The Locality Long Tail: Understanding Data Locality in Parallel Graph Algorithms

***Abstract*—The rise of big data analytics has contributed to the increasing popularity and scale of graph datasets, positioning graph analysis as an important and growing research area. Real-world graphs, derived from data such as social networks, web page links, and paper citations, possess fundamental properties that differ from traditional graphs like trees or meshes, resulting in different execution characteristics.**

In this paper, we analyze the performance characteristics of a core set of graph analysis algorithms across several informatics, physical, and synthetic graph datasets using a multicore microarchitectural simulator. Despite some inherent data locality, modern caching techniques show limited effectiveness for graph algorithms, which are communication-bound. We find that cache miss rates are an unreliable metric for data locality because they are also dependent on cache size, dataset size, and cache replacement policy. Thus, we use cache configuration-independent locality analysis techniques, including reuse distance and a probability-based locality score, to analyze data locality in graph algorithms.

Based on our analysis of data locality, we find that graph algorithm data access patterns are not friendly to an LRU cache replacement policy. Further, we show that data access patterns relate to algorithm characteristics, graph dataset structure, and vertex degree. These insights indicate that utilization of algorithm- and dataset-specific information paired with small changes to the memory hierarchy could improve graph analysis algorithm performance.

I. Introduction

In recent years, interest in graph analytics has exploded across many domains, including machine learning, data mining, and scientific computation. Increasing dataset size coupled with a transition from graphs based on physical systems to informatics graphs has brought the performance challenges of graph analytics to the forefront. These large-scale graph datasets come from online social networks [1], genomics [2], computational biology [3], and the Web [4].

With datasets containing millions or billions of vertices and edges, the use of parallel algorithms is essential for time- and resource-efficient graph analysis. The data parallelism inherent in graph analytics maps well to the dozens of cores in modern multicore systems, which support over a terabyte of main memory that can often store the algorithm's working set. Though compute clusters are used for large-scale graph processing, including the distributed graph analytics frameworks Pregel [5] and GraphLab [6], [7], when the dataset can fit in memory on a single machine, execution is much faster due to the communication-intensive nature of graph algorithms. In fact, several graph analytics frameworks, such as GraphMat [8], Polymer [9], Ligra [10], X-Stream [11], and Green-Marl [12], target single-machine multicore systems.

Due to graph algorithms' data-intensive nature, the memory wall, or the lower speed of main memory compared to compute, is a fundamental limiting factor in performance. Modern processors use a hierarchy of on-chip data caches paired with hardware data prefetching to work around the memory wall. While it is well established that these techniques are less effective for graph algorithms than regular codes, recent literature disagrees on cache miss rates by 10x [13], [14]. Further, some sources claim inherent 'low' or 'lack of' data locality in graph algorithms [13], [15], [16], and others claim 'substantial' locality [14].

To understand data locality in graph algorithms, we perform a thorough quantitative analysis. Using architectural simulation, we analyze the data access patterns and locality of a set of optimized graph algorithms on a shared-memory multicore architecture. This paper aims to formalize the characteristics of parallel graph algorithms and provide insights for optimizing the performance and efficiency of graph analytics.

The insights from our analysis are:

- Cache miss rates are dependent on dataset size, cache size, and replacement policy, and thus do not directly correspond to data locality.
- Data reuse patterns in graph algorithms are poorly exploited by an LRU cache replacement policy.
- Locality is impacted more by algorithm than dataset structure.
- Data access patterns correlate with graph dataset structure and vertex degree.

II. Graph Background and Properties

The properties of graph datasets can significantly impact graph algorithm execution patterns and performance. The *diameter* of a graph is the maximum shortest distance between any pair of vertices. A vertex's *degree* is equal to the number of edge connections it has to other vertices. The *degree distribution* is the probability distribution of the degrees over all of the vertices in the graph.

Graphs commonly used for analysis fall into two distinct classes: informatics graphs and meshes. Graphs representing physical networks or structures, such as road networks or finite-element meshes, typically have large diameters and a uniform distribution of small degrees. Thus, these graphs can be partitioned effectively for parallel execution; most communication is local and the amount of work per vertex is similar across vertices. These properties simplify load balancing, and are typical of datasets used in traditional graph

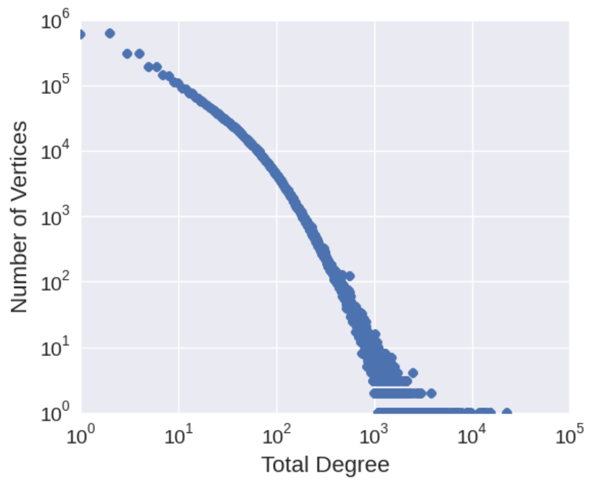


Fig. 1: The total degree distribution of the LiveJournal online social network follows a power law (scale-free property).

processing, such as path-finding in road networks or finite element analysis.

On the other hand, informatics graphs, like social networks and web-link networks, typically exhibit both the small-world and scale-free properties. The *small-world property* means that the graph diameter is very small: $\mathrm{O}(\log\ n)$, where n is the number of vertices [17]. The *scale-free property* states that the degree distribution of the vertices follows a power law; many vertices have only a few edges and a few vertices have a very large number of edges [18]. The degree distribution of the LiveJournal online social network in Figure 1 demonstrates the scale-free property.

The properties of informatics graphs reduce the efficiency of parallel algorithms. Small-world graphs are often difficult to partition – the 'short-cut' edges that connect otherwise far away regions of the graph make it difficult to partition the graph with few edge crossings, or a small cut. This leads to load imbalance and additional communication overhead. The scale-free property makes load-balancing difficult. Since the computation per vertex is often proportional to the vertex's degree, high degree vertices do more work.

III. Data Locality

We seek to understand the sources of locality in graph algorithms to boost performance by maximizing data reuse. We consider both temporal and spatial data locality.

Given that a particular memory location is accessed at a point in time, temporal locality is the likelihood that the same location will be accessed again in the future; spatial locality is the likelihood that a nearby location will be accessed in the future.

We compute a temporal locality function for each algorithm and dataset combination based on reuse distance. For an application memory trace, the reuse distance is defined as the number of distinct data elements accessed between two consecutive accesses of the same element. Measuring reuse distance is advantageous because it is independent of cache configuration and allows analysis of program behavior separately from a given machine or cache design [19]. Our temporal locality function is based on the one described in [20], which is the cummulative distribution (CDF) of reuse distances from an application memory trace. To make the numbers easier to manipulate, we use the reuse distance plus one. This means that a sequential access to the same address has a distance of one instead of zero.

Algorithm	Type	Iterative	Result Type
BFS	Traversal	N	Vertex-to-vertex distance
BC	Centrality	N	Vertex property
PR	Link Analysis	Y	Vertex property
HITS	Link Analysis	Y	Vertex property
CDT	Connectivity	N	Graph or cut property

TABLE I: Comparison of graph analysis algorithm properties.

IV. Methodology

A. Algorithms

We select five common graph kernels with a variety of characteristics. Their properties are listed in Table I.

Breadth-First Search (BFS) is a graph traversal kernel that explores the vertices in a graph beginning from a root vertex and records their distance from the root. BFS is a foundational component of many graph algorithms, including betweenness centrality and connected components. We implement a level-synchronous, direction-optimizing algorithm [28], [29]. We run BFS from five random vertices.

Betweenness Centrality (BC) describes a vertex's importance in a graph based on how frequently it lies on shortest paths between pairs of vertices, an important metric in social network analysis [30]. A vertex's BC equals the fraction of shortest paths which pass through that vertex, summed over paths between all pairs of vertices. It is computed with a parallel BFS traversal followed by a parallel BFS over the reverse edges [31]. We estimate BC from five random vertices.

PageRank (PR) is an iterative link analysis algorithm that assigns a numerical weight to vertices, indicative of their relative importance. It was originally developed to rank Web pages [32]. Each vertex's PageRank is a function of the PageRank scores of its incoming neighbors. We run the algorithm for 30 iterations.

Hubs and Authorities (HITS) is an iterative algorithm that assigns hub and authority scores to vertices based on a mutually-reinforcing relationship [33]. A good hub points to many authorities, and conversely, a good authority is pointed to by many hubs. HITS employs the same meta-algorithm as SALSA [34], which can be used as the basis of a user recommendation algorithm for online social networks [35]. We run HITS for 30 iterations.

Conductance (CDT) measures the degree that a graph is tightly-knit [36]. It equals the fraction of edges which cross a cut, or partition of a graph into two sections, normalized by the sum of degrees on one side of the cut. The conductance of a graph is the sum of conductances

Name	Graph Description	Type	Vertices	Edges	Degree	Diameter	Size (MB)	Directed
Flickr	Flickr online social network [21]	Social	2,302,925	33,140,018	28.8	28	136	Y
Livej	LiveJournal online social network [22], [23]	Social	4,848,571	68,993,773	28.5	22	282	Y
Orkut	Orkut online social network [24]	Social	3,072,441	117,185,083	76.3	11	906	N
Wiki	Links in English Wikipedia [25]	Web	15,172,740	131,166,252	17.2	376	555	Y
Germanroad	German road network [26]	Road	11,548,845	12,369,181	2.1	6,322	139	N
Ferrari	Ferrari 333SP finite element (FE) mesh [26]	Mesh	3,712,815	11,108,633	6.0	1,294	99	N
RandomGnm*	Uniform random synthetic graph [27]	Uniform	10,000,000	79,999,969	16.0	14	344	Y
RMATg500	R-MAT synthetic graph, Graph500 benchmark [26]	R-MAT	1,048,576	44,619,402	85.1	7	345	N

TABLE II: Real-world and synthetic graph datasets used for experiments. Dataset file sizes are in the Green-Marl binary format. Degree is the average degree. For undirected graphs, we list the number of undirected edges, though our implementation represents undirected edges using two directed edges, one in each direction. *Duplicate edges are removed.

over all possible graph cuts. We compute conductance for four random graph cuts.

We implement all of the graph algorithms in Green-Marl, a domain-specific language for parallel graph analysis on shared memory multicores [12]. From the user's high-level code, the Green-Marl compiler produces an efficient C++ implementation, parallelized using OpenMP. Results are measured during the main loop of each algorithm, excluding initial graph dataset loading and final results reporting.

B. Graph Datasets

We consider eight graph datasets, both real-world and synthetic, as inputs to the algorithms described in Section IV-A. The graph instances are large enough to parallelize and tax the memory hierarchy, but can fit into a single machine's memory and be simulated in manageable time. Table II describes these graphs and their basic properties. The datasets were acquired from public repositories [37], [38], [39], and RandomGnm was generated using the GTgraph software [27].

We focus on real-world informatics graphs, such as social networks and web link graphs, that exhibit the small-world and scale-free properties (Section II). We include four graphs in this category, with a range of sizes and average degrees.

For comparison, we include two datasets representing physical networks, which do not have the small-world or scale-free properties: a road network (Germanroad) and a finite element mesh (Ferrari). In contrast to the informatics graphs, they have diameters in the thousands, and average degrees no more than six.

We also include two synthetic networks: a uniform random graph (RandomGnm), constructed by adding edges between randomly chosen pairs of vertices, and an R-MAT graph (RMATg500), which has the small-world and scale-free properties similar to real-world networks [40].

C. Zsim Architectural Simulator

For simulations, we use zsim [41], which models a multiprocessor system comprised of processing cores, each with a single execution thread, and a multi-level cache hierarchy connected to a main memory model. Zsim leverages dynamic binary translation and instruction-driven timing models to achieve fast simulation times. Since memory access behavior dominates graph algorithm performance, we can evaluate real-world datasets, which provides more realistic results.

The architectural parameters for our 16-core simulator configuration appears in Table III. For applicable experiments, we note variation to these baseline system parameters. We use zsim's out-of-order quad-issue core model, based on Intel's Westmere architecture. Where specified, we instead run zsim with a simple core model, with core IPC equal to one. We use the MD1 memory model, which is based on a nominal memory latency and maximum memory bandwidth. To model queueing delay, as load on the memory increases, the latency experienced by each access increases.

Parameter	Value
Num. Cores	16 @ 3.0 GHz
L1-I Cache per core	32 KB, 4-way assoc., 4 cycles
L1-D Cache per core	32 KB, 8-way assoc., 4 cycles
L2 Cache shared per 2 cores	512 KB, 16-way assoc., 10 cycles
L3 Cache shared	16 MB, 16-way assoc., 38 cycles
Cache Line Size (bytes)	64
Cache Configuration	Inclusive, MESI coherence
DRAM Bandwidth	12.8 GB/sec
DRAM Latency	175 cycles
Num. Memory Controllers	1

TABLE III: Zsim architectural parameters.

The cache and memory latencies are estimated based on measurements of the Nehalem architecture [42]. We use the lowest latency values for cache and memory, which makes the performance of our simulated memory hierarchy optimistic compared with a real system with non-uniform cache and memory access effects. Thus, we expect worse cache performance on real machines than our simulations predict, which makes cache optimizations for graph algorithms even more critical.

We run a recent version of Zsim from the GitHub repository [43], instrumented with additional performance counters, and Pin library version 2.14.

D. Locality Measurements

We use two algorithms for data locality analysis. One is the reuse distance algorithm, also known as LRU stack distance [44], which measures temporal locality. The other is a probability-based locality analysis [45], which quantifies both spatial and temporal locality.

To create the memory traces used in these analyses, we generate a memory trace for each thread's loads and stores with zsim using the simple core model. We interleave all threads' accesses into one stream, with ordering based on the execution cycle of the accesses. Due to the length of the memory traces produced from our graph algorithms, we analyze locality for only one "iteration" of each algorithm: one start node for BFS and BC, one iteration for PR and HITS, and one graph cut for CDT. The address granularity for accesses is 4 bytes, which independently captures the reuse of most data elements, including `int`, `float`, `long`, and `double`. For data structure-specific reuse distance, we compute the reuse distances from the full trace, then filter by the addresses specific to the data structure.

Our reuse distance algorithm implements in C++ the Bennett and Kruskal partial sum optimization [46], which reduces the computational complexity from O (n^2) to O $(log\ n)$, where n is the length of the memory trace. We use locality probability analysis code provided by its author, which is implemented in C [45]. Our code is compiled with gcc/g++ version 4.8.2 with the -O3 option, and run on Ubuntu 14.04.

V. Experimental Results

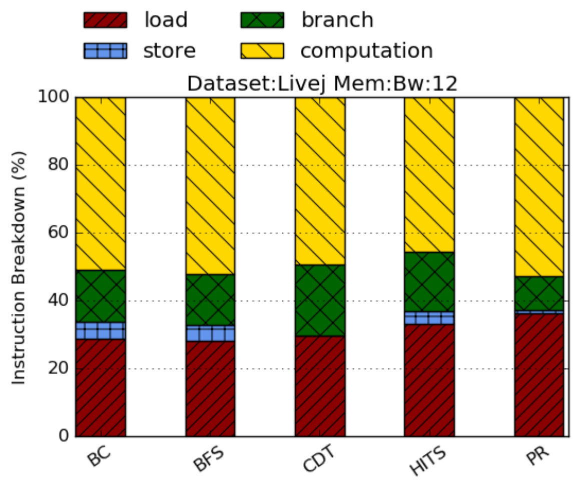
In this section, we present our experimental results, which includes an overview of the bottlenecks in the memory subsystem, and results of the locality techniques described in Section III. As a running example, we consider the PageRank algorithm on the Livej dataset. We select this algorithm because it has taxing memory access patterns and is a popular graph analysis algorithm. We choose this dataset because it is a social network with the small-world and scale-free properties, which makes it a good representative of the class of informatics graphs.

```
Procedure PageRank(
 G: Graph, e,d: Double, max_iter: Int,
 PR: Node_Prop<Double>(G)) {
  Double diff = 0; // Convergence error
  Int cnt = 0; // Num. of iterations
  Double N = G.NumNodes();
  G.PR = 1.0; // Init PR
  Do { // Main Iteration
    diff = 0.0;
    Foreach (t: G.Nodes) {
      Double val = (1-d) + d * Sum(w: t.InNbrs) {
        w.PR / w.OutDegree() }; // compute new PR
      diff += | val - t.PR |;
      t.PR <= val @ t; } // synchronous write
    cnt++;
  } While ((diff > e) && (cnt < max_iter)); }
```

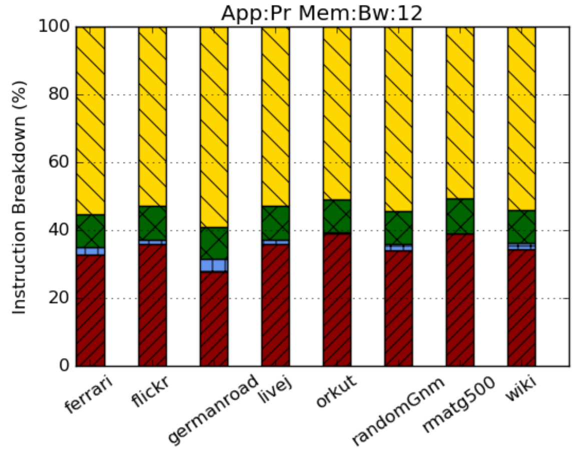
Fig. 2: PageRank algorithm implemented in Green-Marl

A. The Memory Wall

To achieve good performance, the data-intensive nature of graph algorithms necessitates fast access to large amounts of data. Modern processors partially mitigate the negative performance effects of the memory wall with a multi-level hierarchy of high-capacity data caches. Caches, which are on the order of tens of megabytes even for large server chips, are not particularly effective for graph algorithms, which must access gigabytes of data.



(a) All algorithms on Livej dataset



(b) PR algorithm on all datasets

Fig. 3: Dynamic instruction breakdown.

The memory wall causes particular performance difficulty because there is little computation for each data access, which places a heavy load on the memory hierarchy in comparison to the compute resources. Load and store instructions make up about one-third of the total instructions executed, as shown in Figure 3. There are on average five computation instructions for every three memory instructions, which gives few compute operations between memory accesses to hide the many-cycle latency to the last-level cache (LLC) or main memory. The computational instructions are often integer-based, but some algorithms perform floating point operations, including BC, HITS, and PR.

Due to the low computation-to-memory access ratio and frequent cache misses, many of the loads executed in an algorithm suffer the latency of an access to main memory. In combination with limited memory bandwidth, this manifests as low processor efficiency, which we observe as low instructions per cycle (IPC <1 on a quad-issue core), across all applications and datasets.

Scatter-gather memory access patterns are another factor that results in inefficient use of the memory system. In the

PageRank algorithm (see code listing in Figure 2), each vertex reads the PageRank values of its in-neighbors (line 11), which are distributed throughout the PageRank value array, called PR. Though each PageRank value is an 8-byte `double`, the system must load a full 64-byte cache line, even if only a single PageRank value is accessed before eviction.

Figure 4 shows the percentage of 8-byte words accessed in all 64-byte cache lines before the line is evicted. Generally, only one 8-byte word is touched for over 80% of cache lines brought into the cache. The Ferrari and Germanroad datasets have higher cache line utilizations because of their smaller sizes, so more data elements are accessed before a cache line is evicted.

In addition to wasting bandwidth with unused data, modern DRAMs are much slower at random data access than sequential. If all access is random, peak memory bandwidth can only reach 22% of maximum sequential bandwidth for one core, or 55% for 16 cores [11]. Since most of the graph algorithm data access patterns are not sequential, this further compounds the issue of limited memory bandwidth.

B. Locality Results

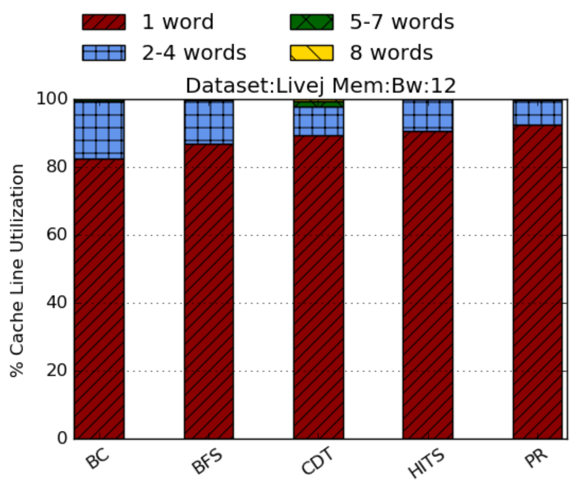
Figure 5 shows the temporal locality function of the PageRank algorithm on the Livej dataset. This function, which is a CDF, shows the percentage of total accesses which are captured by a given reuse distance. It may not reach 100% if some data is never reused during one iteration of the algorithm.

To show how many accesses would be hits in a hypothetical fully associative cache with the least recently used (LRU) replacement policy, we superimpose vertical lines on the histogram corresponding to various cache sizes. Accesses to the right of a cache size line would be cache misses, because too many pieces of data have been brought into the cache since the address's previous reference. For example, about 39% of data accesses would be hits on a machine with a single fully-associative LRU cache sized at 16 MB.

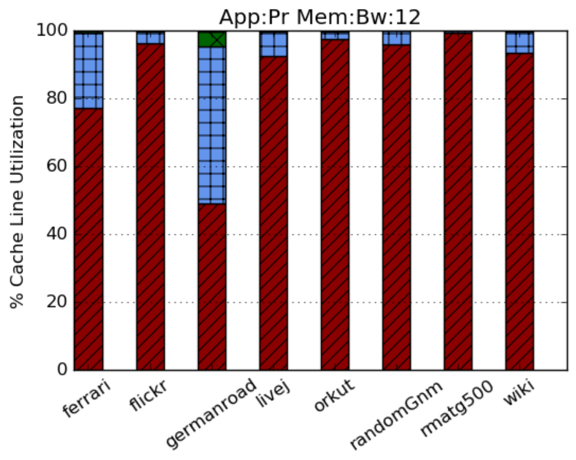
In Figures 6 and 7, we present a comparison of the temporal locality functions for each algorithm on all datasets and all algorithms on each dataset, respectively. From Figure 6, it is clear that BFS and BC have much better temporal locality PR, HITS, and CDT. From Figure 7, we can see that the temporal locality differences between graph datasets are much less significant than the differences between graph algorithms.

The long reuse distance accesses on the right side of the plots are a poor match for the LRU replacement policy because cached data is likely to become least-recently used and evicted before it has the opportunity to be reused. This suggests that a thrash-resistant replacement policy, such as the simple least-frequently used (LFU) policy, or a more sophisticated policy like DIP [47] or RRIP [48], may improve cache hit rates.

Figures 8a and 8b show data locality as a function of neighborhood size (spatial locality) and near future window size (temporal locality) for the Livej dataset on PR. In Figure 8a, slices of constant neighborhood size correspond to the reuse distance CDFs for different cache block sizes. To match the reuse distance plots, which have a 4-byte address mask, we



(a) All algorithms on Livej dataset



(b) PageRank algorithm on all datasets

Fig. 4: Percentage of 8B words accessed in a 64B cache line before the line is evicted, over all cache lines in the L1 data cache accessed during execution.

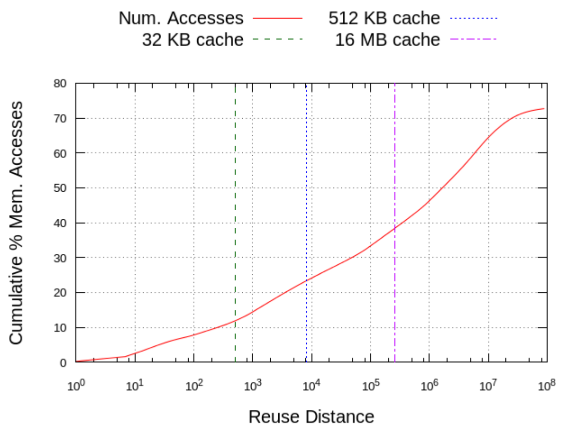
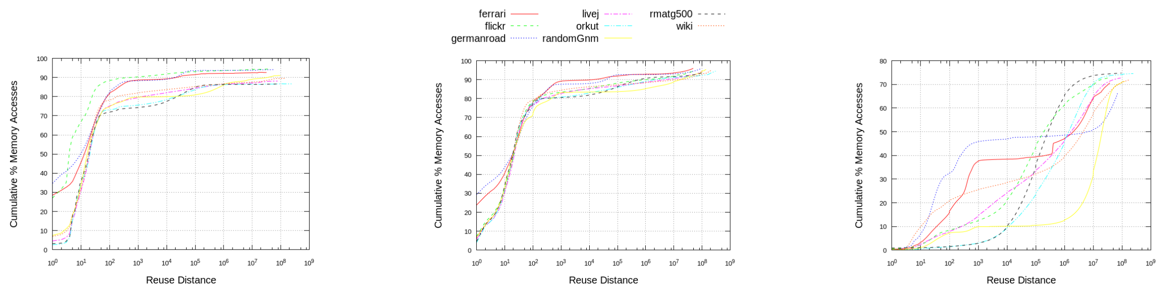


Fig. 5: Temporal locality function for the PageRank algorithm on the Livej dataset. Vertical lines represent cache sizes.


(a) BFS (b) BC (c) PR

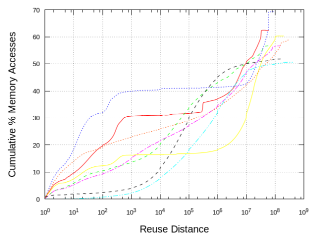
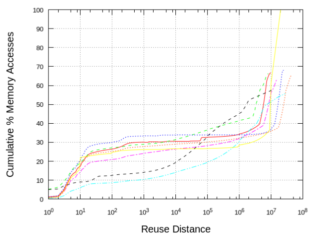
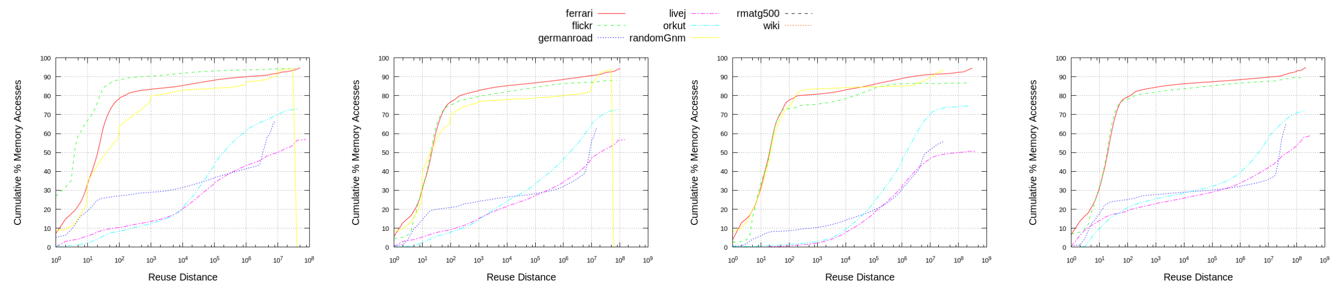


(d) HITS (e) CDT

Fig. 6: Comparison of temporal locality functions for each algorithm on all datasets.


(a) Flickr (b) Livej (c) Orkut (d) Wiki

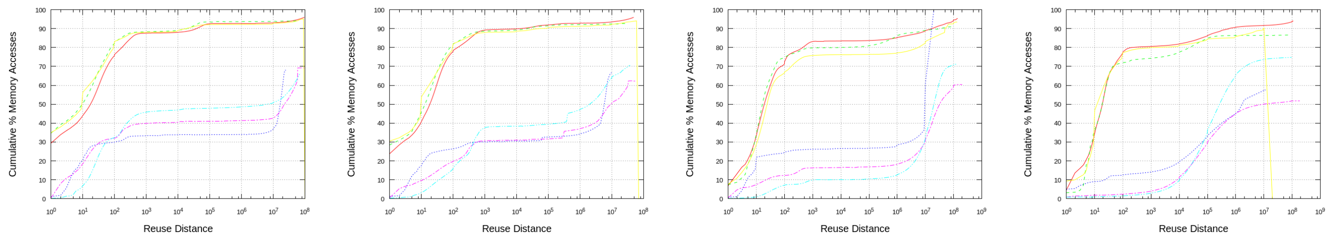

(e) Germanroad (f) Ferrari (g) RandomGnm (h) Rmatg500

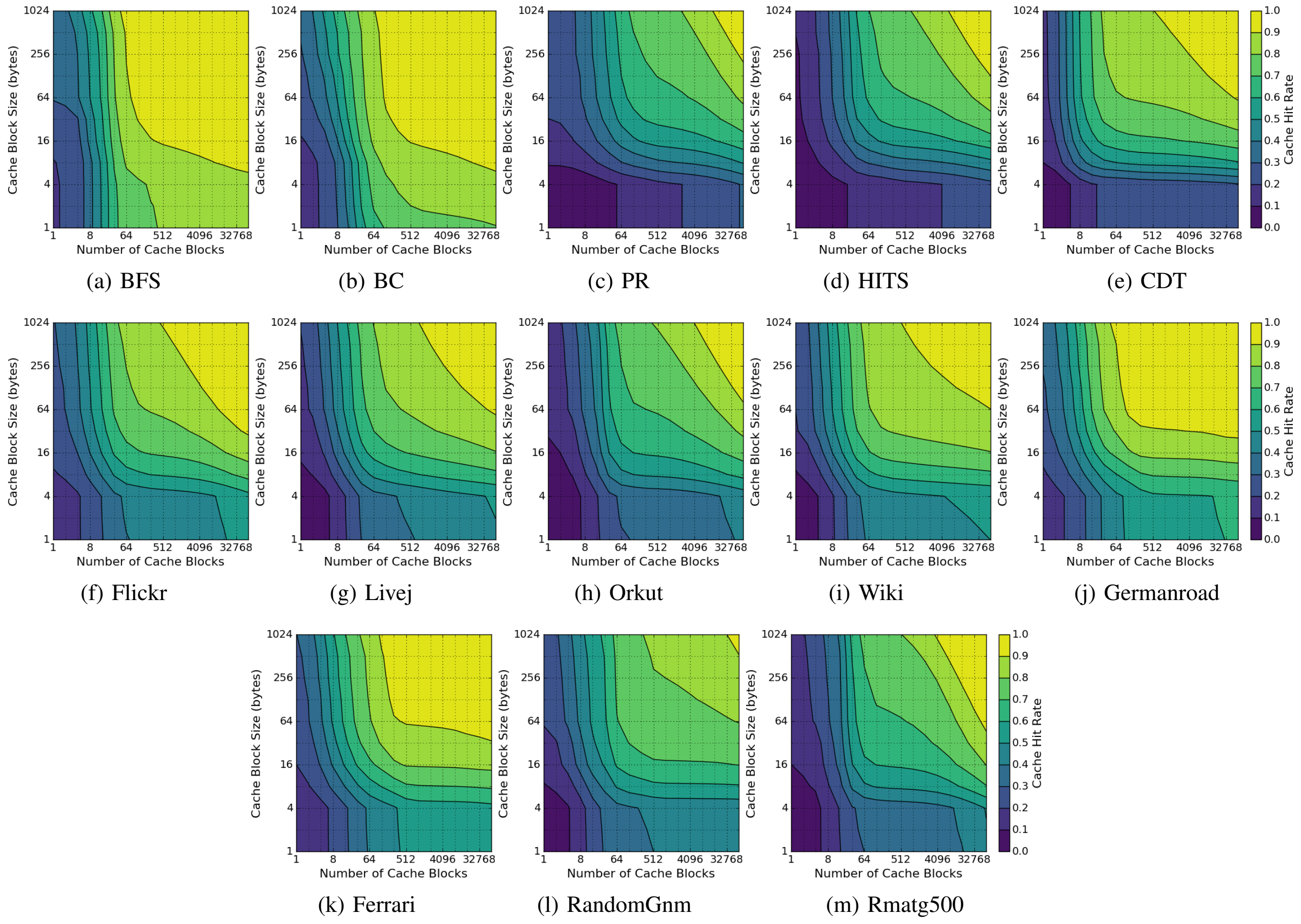
Fig. 7: Comparison of temporal locality functions for all algorithms on each dataset.

use locality measurements with neighborhood size determined by address mask.

Most of the locality is temporal, shown in Figure 8a as an increase in locality with an increase in near future window size. Increasing the neighborhood size has some effect on the overall locality, but it increases slowly once the neighborhood size reaches four bytes, which is the size of most data elements. Similar results are observed for PR on the other datasets, and for HITS and CDT. For BFS and BC, spatial locality has a less significant contribution.

In Figure 10, we plot average locality profiles as 2D contours for each algorithm and each dataset. For example, PR (c) is an average of the locality scores of each of the eight input graphs run on the PR algorithm; Livej (g) is an average of the Livej input graph run on all five algorithms. The average standard deviation for each locality profile is shown in Table IV. There is more variation in locality for different applications than for different datasets on the same application. As such, locality optimizations may require tailoring for each algorithm or class of algorithms, but are more generalizable across datasets.

The size and structure of graph datasets are important factors in data locality. For example, the small degree and uniform structure of the Ferrari and Germanroad graphs provide better temporal locality than scale-free graphs because of the greater potential for reuse when vertices are more tightly-knit.



(a) BFS (b) BC (c) PR (d) HITS (e) CDT

(f) Flickr (g) Livej (h) Orkut (i) Wiki (j) Germanroad

(k) Ferrari (l) RandomGnm (m) Rmatg500

Fig. 10: Locality score 2D contour plots averaged over algorithms or datasets.

Algorithm	Std. Dev. (10^{-3})
BFS	7.0
BC	1.7
PR	10.2
HITS	10.1
CDT	7.7

Dataset	Std. Dev. (10^{-3})
Flickr	39.3
Livej	26.2
Orkut	38.4
Wiki	22.4
Germanroad	18.1
Ferrari	23.8
RandomGnm	36.4
Rmatg500	37.7

TABLE IV: Average standard deviations of locality scores for algorithms and datasets.

C. Locality Prediction

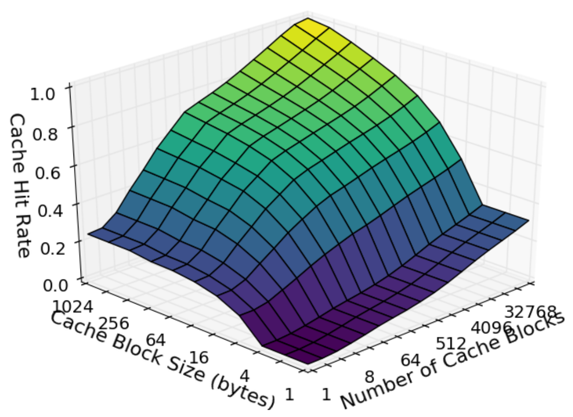
We can leverage knowledge about each algorithm's data access patterns and the structure of graph datasets to predict the locality of data, which could be used to improve cache performance.

By performing a simple hand analysis of the access patterns to the primary data structures in each algorithm, we can predict which data structures are reused within one iteration of the algorithm and whether the reuse distance correlates to vertex in- or out-degree. For each application, each data structure has an access pattern with some consistency across input datasets.

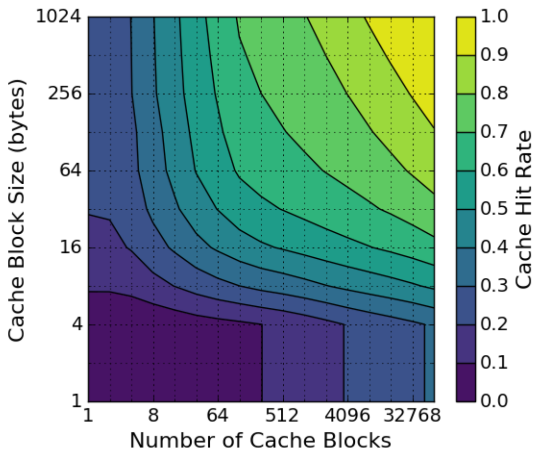
We extend reuse distance analysis to study the access patterns of specific data structures in each algorithm. Figure 9 shows the reuse distance PDF for the array that holds the PageRank values for each vertex in the PageRank algorithm. The shape of the reuse distance distribution is similar to that of reuse distance for the full algorithm.

To fully understand locality at the data structure level, we must examine the inner workings of our algorithms. Green-Marl represents the graph vertices and edges using the compressed sparse row (CSR) adjacency list format. This consists of an array of row pointers of length number of vertices (G.begin) and an array of column indices of length number of vertices (G.node_idx). Additionally, Green-Marl creates a set of reverse graph CSR arrays, which are used in BFS for the bottom-up portion of the direction-optimizing algorithm, in BC for the reverse BFS, and for traversing in-neighbors in PR and HITS.

While we look at the locality of each iteration in isolation, there is of course opportunity for reuse between iterations. Particularly, there is no reuse of the G.node_idx data structure within an iteration. Due to the immense size of this array, today's caches will not be able to maintain its full state, let alone the other data structures needed during execution, within the cache across iterations. The G.node_idx array of our largest



(a) Locality score as a 3D surface



(b) Locality score as a 2D contour

Fig. 8: Locality score as a function of near future window size and cache block size for the PageRank algorithm on the Livej dataset.

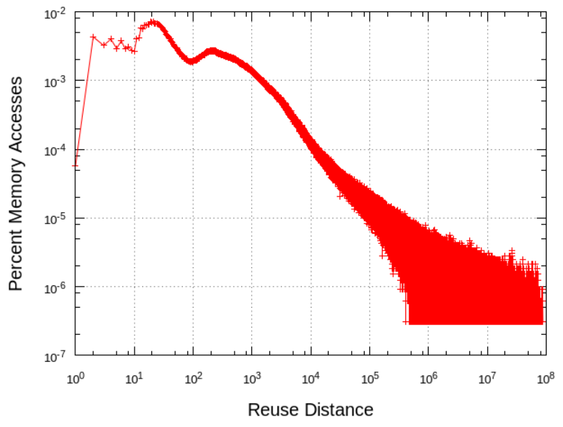


Fig. 9: Reuse distance PDF of PageRank values array for Livej dataset.

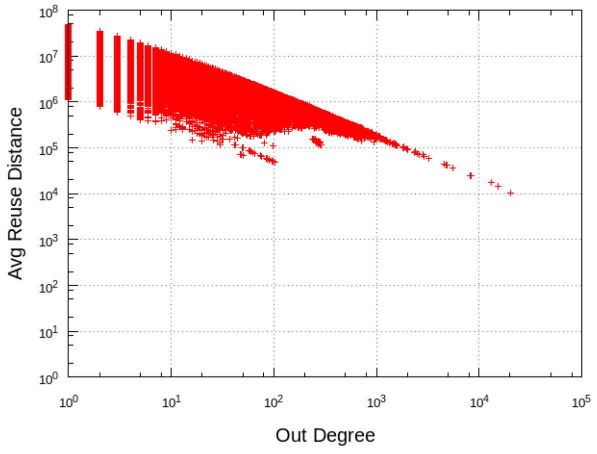


Fig. 11: Average reuse distance relative to vertex out-degree for the PageRank data structure for the PageRank algorithm on the Livej dataset.

dataset, Wiki, is 500 MB, far too large even for the latest on-chip LLCs, which are less than 100 MB.

If column indices were not cached, there would be potential to reuse other data structures between iterations, however these data structures are still tens of megabytes. For our example of Wiki, an array of vertex properties of size `double` (8 bytes), such as the PageRank values, is 116 MB.

Clearly, simply selecting data structures to avoid caching, such as cache hints based on particular instructions (proposed in [13]), will not be sufficient to avoid thrashing, or evicting data with potential for reuse.

We find that there is a correlation between vertex degree and reuse distance of that vertex's properties, which is most pronounced for scale-free graphs because of their large range of degree values. Figure 11 shows the relationship between average reuse distance and vertex out-degree for the PageRank data structure. We expect a correlation with out-degree because for all of the in-neighbors of each vertex, one data element is read from the PageRank values array. The more out-neighbors a vertex has (high out-degree), the more likely it is to be an in-neighbor of any vertex, thus the expected correlation.

VI. Related Work

We focus our study of related work on two main areas: characterization of graph algorithms on CPU and GPU platforms, and data locality analysis as it relates to graph algorithms.

Two recent studies characterize parallel graph algorithms on CPU. Beamer et al. [14] identify hardware factors that prevent full utilization of memory bandwidth by graph algorithms. They claim that graph algorithms have 'substantial locality' and 'moderately high' cache hit rates, contrary to most other literature. The authors also find that input graph properties affect performance, which we confirm.

The CRONO graph benchmark paper [13] characterizes a suite of graph algorithms using simulation. They find high data sharing and network traffic as the primary bottlenecks. The authors state that graph algorithms have 'low locality,' but their results show low cache miss rates and they conclude that memory bandwidth does not limit scaling. We hypothesize that small input graphs (most with less than 3 million edges) paired with large caches (65 MB total L2) result in the observed low cache miss rates and memory bandwidth utilization.

Two earlier studies discuss graph algorithm bottlenecks. Hendrickson et al. [16] and Lumsdaine et al. [15] state the need for parallel graph processing due to large data size, and agree that graph algorithms have a high data access-to-computation ratio. Both argue for massively multithreaded systems due to poor locality. Yang and Chien [49] find a 1000-fold variation in graph behaviors across different scales of algorithms and graph structures on a cluster of CPU machines.

Several works analyze the behavior of graph algorithms on GPU. Burtscher et al. [50] study irregular algorithms, including graph algorithms, on GPU using hardware performance counters and quantify control-flow and memory access regularity. They find that irregularity in graph applications is sensitive to graph type. A follow-up work by O'Neil et al. [51] further investigates algorithm behavior in simulation. Compared to regular algorithms, graph algorithms exhibit lower IPC, higher L1 MPKI, and more stalls due to uncoalesced memory accesses. Notably, graph applications were more sensitive to cache than DRAM bandwidth and latency.

Via GPU performance counters and simulation Xu et al. [52] identify the main graph algorithm bottlenecks as synchronization with the CPU and memory latency. For the Pannotia GPU graph benchmark suite, Che et al. [53] identify the main performance challenges as branch and memory divergence, load imbalance, and variation in parallelism over the execution. They find diverse behaviors across workloads. Wu et al. [54] analyze graph algorithm primitives on GPU across programming models. They conclude that graph topology significantly affects performance; particularly, graphs with a larger traversal frontier (small diameter) and vertex degree perform better.

We examine work on modeling and analysis of temporal and spatial locality as it applies to graph algorithms. The concept of reuse (or LRU stack) distance was first introduced by Mattson et al. [44]. Bunt and Murphy [55] investigate locality measurements that are independent of machine configuration.

Jiang et al. [56] propose concurrent reuse distance, which extends reuse distance to multithreaded workloads on shared-memory multicore machines. Wu et al. [57] apply concurrent and private reuse distance to study the scalability of multicore cache hierarchies.

Gu et al. [58] propose a model for spatial locality based on the change in reuse distance as a function of data block size. Gupta et al. [45] quantify spatial and temporal locality as conditional probabilities and visualize locality as a function of near future window size and neighborhood size. Weinberg et al. [20] propose a methodology for characterizing spatial and temporal locality and apply it to analyze benchmarks for HPC applications. Yuan et al. [59] apply locality modeling to graph traversal and introduce vertex distance, with which they predict BFS reuse distance and number of BFS levels for synthetic graphs.

A common theme across the literature is the diversity of behavior across graph algorithms and datasets. To capture variation in behavior, we characterize diverse algorithms and datasets. The variability in cache miss rates indicates that they can be unreliable for characterization due to dependence on input graph and cache sizes. While previous work has presented cache miss rates and reuse distance on synthetic graphs, this paper provides a quantitative analysis of locality and its sources on large, real-world graphs.

VII. Conclusions

This paper presents a characterization of the data locality of several parallel graph analysis algorithms on real-world datasets. Our analysis is based on comprehensive experiments from architectural simulation of shared-memory multicore systems. We find that data locality is impacted more by the algorithm than by the structure of the graph dataset. We find that graph algorithm data reuse patterns correlate with graph dataset structure and vertex degree, but cannot be effectively exploited by LRU cache replacement policies. These insights can be leveraged to improve graph analysis algorithm performance.

References

[1] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pp. 611–617, 2006.

[2] P. Novák, P. Neumann, and J. Macas, "Graph-based clustering and characterization of repetitive sequences in next-generation sequencing data," *BMC bioinformatics*, vol. 11, no. 1, p. 378, 2010.

[3] M. B. Gerstein, A. Kundaje, M. Hariharan, S. G. Landt, K.-K. Yan, C. Cheng, X. J. Mu, E. Khurana, J. Rozowsky, R. Alexander, *et al.*, "Architecture of the human regulatory network derived from encode data," *Nature*, vol. 489, no. 7414, pp. 91–100, 2012.

[4] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," *Computer networks*, vol. 33, no. 1, pp. 309–320, 2000.

[5] 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 '10*, pp. 135–146, ACM, 2010.

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

[7] 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," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

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

[9] K. Zhang, R. Chen, and H. Chen, "Numa-aware graph-structured analytics," in *ACM SIGPLAN Notices*, vol. 50, pp. 183–193, ACM, 2015.

[10] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, pp. 135–146, 2013.

[11] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: Edge-centric graph processing using streaming partitions," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 472–488, 2013.

[12] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun, "Green-marl: a DSL for easy and efficient graph analysis," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pp. 349–362, 2012.

[13] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, pp. 44–55, 2015.

[14] S. Beamer, K. Asanović, and D. Patterson, "Locality exists in graph processing: Workload characterization on an ivy bridge server," in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, pp. 56–65, 2015.

[15] A. Lumsdaine, D. Gregor, B. Hendrickson, J. Berry, and J. Guest Editors, "Challenges in parallel graph processing," *Parallel Processing Letters*, vol. 17, no. 1, pp. 5–20, 2007.

[16] B. Hendrickson and J. Berry, "Graph analysis with high-performance computing," *Computing in Science Engineering*, vol. 10, no. 2, pp. 14–19, 2008.

[17] D. Watts and S. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, 1998.

[18] A. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[19] C. Ding and Y. Zhong, "Reuse distance analysis," tech. rep., TR 741, Department of Computer Science, University of Rochester, 2001.

[20] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snavely, "Quantifying locality in the memory access patterns of hpc applications," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pp. 50–50, IEEE, 2005.

[21] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the flickr social network," in *Proceedings of the 1st ACM SIGCOMM Workshop on Social Networks (WOSN'08)*, August 2008.

[22] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: membership, growth, and evolution," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 44–54, 2006.

[23] 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.

[24] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, pp. 3:1–3:8, 2012.

[25] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "DBpedia: A nucleus for a web of open data," in *Proc. Int. Semantic Web Conf.*, pp. 722–735, 2008. the DBpedia dataset.

[26] D. A. Bader, H. Meyerhenke, P. Sanders, C. Schulz, A. Kappes, and D. Wagner, "Benchmarking for graph clustering and partitioning," in *Encyclopedia of Social Network Analysis and Mining*, pp. 73–82, Springer, 2014.

[27] K. Madduri and D. A. Bader, "Gtgraph: A suite of synthetic random graph generators." http://www.cse.psu.edu/~madduri/software/GTgraph/. Accessed: 2014-04-19.

[28] S. Hong, T. Oguntebi, and K. Olukotun, "Efficient parallel graph exploration for multi-core cpu and gpu," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, 2011.

[29] S. Beamer, K. Asanović, and D. Patterson, "Direction-optimizing breadth-first search," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pp. 1–10, 2012.

[30] D. Prountzos and K. Pingali, "Betweenness centrality: algorithms and implementations," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, pp. 35–46, 2013.

[31] U. Brandes, "A faster algorithm for betweenness centrality," *The Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[32] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: bringing order to the web.," tech. rep., 1999.

[33] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999.

[34] R. Lempel and S. Moran, "Salsa: the stochastic approach for link-structure analysis," *ACM Transactions on Information Systems (TOIS)*, vol. 19, no. 2, pp. 131–160, 2001.

[35] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh, "WTF: The who to follow service at twitter," in *Proceedings of the 22nd international conference on World Wide Web*, pp. 505–514, International World Wide Web Conferences Steering Committee, 2013.

[36] F. Chierichetti, S. Lattanzi, and A. Panconesi, "Rumour spreading and graph conductance.," in *SODA*, pp. 1657–1663, SIAM, 2010.

[37] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data. Accessed: 2014-06-01.

[38] "Koblenz network collection." http://konect.uni-koblenz.de. Accessed: 2014-03-07.

[39] "10th DIMACS implementation challenge." http://www.cc.gatech.edu/dimacs10/downloads.shtml. Accessed: 2014-03-07.

[40] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining.," in *SDM*, vol. 4, pp. 442–446, SIAM, 2004.

[41] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Computer Architecture (ISCA), 2013 40th Annual International Symposium on*, 2013.

[42] D. Molka, D. Hackenberg, R. Schone, and M. S. Muller, "Memory performance and cache coherency effects on an intel nehalem multiprocessor system," in *Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on*, pp. 261–270, 2009.

[43] D. Sanchez, "Github: zsim." http://github.com/s5z/zsim, jun 2015.

[44] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems journal*, vol. 9, no. 2, pp. 78–117, 1970.

[45] S. Gupta, P. Xiang, Y. Yang, and H. Zhou, "Locality principle revisited: A probability-based quantitative approach," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 995–1009, 2012.

[46] B. T. Bennett and V. J. Kruskal, "Lru stack processing," *IBM Journal of Research and Development*, vol. 19, no. 4, pp. 353–357, 1975.

[47] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *ACM SIGARCH Computer Architecture News*, vol. 35, pp. 381–391, ACM, 2007.

[48] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *ACM SIGARCH Computer Architecture News*, vol. 38, pp. 60–71, ACM, 2010.

[49] F. Yang and A. A. Chien, "Understanding graph computation behavior to enable robust benchmarking," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 173–178, ACM, 2015.

[50] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on gpus," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, pp. 141–151, 2012.

[51] M. A. O'Neil and M. Burtscher, "Microarchitectural performance characterization of irregular gpu kernels," in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pp. 130–139, Oct 2014.

[52] Q. Xu, H. Jeon, and M. Annavaram, "Graph processing on gpus: Where are the bottlenecks?," in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pp. 140–149, Oct 2014.

[53] S. Che, B. Beckmann, S. Reinhardt, and K. Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *Workload Characterization (IISWC), 2013 IEEE International Symposium on*, pp. 185–195, Sept 2013.

[54] Y. Wu, Y. Wang, Y. Pan, C. Yang, and J. D. Owens, "Performance characterization of high-level programming models for gpu graph analytics," in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, pp. 66–75, 2015.

[55] R. B. Bunt and J. M. Murphy, "The measurement of locality and the behaviour of programs," *The computer journal*, vol. 27, no. 3, pp. 238–245, 1984.

[56] Y. Jiang, E. Z. Zhang, K. Tian, and X. Shen, "Is reuse distance applicable to data locality analysis on chip multiprocessors?," in *Compiler Construction*, pp. 264–282, Springer, 2010.

[57] M.-J. Wu, M. Zhao, and D. Yeung, "Studying multicore processor scaling via reuse distance analysis," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pp. 499–510, 2013.

[58] X. Gu, I. Christopher, T. Bai, C. Zhang, and C. Ding, "A component model of spatial locality," in *Proceedings of the 2009 International Symposium on Memory Management*, ISSM '09, pp. 99–108, 2009.

[59] L. Yuan, C. Ding, D. Sefankovic, and Y. Zhang, "Modeling the locality in graph traversals," in *Parallel Processing (ICPP), 2012 41st International Conference on*, pp. 138–147, 2012.