improvement for 16-way partitioning of uk-2005. We note that, as expected, single-constraint METIS optimizing for the only the single edge cut objective demonstrates the best improvement for edge cut (EC). It however shows weaker improvements in the maximum per-part edge cut (EC_{max}) and edge balance (E_{max}) relative to the other methods. The PuLP-MM method shows the best improvement in terms of the EC_{max} metric while satisfying both vertex and edge constraints. The PuLP-M and METIS-M methods offer middle-of-the-road performance in terms of EC and EC_{max} while also achieving both balance constraints. Our goal is to experimentally evaluate how these differences and tradeoffs in optimization criteria and part balance impact our selected graph analytic workloads.

We also compare the ordering performance in Table 3. Table 3 gives both the co-location ratio (Co-loc. Ratio) as well as the log sum of gap distances ratios (Gap Sum Ratio) for all ordering combinations across all graphs for 16 and 64 parts. In an attempt to give a graph-independent ratio for the Gap Sum Ratio, we scale the log sum by a worst-case possible value of $m \log n$. We report the geometric mean values across all five partitioning strategies (Random, METIS, METIS-M, PuLP-M, PuLP-MM). For co-location ratio, higher indicates better locality, while for the log gap sum ratio, lower indicates better locality. We omit reporting

TABLE 2. AVERAGE PARTITIONING CHARACTERISTICS ACROSS ALL GRAPHS. GEOMETRIC MEAN OF VERTEX BALANCE V_{max}, EDGE BALANCE E_{max}, IMPROVEMENT OVER RANDOM PARTITIONING FOR EDGE CUT RATIO EC AND MAX PER-PART EDGE CUT EC_{max} ARE SHOWN. THE BEST VALUES FOR EACH COLUMN ARE IN BOLD FONT.

Partitioner	V_{max}	E_{max}	EC (imp) avg	min	max	EC_{max} (imp) avg	min	max
Random	1.15	1.7	1	1	1	1	1	1
METIS	1.1	3.88	**7.71**	**1.5**	**107**	2.39	0.25	63
METIS-M	1.1	1.5	4.4	1.02	41	2.16	0.77	22
PuLP-M	1.1	1.5	5.5	1.17	64	2.1	0.54	23
PuLP-MM	**1.1**	**1.5**	5	1.19	63	**3.18**	**2.54**	**204**

the co-location ratio for Random ordering in Table 3, as all values are close to zero, a few orders of magnitudes less than RCM and BFS. We observe nearly that the BFS-based BFS ordering results in the best co-location ratio and lowest log gap sum ratio across almost all instances. The computational timings results we'll report next in our benchmarks will demonstrate that optimizing for this objective translates into real performance benefits across a wide range of graph analytics.

5.2. PageRank Performance

We first examine the effect of partitioning and ordering on our distributed PageRank implementation. For these experiments, we use the three social network graphs (LiveJournal, Orkut, and Twitter) as well as the three web crawls (uk-2005, WebBase, sk-2005). Figure 1 (top) gives the speedups relative to random partitioning for METIS and METIS-M as well as PuLP-M and PuLP-MM. Figure 1 (bottom) gives the speedups relative to random ordering for BFS and RCM. Table 4 gives the explicit speedup values and overall geometric means across the six test graphs. These value are for 20 iterations of PageRank executing on 16 nodes of *Blue Waters*.

We observe that all partitionings offer considerable speedups relative to random. In general, the web crawls show even greater speedups than the social networks. This is due to the web crawls being greater in diameter and more separable than social networks, resulting in a decrease in the number of cut edges and subsequently greater performance improvements relative to random partitioning. Averaged across all six test graphs, the multi-objective and multi-constraint PuLP-MM offers the greatest mean speedup as shown in Table 4. The performance benefit can be explained due to the implementation's use of an iterative bulk synchronous model and moderate communication requirements, so the improved communication balance resulting from PuLP-MM's decrease in max per-part cut becomes apparent in the timings.

Additionally, we note that both RCM and BFS offer considerable speedups for total computation times relative to random ordering. On the uk-2005 and WebBase graphs, the computational speedups for BFS are about 5×. Again we observe that the social networks generally show less performance benefit relative to random, and this is again due to their lower diameter and greater small-world characteristics, which makes effective ordering more difficult to achieve. However, we still observe consistent 20%-40% speedups with the improved orderings. Overall, BFS gives a greater performance speedup over RCM by about 10%, a result we explain based on our measurement of potential locality and cache performance as demonstrated in Table 3.

To offer visual explanation of the performance of balanced constraint partitioning on total execution time, we give execution timelines in Figure 2 of a single run of 10 iterations of PageRank on the WebBase graph (top 4 plots). We used the *Compton* system for these tests and random, single-constraint METIS, multi-constraint METIS, and PuLP-MM partitioning (from left to right, respectively) with random ordering. We note a large performance gap between Random partitioning and the other strategies. This is due to the implementation's computational and communication requirements for each task being dependent on the one hop neighborhood and per-part cut. These values are much higher with Random partitioning. We observe that PuLP-MM demonstrates the best performance, due to the fact that the multiple objectives are explicitly optimizing for these metrics while keeping work balance very consistent. Overall, we notice about a 5-10% total execution time improvement for PuLP versus the METIS variants.

5.3. Subgraph Counting Performance

We next compare partitioning and ordering impact on subgraph counting. We report the results from 16 nodes of *Blue Waters*. The speedups for each strategy on the 6 test graphs are given in Figure 3 and Table 5. Results for METIS on the uk-2005 and WebBase graphs is absent since execution times took over 24 hours for these instances. Several trends can be observed in Figure 3. The top subfigure gives the speedup of the communication phase of subgraph counting for each of the partitioning strategies relative to random partitioning. We again note considerable speedup for all partitioning methods. We observe overall that the single-objective multi-constraint methods, PuLP-M and METIS-M, result in the highest speedup overall. This analytic doesn't benefit as highly from the more communication-balanced multi-objective PuLP-MM method due to the overall higher communication requirements (the Twitter graph requires compression and transfer of several terabytes in total for data exchanges between tasks) and lower overall synchronization cost relative to PageRank, so total edge cut is observed to have a greater effect in practice. However, single constraint METIS showed the worst performance, due to large computational imbalance resulting from large differences in edges, and therefore computational load, between parts. This emphasizes the fact that a one-size-fits all solution is not optimal in practice, and implementation knowledge is required to extract the best performance for any given running application when deciding on a layout strategy.

The bottom subfigure of Figure 3 plots the speedup relative to random ordering for the BFS and RCM reordering strategies with PuLP-MM partitioning. Overall, we note about a 6% improvement for BFS and 5% improvement for RCM ordering relative to random. These improvements are

TABLE 3. ORDERING PERFORMANCE FOR BFS, RCM, AND RANDOM IN TERMS CO-LOCATION RATIO (CO-LOC. RATIO) AND log SUM OF GAP DISTANCES (GAP SUM RATIO) FOR 16-WAY AND 64-WAY PARTITIONING, AVERAGED ACROSS THE FIVE DIFFERENT PARTITIONING STRATEGIES.

Network	16-way partitioning					64-way partitioning				
	Co-loc. Ratio		Gap Sum Ratio			Co-loc. Ratio		Gap Sum Ratio		
	BFS	RCM	BFS	RCM	Rand	BFS	RCM	BFS	RCM	Rand
LiveJournal	0.115	0.010	0.036	0.034	0.043	0.104	0.014	0.009	0.006	0.012
Orkut	0.028	0.001	0.046	0.057	0.054	0.021	0.001	0.010	0.020	0.011
Twitter	0.032	0.005	0.037	0.035	0.038	0.026	0.006	0.013	0.010	0.016
uk-2005	0.659	0.176	0.015	0.022	0.046	0.582	0.184	0.005	0.006	0.011
WebBase	0.562	0.162	0.020	0.039	0.050	0.519	0.172	0.005	0.006	0.011
sk-2005	0.613	0.149	0.018	0.026	0.050	0.689	0.167	0.002	0.004	0.013
BSBM	0.146	0.146	0.040	0.040	0.062	0.146	0.153	0.007	0.006	0.009
LUBM	0.105	0.105	0.026	0.026	0.045	0.094	0.105	0.006	0.006	0.009
DBpedia	0.442	0.257	0.028	0.036	0.046	0.398	0.267	0.006	0.008	0.010
Overall	**0.298**	0.112	**0.030**	0.034	0.048	**0.274**	0.119	**0.007**	0.008	0.011

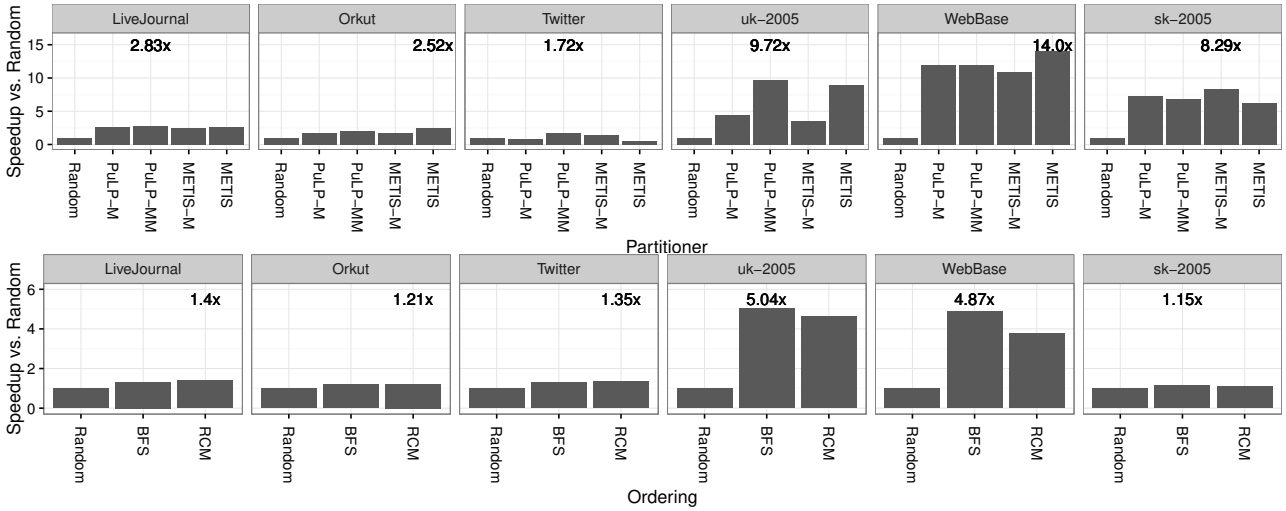

Figure 1. Communication speedup of the PageRank implementation on 16 nodes with various partitioning options (top) and computation speedup of PageRank with various ordering strategies (bottom).

TABLE 4. SPEEDUPS OF VARIOUS PARTITIONING AND ORDERING STRATEGIES VERSUS RANDOM PARTITIONING AND RANDOM ORDERING FOR PAGERANK.

Network	Partitioning				Ordering	
	METIS	METIS-M	PuLP-M	PuLP-MM	RCM	BFS
LiveJournal	2.560	2.453	2.561	2.832	1.404	1.325
Orkut	2.519	1.689	1.784	2.068	1.214	1.205
Twitter	1.454	1.459	1.871	1.716	1.346	1.292
uk-2005	8.913	3.518	4.427	9.725	4.641	5.039
WebBase	13.99	10.87	11.92	11.93	3.776	4.870
sk-2005	6.170	8.293	7.287	6.797	1.100	1.155
Overall	3.621	3.525	3.395	**4.465**	1.881	**1.970**

TABLE 5. SPEEDUPS OF VARIOUS PARTITIONING AND ORDERING STRATEGIES VERSUS RANDOM PARTITIONING AND RANDOM ORDERING FOR THE SUBGRAPH COUNTING BENCHMARK.

Network	Partitioning				Ordering	
	METIS	METIS-M	PuLP-M	PuLP-MM	RCM	BFS
LiveJournal	2.099	2.202	2.211	2.150	1.009	1.020
Orkut	2.307	2.400	2.411	2.350	1.014	1.015
Twitter	1.378	1.399	1.580	1.271	1.041	1.029
uk-2005	–	5.433	5.476	5.642	1.049	1.057
WebBase	–	3.412	3.375	3.311	1.125	1.148
sk-2005	5.568	5.675	5.772	5.621	1.072	1.091
Overall	–	3.033	**3.106**	2.961	1.051	**1.059**

much lower than PageRank's improvement due to considerably more information stored per-vertex, so greater cache locality has less of an effect in preventing re-accesses to main memory; however, we note even a modest 5%-6% consistent improvement can be noteworthy in this instance. On processors with larger cache, this relative improvement would be expected to increase. We also again note that using an ordering method which better optimizes for gap distances and co-location ratio appears to, on average, improve analytic performance.

We also give subgraph counting timelines in Figure 2 (bottom 4 plots). We note first the two extreme cases. Random shows the lowest total computation times at a high cost of communication, while single objective METIS results in low communication times but high total times during the execution stages. This is due to, as mentioned previously, unbalanced work among each task. We observe that balanced multi-objective PuLP partitioning gives the best tradeoff in terms of work balance and communication requirements and again notice about a 5-10% total execution time improvement for PuLP versus the METIS variants.

5.4. SSSP and BFS Performance

In this section, we analyze the performance of our SSSP and BFS implementation when using the different partition-

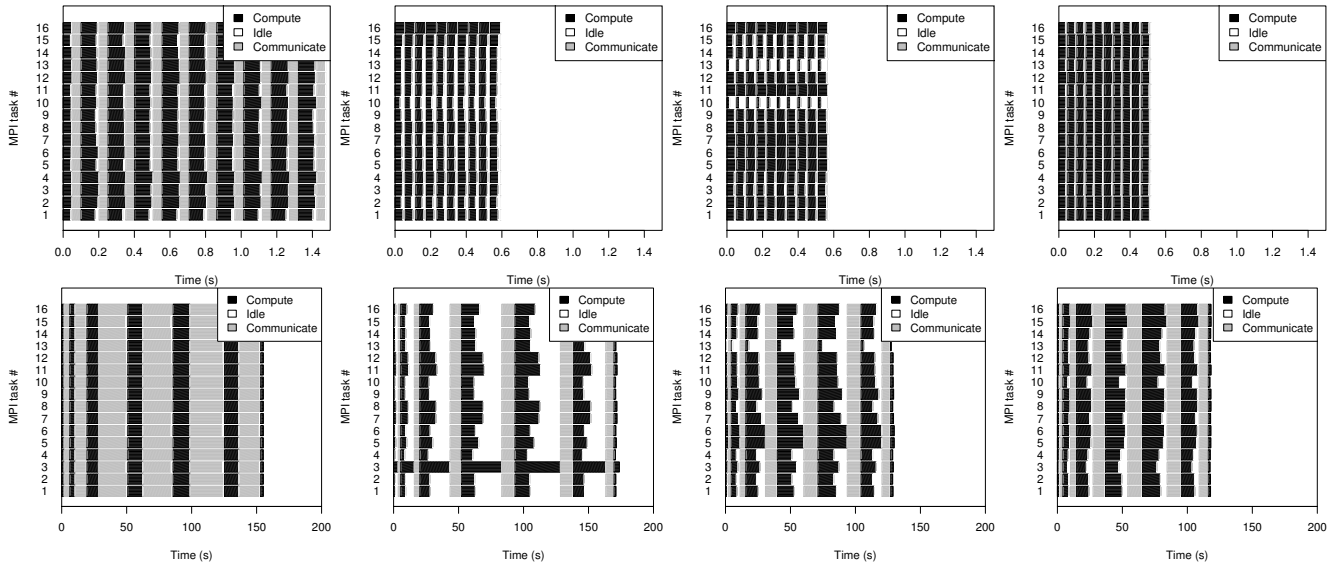

Figure 2. PageRank (top, 10 iterations on WebBase) and Subgraph counting (bottom, single color-coding iteration with a 10-vertex template on LiveJournal) execution timelines on 16 tasks and 32 threads with (left to right) random, single and multi-constraint METIS, and PuLP-MM partitioning strategies. Random ordering was used in all cases.

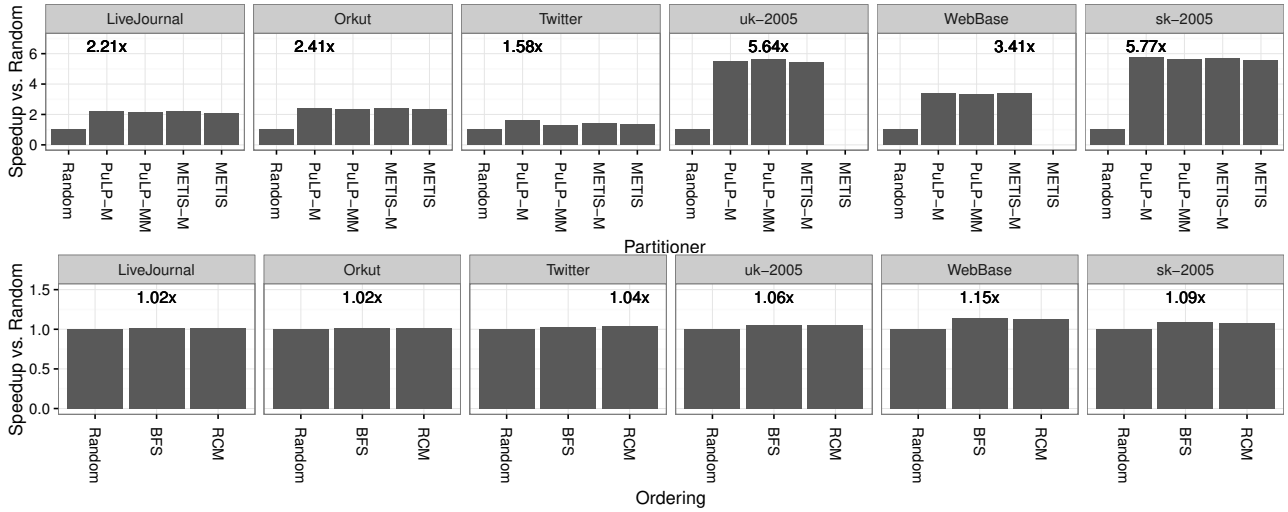

Figure 3. Speedups achieved with subgraph counting for total communication time of the various partitioning strategies relative to random partitioning, all with random ordering. Additionally, the speedups for the RCM and BFS orderings relative to random ordering with PuLP-MM partitioning.

TABLE 6. SPEEDUPS OF VARIOUS PARTITIONING AND ORDERING STRATEGIES VERSUS RANDOM PARTITIONING AND RANDOM ORDERING FOR SSSP.

	Partitioning				Ordering	
Network	METIS	METIS-M	PuLP-M	PuLP-MM	RCM	BFS
LiveJournal	1.575	1.422	1.400	1.372	1.117	1.097
Orkut	1.346	1.123	1.131	1.111	1.041	1.026
Twitter	1.172	1.224	1.109	1.141	1.063	1.094
uk-2005	4.008	3.122	3.044	3.187	1.399	1.328
WebBase	1.971	1.998	2.092	2.035	1.407	1.612
sk-2005	4.693	3.934	4.159	3.963	1.689	1.870
Overall	**2.125**	1.907	1.897	1.884	1.266	**1.304**

TABLE 7. SPEEDUPS OF VARIOUS PARTITIONING AND ORDERING STRATEGIES VERSUS RANDOM PARTITIONING AND RANDOM ORDERING FOR BFS.

	Partitioning				Ordering	
Network	METIS	METIS-M	PuLP-M	PuLP-MM	RCM	BFS
LiveJournal	1.100	1.141	1.110	1.124	1.007	0.987
Orkut	1.125	1.083	1.138	1.038	0.913	0.991
Twitter	1.127	1.060	1.047	1.076	1.099	1.109
uk-2005	1.655	1.929	1.920	1.763	1.023	1.068
WebBase	1.709	1.657	1.732	1.756	1.287	1.335
sk-2005	2.798	2.624	2.393	2.737	0.960	0.977
Overall	**1.494**	1.491	1.480	1.483	1.042	**1.071**

ing and ordering layouts. These results are from 64 node runs on *Blue Waters*. While the running time of distributed subgraph counting is dominated by large-scale data transfers during the communication phases, SSSP's performance is more dependent on intra-task computation, similar to PageRank, but has considerably less communication. BFS has the overall lowest computational requirements out of the tested benchmarks.

Figure 4 and Table 6 show the speedups for communication and computation for SSSP performance with 64 MPI tasks. The top subfigure of Figure 4 shows the communication speedups relative to random partitioning for the other partitioning strategies. Due to the relatively lower communication requirements for this SSSP implementation,

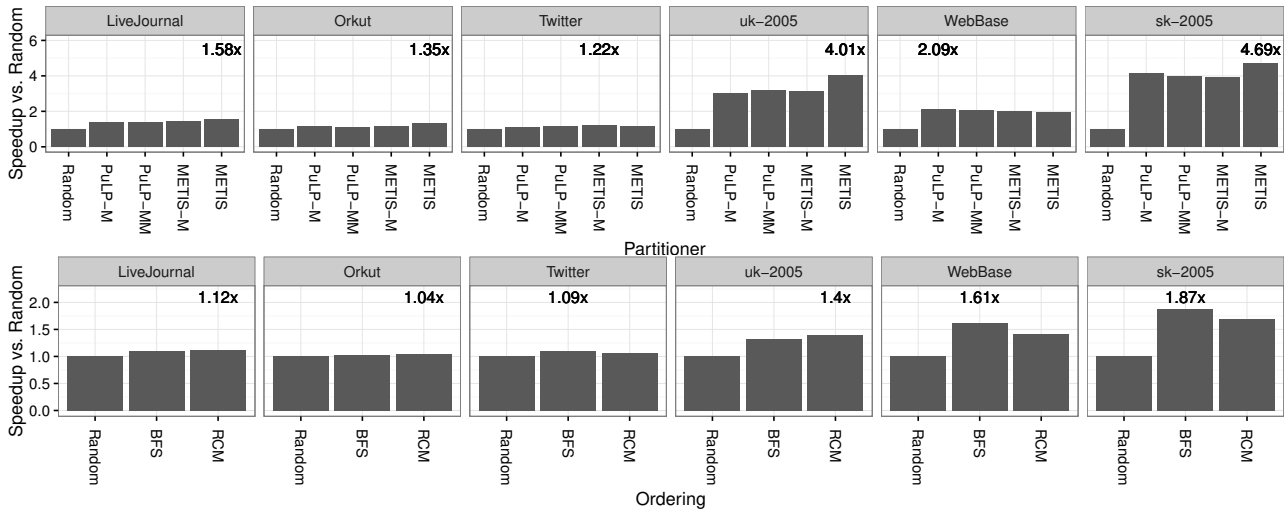

Figure 4. Communication time of SSSP implementation on 64 nodes with various partitioning options (top) and computation time of SSSP with various ordering strategies (bottom).

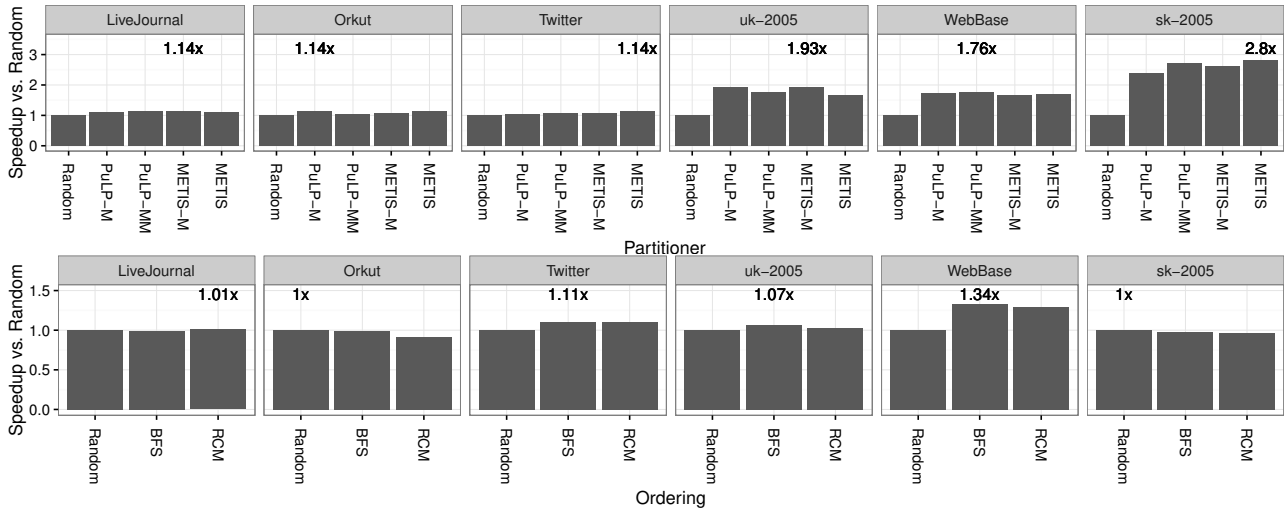

Figure 5. Communication time of BFS implementation on 64 nodes with various partitioning options (top) and computation time of BFS with various ordering strategies (bottom).

we correspondingly observe lower speedups relative to what was observed in Figure 3 with subgraph counting. We note that single-objective METIS gives the highest communication speedup, due to the lower overall communication load and synchronization costs which emphasize a lower total workload than explicit balance. We observe speedups for computation times on all graphs, and especially the web crawls, with both ordering strategies. We again note that an ordering which better optimizes for gap distance demonstrates better performance in practice, with BFS ordering showing around 30% speedup overall and about a 4% greater speedup relative to RCM.

The two subfigures of Figure 5 and Table 7 give the speedup in communication time with different partitioners and speedups in computation time with different orderings for the BFS implementation. We notice similar trends to SSSP in these plots. Overall, the lower total computation and communication workload of BFS contributes to lower speedups when using better ordering and partitioning strategies compared to random.

5.5. SPARQL Query Processing

We now study the impact of partitioning and ordering on the performance of RDF stores and SPARQL querying, a benchmark algorithm that is very different than the previous ones. In Table 8, we report replication ratios observed when an undirected 2-hop guarantee is enforced. Our RDF3X-MPI implementation uses a 2-hop guarantee to partition the graph, and a lower replication ratio indicates a smaller index size, which should translate to faster query times in practice. Table 8 compares the five partitioning methods using this metric for 16 and 64 parts on the 3 RDF graphs. Out of the 6 total graph-part count scenarios, the PuLP-MM method shows the lowest replication ratio for 3 of them. None of these partitioners are explicitly optimizing for this metric, so the performance of PuLP-MM in this instance is indirect. However, the additional cut balance resulting from the multi-objective optimization might be a possible factor.

In Table 9, we report sum of query times of RDF3X-MPI averaged over the BSBM, LUBM, and DBpedia data

TABLE 8. DISTRIBUTED RDF STORE REPLICATION RATIOS USING VARIOUS PARTITIONING STRATEGIES. AN UNDIRECTED 2-HOP GUARANTEE IS ENFORCED. LOWER VALUES ARE BETTER AND BEST VALUE FOR EACH GRAPH AND PARTS COUNT IS IN BOLD.

Partitioning	16-way			64-way		
	BSBM	LUBM	DBpedia	BSBM	LUBM	DBpedia
Random	7.256	10.58	5.580	22.07	34.84	10.50
METIS	5.566	9.714	**1.552**	19.02	36.56	2.257
METIS-M	5.577	9.146	**1.552**	19.01	38.02	**2.255**
PULP-M	5.308	**8.944**	1.905	14.73	36.86	2.815
PULP-MM	**5.112**	9.227	2.448	**13.78**	**29.94**	2.963

sets. We use a selection of queries modified from the Berlin SPARQL Benchmark. We report results from the 16 part runs for this test and additionally look at the performance effects of the three ordering strategies. PULP-MM partitioning with random ordering yields the best overall performance, with PULP-MM further demonstrating high performance when using the other two ordering strategies. This correlates loosely with PULP-MM demonstrating good 2-hop replication ratios. However, this relationship is tenuous at best, considering the speedups relative to random partitioning aren't nearly as drastic as with the other analytics we have run and PULP-MM doesn't show the best 2-hop replication ratios across all graphs when computing 16 parts.

The effect of ordering strategy on query times is interesting, in that the higher-locality orderings demonstrate correspondingly worse performance. To store the RDF data, RDF3X-MPI converts the input RDF graph structure into multiple indexes, which are created by sorting the RDF data, creating B+ trees, and then performing compression. We note that the worsened performance with locality-optimized ordering is most likely an artifact of this pre-processing stage. These results overall further emphasizes that knowledge of a graph analytic's algorithmic details is important when determining an optimal graph layout, as unexpected and counter-intuitive performance impacts are a real possibility.

TABLE 9. TOTAL QUERY TIMES IN SECONDS USING VARIOUS PARTITIONING AND ORDERING STRATEGIES, SUMMED OVER ALL 3 GRAPHS WITH 16 PARTS.

Partitioning	Ordering		
	Random	BFS	RCM
Random	3.41	4.58	4.32
PULP-M	3.41	3.97	3.94
PULP-MM	**3.32**	4.01	3.51
METIS	3.71	4.41	3.91
METIS-M	3.87	4.20	4.13

5.6. Empirical Evaluation Limitations

The current work is a preliminary evaluation of the impact of layout on distributed graph processing, and we recognize some limitations with this study.

1) *Size of graph instances studied:* We selected graphs to be of size large enough to give meaningful results, yet small enough to be processed by all partitioners and applications for consistency (memory limitations of METIS and subgraph counting set an upper bound on graph size). Based on our current analysis combined with prior work, we could expect similar results at larger scales.
2) *Choice of number of tasks:* Our results represent several hundred experimental runs in total, from the combinations of application-tasks-partitioning-ordering. More fine-grained comparative results in terms of task counts would have increased our experimental load by an order of magnitude, and we could not include them in this work.
3) *Omission of random graphs:* As we intend to study performance on realistic datasets, we omit performance analysis on synthetic graphs that do not directly mirror real-world graph topologies (e.g., Erdös-Rényi and Graph500 graphs).

6. Conclusions

In this paper, we evaluated the effects of varying partitioning and ordering objectives and constraints when running on graph analytics with differing computation and communication characteristics. We summarize our primary observations below:

1) The overall effect of an improved partitioning and ordering increases correspondingly with graphs of high diameter and with better vertex separators, such as with web crawls relative to social networks.
2) Constraining vertices and edges per part for partitioning is critical for performance on analytics with a moderate to high computational load.
3) Use of the secondary partitioning objective, max per-part cut, made a large impact on an iterative and bulk synchronous analytic PageRank.
4) Using an ordering that optimizes for cache locality versus bandwidth demonstrates consistent computational speedups for most of our analytics.
5) For optimal performance, knowledge of an analytic's internal algorithmic details is critical. Counter-intuitive performance may otherwise result.

This paper opens up several avenues for future work. Future work might involve more rigorously quantifying the link between partitioning/ordering objectives and real-world performance; e.g., if I improve a partition X by Y% in objective Z, how much will that improve the end-to-end execution time of analytic A? Larger graph datasets, varied novel applications, more complex ordering strategies, and additional complex objectives might also be investigated. Partitioning and ordering methods that better optimize for these graphs/application/objectives would prove promising for enabling scalability on future large-scale systems and datasets.

Acknowledgment

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070, ACI-1238993, and ACI-1444747) and the state of Illinois. Blue

Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This work is also supported by NSF grants ACI-1253881, CCF-1439057, and the DOE Office of Science through the FASTMath SciDAC Institute. Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. We thank Jeonghyung Park for preliminary investigations on the impact of ordering schemes on shared-memory FASCIA, Thap Panitanarak for developing the BFS and SSSP codes, and Sai Chirravuri for the RDF3x-MPI code.

References

[1] Y. Yasui and K. Fujisawa, "Fast, scalable, and energy-efficient parallel breadth-first search," in *Proc. Forum Math-for-Industry*, 2017.

[2] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM Journal on Computing*, vol. 14, no. 1, pp. 210–223, 1985.

[3] M. Ortmann and U. Brandes, "Triangle listing algorithms: Back from the diversion," in *Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2014.

[4] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Proc. Int'l. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[5] F. Checconi and F. Petrini, "Traversing trillions of edges in real-time: Graph exploration on large-scale parallel machines," in *Proc. IEEE Int'l. Parallel and Distributed Proc. Symp. (IPDPS)*, 2014.

[6] R. Pearce, M. Gokhale, and N. M. Amato, "Scaling techniques for massive scale-free graphs in distributed (external) memory," in *Proc. IEEE Int'l. Parallel and Distributed Proc. Symp. (IPDPS)*, 2013.

[7] G. M. Slota, K. Madduri, and S. Rajamanickam, "PuLP: Scalable multi-objective multi-constraint partitioning for small-world networks," in *Proc. IEEE Int'l. Conf. on Big Data (BigData)*, 2014.

[8] ——, "Complex network partitioning using label partitioning," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, pp. S620–S645, 2016.

[9] G. Slota and K. Madduri, "Parallel color-coding," *Parallel Computing*, vol. 47, pp. 51–69, August 2015.

[10] M. Deveci, K. Kaya, B. Uçar, and U. V. Catalyurek, "Fast and high quality topology-aware task mapping," in *Proc. IEEE Int'l. Parallel and Distributed Proc. Symp. (IPDPS)*, 2015.

[11] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in *Proc. USENIX Conf. on Operating Systems Design and Implementation (OSDI)*, 2012.

[12] A. Ching and C. Kunz, "Giraph: Large-scale graph processing infrastructure on Hadoop," in *Proc. Hadoop Summit*, 2011.

[13] G. Karypis and V. Kumar, "MeTis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. version 5.1.0," http://glaros.dtc.umn.edu/gkhome/metis/metis/download, last accessed Mar 2017.

[14] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," in *Proc. IEEE Int'l. Parallel and Distributed Proc. Symp. (IPDPS)*, 2015.

[15] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proc. 1969 24th ACM Nat'l. Conf.*, 1969.

[16] K. I. Karantasis, A. Lenharth, D. Nguyen, M. J. Garzarán, and K. Pingali, "Parallelization of reordering algorithms for bandwidth and wavefront reduction," in *Proc. Int'l. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.

[17] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *Proc. ACM Int'l. Conf. on Knowledge Discovery and Data mining (KDD)*, 2009.

[18] G. M. Slota, S. Rajamanickam, and K. Madduri, "Distributed graph layout for scalable small-world network analysis," arXiv.org e-Print archive, https://arxiv.org/abs/1701.00503, 2017.

[19] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy, "Load balancing and data locality in adaptive hierarchical n-body methods: Barnes-hut, fast multipole, and radiosity," *Journal of Parallel and Distributed Computing*, vol. 27, no. 2, pp. 118–141, 1995.

[20] C.-W. Ou, M. Gunwani, and S. Ranka, "Architecture-independent locality-improving transformations of computational graphs embedded in k-dimensions," in *Proc. ACM Int'l. Conf. on Supercomputing (ICS)*, 1995.

[21] G. M. Slota, S. Rajamanickam, and K. Madduri, "A case study of complex graph analysis in distributed memory: Implementation and optimization," in *Proc. IEEE Int'l. Parallel and Distributed Proc. Symp. (IPDPS)*, 2016.

[22] N. Alon, R. Yuster, and U. Zwick, "Color-coding," *J. ACM*, vol. 42, no. 4, pp. 844–856, 1995.

[23] A. Buluç and K. Madduri, "Graph partitioning for scalable distributed graph computations," in *Graph Partitioning and Graph Clustering*, D. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, Eds. AMS, 2013, ch. 6, pp. 81–100.

[24] E. G. Boman, K. D. Devine, and S. Rajamanickam, "Scalable matrix computations on large scale-free graphs using 2D graph partitioning," in *Proc. Int'l. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[25] T. Panitanarak and K. Madduri, "Performance analysis of single-source shortest path algorithms on distributed-memory systems," in *Proc. SIAM Workshop on Comb. Sci. Comp. (CSC)*, 2014.

[26] U. Meyer and P. Sanders, "Δ-stepping: a parallelizable shortest path algorithm," *J. Algs.*, vol. 49, no. 1, pp. 114–152, 2003.

[27] V. T. Chakaravarthy, F. Checconi, F. Petrini, and Y. Sabharwal, "Scalable single source shortest path algorithms for massively parallel systems," in *Proc. IEEE Int'l. Parallel and Distributed Proc. Symp. (IPDPS)*, 2014.

[28] T. Neumann and G. Weikum, "The RDF-3X engine for scalable management of RDF data," *VLDB J.*, vol. 19, no. 1, pp. 91–113, 2010.

[29] S. K. Chirravuri, "RDF3X-MPI: A partitioned RDF engine for data-parallel SPARQL querying," Master's thesis, The Pennsylvania State University, 2014.

[30] "Stanford large network dataset collection," http://snap.stanford.edu/data/index.html, last accessed Mar 2017.

[31] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in Twitter: The million follower fallacy," in *Proc. Int'l. Conf. on Weblogs and Social Media (ICWSM)*, 2010.

[32] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. Int'l. Conf. on World Wide Web (WWW)*, 2004.

[33] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "UbiCrawler: A scalable fully distributed web crawler," *Software: Practice & Experience*, vol. 34, no. 8, pp. 711–726, 2004.

[34] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, pp. 1–25, 2011.

[35] C. Bizer and A. Schultz, "The Berlin SPARQL benchmark," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, pp. 1–24, 2009.

[36] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2-3, pp. 158–182, 2005.

[37] M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo, "DBpedia SPARQL benchmark - performance assessment with real queries on real data," in *Proc. Int'l. Semantic Web Conf. (ISWC)*, 2011.

[38] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[39] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. IEEE Int'l. Conf. on Data Mining (ICDM)*, 2012.