SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems

Maciej Besta[1], Raghavendra Kanakagiri[2], Grzegorz Kwasniewski[1], Rachata Ausavarungnirun[3], Jakub Beránek[4], Konstantinos Kanellopoulos[1], Kacper Janda[5], Zur Vonarburg-Shmaria[1], Lukas Gianinazzi[1], Ioana Stefan[1], Juan Gómez Luna[1], Marcin Copik[1], Lukas Kapp-Schwoerer[1], Salvatore Di Girolamo[1], Marek Konieczny[5], Onur Mutlu[1], Torsten Hoefler[1]

[1] *ETH Zurich* [2] *IIT Tirupati* [3] *King Mongkut's University of Technology North Bangkok*
[4] *Technical University of Ostrava* [5] *AGH-UST*

arXiv:2104.07582v1 [cs.AR] 15 Apr 2021

ABSTRACT

Simple graph algorithms such as PageRank have recently been the target of numerous hardware accelerators. Yet, there also exist much more complex graph *mining* algorithms for problems such as clustering or maximal clique listing. These algorithms are memory-bound and thus could be accelerated by hardware techniques such as Processing-in-Memory (PIM). However, they also come with non-straightforward parallelism and complicated memory access patterns. In this work, we address this with a simple yet surprisingly powerful observation: operations on sets of vertices, such as intersection or union, form a large part of many complex graph mining algorithms, and can offer rich and simple parallelism at multiple levels. This observation drives our cross-layer design, in which we (1) expose set operations using a novel programming paradigm, (2) express and execute these operations efficiently with carefully designed *set-centric* ISA extensions called SISA, and (3) use PIM to accelerate SISA instructions. The key design idea is to alleviate the bandwidth needs of SISA instructions by mapping set operations to two types of PIM: in-DRAM bulk bitwise computing for bitvectors representing high-degree vertices, and near-memory logic layers for integer arrays representing low-degree vertices. Set-centric SISA-enhanced algorithms are efficient and outperform hand-tuned baselines, offering more than 10× speedup over the established Bron-Kerbosch algorithm for listing maximal cliques. We deliver more than 10 SISA set-centric algorithm formulations, illustrating SISA's wide applicability.

1. INTRODUCTION

Research in graph analytics in the architecture community has mostly targeted graph algorithms based on vertex-centric formulations [5, 6, 13, 26, 73, 93, 123, 131, 154, 196]. Some works also focus on edge-centric or linear algebra paradigms [97, 145, 162, 164]. Such algorithms have complexities described by *low-degree* polynomials [98], for example Breadth-First Search (BFS) [47] ($O(n+m)$) or iteration-based PageRank (PR) [25] ($O(m \cdot \#\text{iterations})$), where n and m are numbers of vertices and edges, respectively.

Yet, there are numerous important problems and algorithms in the area of ***graph mining*** [28, 45, 89, 148, 170] that received little or no attention in the architecture community. One large class is ***graph pattern matching*** [89], which focuses on finding certain specific subgraphs (also called motifs or graphlets). Examples of such problems are k-clique listing [53], maximal clique listing [33, 36, 62, 172], k-star-clique mining [84], and many others [45]. Another class is broadly referred to as ***graph learning*** [45], with problems such as unsupervised learning or clustering [86], link prediction [8, 109, 113, 168], or vertex similarity [104]. All these problems are used in social sciences [62], bioinformatics [62], computational chemistry [166], medicine [166], cybersecurity [59], healthcare [171], web graph analysis [90], and many others [37, 45, 80, 89]. These problems often run in time at least quadratic in the number of vertices, and many problems are NP-complete [33, 45, 53, 173]. Thus, they often differ significantly in their performance properties from "low-complexity" problems such as BFS or PageRank.

Importantly, the established vertex-centric model, originally proposed in the Pregel graph processing system [117], is *not* the right tool for expressing graph mining problems. This paradigm exposes only the local graph structure: A thread executing a vertex kernel for any vertex v can only access the neighbors of v. While this suffices for algorithms such as PageRank, graph mining often requires non-local knowledge of the graph structure [45]. Obtaining such knowledge in the vertex-centric paradigm is hard or infeasible, as noted by Kalavri et al. [93] (*"(...) graph algorithms, like triangle counting, are not a good fit for the vertex-centric model"*) and many others [100, 110, 147, 187]. Similar arguments apply to other paradigms such as GraphBLAS [97, 145] and to frameworks such as Ligra [157]. None of them supports a wide selection of graph mining problems (e.g., GraphBLAS only enables subgraph isomorphism assuming patterns are trees); we show it in detail in Table 1 and Section 4.

Several graph mining software frameworks (Peregrine [85] and others [40,41,58,83,91,121,122,170,186,188,198]) have been proposed. Yet, they focus exclusively on *a few* graph pattern matching problems. Moreover, these frameworks usually do *not* offer theoretical guarantees (unlike parallel graph algorithms for specific mining problems). Overall, there is a need for a graph mining paradigm that would enable expressing many graph mining problems, and ideally offer competitive theoretical guarantees on their runtimes.

Moreover, past works illustrated that graph mining algorithms are memory bound [42, 60, 85, 193, 197]. This is because these algorithms generate and heavily use large intermediate structures, but – similarly to algorithms such as PageRank – they are not compute-heavy [62, 85, 194]. We show this in Figure 1. When increasing the number of parallel threads,

Abstraction or programming model	A?	Pattern M.				Learning				“Low-c.”				Remarks
		mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)	✖	✖	✖	✖	✖	✖	✖	✖	✖	▄*	■	▄	■	*High comm. costs
Edge-centric (edge-c)	✖	✖	✖	✖	✖	✖	✖	✖	✖	▄*	▄*	▄*	■	*High work and depth
Array maps	✖	✖	✖	■	✖	✖	✖	▄*	✖	▄	■	■	■	*Only low-diameter decomposition
GraphBLAS [97]	■ (L)	✖	✖	✖	▄*	✖	✖	✖	✖	■	■	▄†	■	*Only trees as patterns
GNN, GCN	▄ (L)	✖	✖	✖	▄†	▄	■	■	■	✖	✖	✖	▄	†GNNs are as powerful as the Weisfeiler-Lehman test [185].
Pattern matching	✖	▄*	▄*	▄*	▄*	✖	✖	✖	✖	▄*	✖	✖	✖	*No bounds, low perf.
Joins [43]	■ (R)	✖	▄*	▄*	✖	▄*	▄*	▄*	✖	▄*	✖	▄*	▄*	*No bounds, low perf.
Set-Centric / SISA	■ (S)	■	■	■	■	■	■	■	■	■	✖	✖	✖	

Table 1: **Comparison of the set-centric programming approach and SISA to existing graph processing abstractions/programming models, focusing on support for selected graph mining problems (pattern matching, learning), and for “low-complexity” graph problems**. **A?**: Underlying algebra? **L**: linear, **R**: relational, **S**: set. “■”: Support / significant focus. “▄”: Partial support / some focus. “✖”: no support / no focus. **Pattern M.**: selected graph pattern matching problems, **mc**: maximal clique listing, **kc**: k-clique listing, **ds**: dense subgraph, **si**: subgraph isomorphism, **Learning**: selected graph learning problems, **vs**: vertex similarity, **lp**: link prediction, **cl**: clustering or community detection, **av**: accuracy verification (of link prediction outcomes), **“Low-c.”**: selected “low-complexity” problems targeted by vast majority of existing works on graph processing. **tc**: triangle counting, **bf**: BFS, **cc**: connected components, **pr**: PageRank. **The analysis in this table is extended in Section 10 and Table 7 by detailing specific hardware accelerators for graph processing.**

speedups are decreasing and the counts of stalled CPU cycles increase. This motivates using processing-in-memory (PIM) to gain the much needed speedups in graph mining. While PIM is not the only potential solution for hardware acceleration of graph mining, we select PIM because (1) it represents one of the most promising trends to tackle the memory bottleneck [69, 128] outperforming other approaches [153], (2) it offers well-understood designs [129], and (3) numerous works illustrate it brings very large speedups in *simple* graph algorithms such as BFS or PageRank (see more than 15 works in Table 7), also using processing fully inside DRAM [10]. Yet, graph mining algorithms are *much more complex*: they employ deep recursion, create many intermediate data structures with non-trivial inter-dependencies, and have high load imbalance [62, 186]. As we show in Section 10, *no existing HW design targets broad graph mining (i.e., both graph pattern matching and graph learning), or explores PIM techniques for accelerating broad graph mining*.

To address all these issues, we propose a novel design that is high-performance (both empirically and theoretically), applicable to many graph mining problems, and easily amenable to PIM acceleration. We first observe that large parts of many graph mining algorithms can be expressed with simple set operations such as intersection ∩ or union ∪, where sets contain vertices or edges. Driven by this observation, we propose a **set-centric programming paradigm**, in which the developer identifies sets and set operations in a given graph mining algorithm. Then, these operations are mapped to a small and simple yet expressive group of instructions that implement many set operations and their variants, offering a rich selection of storage/performance tradeoffs. Finally,

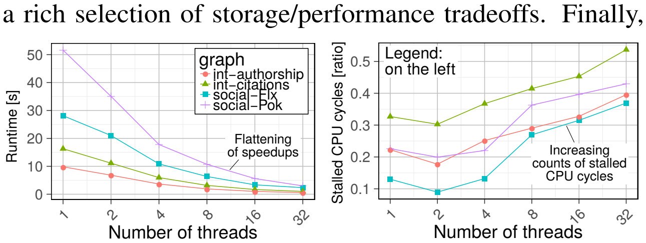


Figure 1: **Speedups and counts of stalled CPU cycles** when increasing parallelism, for the Bron-Kerbosch algorithm for listing maximal cliques, for different input graphs (evaluation methodology is discussed in Section 9).

these instructions are offloaded to PIM units. We call these instructions ***SISA*** as they form a “Set-centric” ISA extension that enables a simple interface between numerous graph mining algorithms and PIM hardware. Overall, our cross-layer design consists of three key elements: a new set-centric programming paradigm and formulations of graph algorithms (contribution **#1**), the actual set-centric ISA extension with its instructions, implemented set operations, set organization, and a thin software layer (contribution **#2**), and PIM acceleration (contribution **#3**).

Using set algebra as a basis for algorithm design ensures that SISA set-centric algorithms are succinct, applicable to many problems, and theoretically efficient. Our set-centric paradigm is the first to use set operations as fundamental general building blocks for both algorithmic formulations *and* their execution. Next, when mapping set-centric formulations to the SISA code, one can use different set representations (e.g., a sparse integer array or a dense bitvector), and set operations such as intersection can be executed using different set algorithms (e.g., merge or galloping intersection [74]). These choices enable flexibility as they come with different performance/storage tradeoffs, which we analyze in detail.

For the in-memory acceleration of SISA, we investigate which types of PIM are beneficial for which set operations. We process sets stored as bitvectors using in-situ PIM [70], as offered in Ambit [153], ELP2IM [183], DRISA [107], or ComputeDRAM [65], for highest performance and energy efficiency (“**SISA processing using memory**” – **SISA-PUM**). In contrast, while sets stored as sparse arrays cannot be simply processed in situ with today’s technology, they can use the high throughput of near-memory PIM [112] as offered in the 2D UPMEM architecture [101] or logic layers of 3D DRAM such as Hybrid Memory Cube (HMC) [88] (“**SISA processing near memory**” – **SISA-PNM**). For even higher speedups, we provide a small HW controller that selects the best variant of a set instruction to be executed on-the-fly.

Overall, we show that graph mining algorithms, despite being complex and lacking straightforward parallelism (unlike PageRank and similar), benefit from PIM. Our key solution is using parallelism offered by set operations and exposed with the set-centric approach. This harnesses parallelism at the level of bits, DRAM subarrays, and vaults. We build upon recent HW developments and are the first to comprehensively implement graph *mining* algorithms with PIM. We integrate SISA with the RISC-V ISA [181] and we show that SISA-enhanced algorithms are theoretically efficient (contribution **#4**) *and* empirically outperform tuned parallel baselines (contribution **#5**), for example offering more than 10× speedup for many real-world graphs over the established Bron-Kerbosch algorithm for listing maximal cliques [62].

2. NOTATION AND BACKGROUND

We first describe background and notation, see Table 2.

Graphs We model an undirected graph G as a tuple (V,E); V and $E \subseteq V \times V$ are sets of vertices and edges; $|V| = n$, $|E| = m$. Vertices are modeled with integers $1,\dots,n$ and $V = \{1,\dots,n\}$. $N(v)$ denote the neighbors of $v \in V$; d and $d(v)$ denote G’s maximum degree and a degree of v.

Set Representations SISA heavily uses sets. Consider a set of k vertices $S = \{v_1,\dots,v_k\} \subseteq V$ (we focus on vertex sets, but SISA also works with edges). One can represent S as a

	Symbol	Description
Graphs	$G = (V, E)$	An undirected graph; V and E are sets of vertices and edges.
	n, m	The numbers of vertices and edges in G ($\|V\| = n, \|E\| = m$).
	$N(v), N^+(v)$	The neighbors and the out-neighbors of a vertex v.
	$d, d(v)$	The maximum degree of G, the degree of $v \in V$.
	c	The graph degeneracy (a property used in theoretical analysis).
	SA, DB	sparse array, dense bitvector
Architecture	l_M, b_M	The latency and bandwidth of accessing DRAM.
	b_L	The bandwidth of the interconnect (e.g., QPI) between cores.
	l_I	The latency of one bulk bitwise operation run with in-situ PIM.
	q	#rows that can be processed in parallel (e.g., in a DRAM bank).
	R	The size [bits] of a single DRAM row.
	SCU, SM	SISA Controller Unit, Set Metadata

Table 2: The most important symbols and acronyms.

simple contiguous **sparse array (SA)** with integers from S ("sparse" means that only non-zero elements are explicitly stored). SA's size is $W|S|$ [bits] where W is the memory word size (we assume that the maximum vertex ID fits in one word). One can also represent S with a **dense bitvector (DB)** of size n [bits]: the i-th set bit indicates that a vertex $i \in S$ ("dense" means that all zero bits are explicitly stored).

Set Operations SISA uses fundamental set operations: intersection $A \cap B$, union $A \cup B$, difference $A \setminus B$, cardinality $|A|$, and membership $\in A$. We use different algorithms to implement these operations (described later in the paper).

3. OVERVIEW & CROSS-LAYER DESIGN

We now overview SISA's cross-level design, see Figure 2.

(a) Set-Centric Formulations [Section 5 & 5.1] SISA relies on set-centric formulations of algorithms in graph mining. While some algorithms (e.g., Bron-Kerbosch [62]) by default use rich set notation, many others, such as k-clique listing by Danisch et al. [53], do not. In such cases, we develop such formulations. Details on deriving set-centric formulations are in Section 5.1; the key common step is to express two nested loops, commonly used to identify connections between two sets of vertices, with a single intersection of these sets.

A set can be represented in different ways, and a set operation can be executed using different set algorithms. A set-centric formulation hides these details, focusing on *what* a given graph algorithm does, and not *how* it is done.

(b.1) Set-Centric ISA (Instructions) [Section 6] Our ISA extension implements set operations. These instructions support all variants of operations, for example there is an instruction for both merge and galloping set intersection (details in Section 6). We also provide a thin software layer: iterators

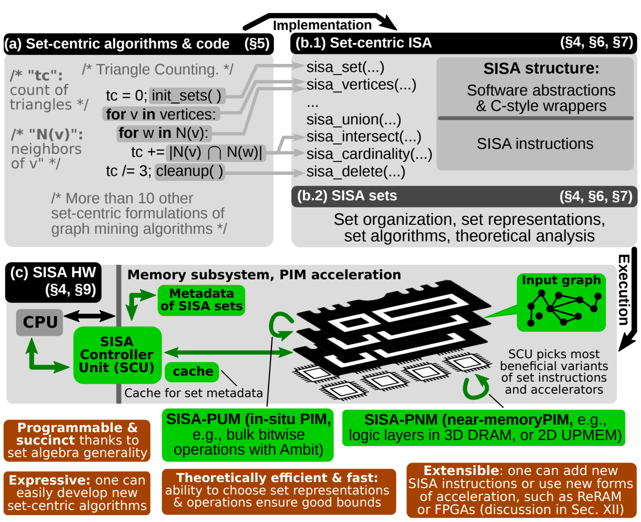


Figure 2: The overview of SISA with a summary of new introduced architecture and graph representation elements (green) and advantages (brown).

over sets and C-style wrappers for SISA instructions. For programmability and performance, many SISA instructions automatize selecting the best set operation variant on-the-fly.

(b.2) Set-Centric ISA (Organization of Sets) [Section 6] We represent sets as DBs or SAa. The former are processed by bulk bitwise in-situ PIM, harnessing huge internal DRAM bandwidth (SISA-PUM). The latter use near-memory PIM, for example DRAM cores in the UPMEM architecture, or logic layers in 3D stacked DRAM, harnessing the large through-silicon via (TSV) bandwidth (SISA-PNM).

(c) HW Implementation Details [Section 8] To maximize SISA's programmability and performance, we use hardware to automatically decide between SISA-PUM and SISA-PNM, or to pick a set algorithm variant (merge vs. galloping). For this, we use a dedicated unit called the SISA Controller Unit (SCU). The SCU can be an additional unit, or it can also be emulated by a process occupying a dedicated core in the logic layer, to avoid any HW modifications. The SCU receives SISA instructions from the CPU, and it appropriately schedules their execution on SISA-PNM and SISA-PUM. Two bitvectors are always processed with SISA-PUM, while in other scenarios SCU uses SISA-PNM. The SCU can also select the most advantageous set algorithm. For example, whenever two sets have similar sizes, it is better to intersect them using a merge-based intersection, in which input sets are streamed and they can harness high sequential bandwidth.

4. PROVABLY FAST GRAPH MINING

We first show that the set-centric approach is superior to existing graph programming paradigms as (1) it supports many graph mining problems and (2) it enables algorithms with theoretical guarantees on performance (e.g., work/depth [31]) that are competitive to those of tuned algorithms. Such provable guarantees are often key to low runtimes and scalability [56, 98]. The analysis results are in Table 1.

To illustrate the above points, we first extensively examined the related literature to identify representative **graph mining problems** and important **graph processing paradigms** [4, 9, 37, 64, 89, 103, 104, 109, 113, 138, 139, 141, 167, 178]. For the former, we pick four problems from both graph pattern matching and graph learning areas (maximal clique listing [33], k-clique listing [44], dense subgraph discovery [72, 103], subgraph isomorphism [173], vertex similarity [104, 142], link prediction [8, 109, 113, 168], graph clustering [86, 148], verification of prediction accuracy [177]). For fairness, we also consider four popular "low-complexity" problems, targeted by many past works (triangle counting, BFS, connected components, and PageRank). For the latter, we first select *vertex-centric* [117] and *edge-centric* [145], two established graph processing paradigms implemented in the Pregel and X-Stream systems. Second, we pick *vertex/edge array maps* from Ligra [157], an approach for developing graph algorithms based on transforming arrays of vertices or edges according to a specified map. Third, we consider *GraphBLAS* and its linear algebraic approach [97], where graph algorithms are expressed with linear algebra building blocks such as matrix-vector products. Moreover, we consider *pattern matching frameworks* [64] that usually employ some form of exploring neighbors of each vertex, combined with user-specified filtering, to search for specified graph patterns. For completeness, we also consider recent attempts at solving

Problem	Algorithm	Used set operations
Maximal clique list.	Bron-Kerbosch [62]	$A\cup B, A\cap B, A\setminus B$
k-clique listing	Danisch et al. [53] **+[This work]**	$A\cap B$
4-clique counting	**[This work]**	$A\cap B$, $\lvert A\cap B\rvert$
Triangle counting	[well-known]	$\lvert A\cap B\rvert$
k-clique-star listing	Jabbour et al. [84]	$A\cap B, A\cup B$
k-clique-star listing	**[This work]**	$A\cap B$
Subgr. isomorphism	**[This work]**	$A\cap B$, $\lvert A\cap B\rvert, A\cup B, A\setminus B$
Vertex similarity	Jaccard coeff., others [23, 142]	$\lvert A\cap B\rvert$, $\lvert A\cup B\rvert$
Clustering	Jarvis-Patrick [86]	$\lvert A\cap B\rvert$, $\lvert A\cup B\rvert$
Link prediction (LP)	Jaccard coeff., others [142]	$\lvert A\cap B\rvert$, $\lvert A\cup B\rvert$
LP accuracy testing	Wang et al. [177]	$A\setminus B$, $\lvert A\cap B\rvert$
Approx. degeneracy	Besta et al. [16]	$A\setminus B$

Table 3: **Overview of set-centric graph algorithms**. In maximal clique listing, subgraph isomorphism, and clustering, one also uses variants of union and difference where one set is always a single-element set (i.e., $A\cup\{b\}, A\setminus\{b\}$). Bolded text indicates algorithms with set-centric formulations derived in this work.

graph problems with *graph neural (GNN) and convolution (GCN) networks*, or general deep learning [15, 182], as well as *joins* and principles from relational databases and the associated algebra [199].

The analysis results are in Table 1. Overall, no single paradigm, except for the set-centric approach, enables efficient graph mining algorithms for the considered problems. Some paradigms, such as the vertex-centric or the edge-centric model, do not focus on such problems at all. Other paradigms, for example array maps or GNNs, address only certain problems. Finally, graph pattern matching or RDBMS can solve different graph mining problems, but they do not offer formal guarantees, as indicated by past work.

5. SET-CENTRIC GRAPH ALGORITHMS

We present set-centric formulations of graph mining algorithms. A list of algorithms and set operations used in each algorithm is in Table 3. Due to space constraints, we provide a few selected key formulations.

Notes on Listings Set operations accelerated by SISA are marked with the gray color. "[in par]" indicates that in a given loop one can issue set operations in parallel. We ensure that the parallelization does not involve conflicting memory accesses. We now focus on formulations and we discuss set representations, instructions, and parallelization later. For clarity, we exclude unrelated optimizations from the listings.

Maximal Cliques Listing [Pattern Matching] A clique is a fully-connected subgraph of an input graph; a maximal clique is a clique not contained in a larger clique. Finding all maximal cliques is an important NP-hard problem [54, 140, 163, 179]. Listing 1 shows the widely used recursive backtracking Bron-Kerbosch algorithm (BK) [33, 36, 62]. BK heavily uses different set operations. The main recursive function BKPivot (Line 5) has three arguments that are dynamic sets containing vertices. R is a partially constructed, non-maximal clique c, P are candidate vertices that may belong to c but are yet to be tried, and X are vertices that definitely do not belong to c. The algorithm recursively calls BKPivot for each new candidate vertex, checks if this gives a clique, and updates accordingly P and X. Some optimizations need more set operations, but they reduce the search space of potential cliques [172]. For example, the set of candidates (for extending a clique c) is $P\setminus N(u)$ instead of P, where $u\in P\cup X$.

k-Clique-Star Listing [Graph Pattern Matching] *k-clique-stars* are dense subgraphs that combine the characteristics of cliques *and stars*. A k-clique-star is a k-clique with additional neighboring vertices that are connected to all

```
1 /* Input: A graph G. Output: Maximal clique R (R ⊆ V).*/
2 P = V; R = ∅; X = ∅; //Init sets appropriately.
3 for v∈V [in par] do: BKPivot({v}, P, X);
4 function BKPivot(R, P, X):
5   if |P| == 0 and |X| == 0: return R; //Found a maximal clique
6   u = /* Choose a pivot vertex from P∪X */
7   for v∈ P\N(u) do: BKPivot( R∪{v} , P∩N(v) , X∩N(v) )
8     P = P\{v} ; X = X∪{v}
```

Algorithm 1: Maximal Clique Listing (Bron-Kerbosch) [33, 36].

the vertices in the clique. k-clique-stars were proposed as graph motifs that relax the restrictive nature of k-cliques [84]. Listing 2 shows our reformulated set-centric algorithm variant. Our observation is that those extra vertices that are connected to the k-clique actually form a $(k+1)$-clique (together with this k-clique). Thus, to find k-clique-stars, we first mine $(k+1)$-cliques. Then, we find k-clique-stars within each $(k+1)$-clique using set union, membership, and difference.

```
/* Input: A graph G. Output: All k-clique-stars, S.*/
C = /* First, find k-cliques (e.g., with Table 4)*/
S = ∅ //S is a set with identified k-clique-stars.
foreach c = (V_c, E_c) ∈ C do: //For each k-clique...
  X = ⋂_{u∈V_c} N(u) //Intersect all N(u) such that u∈V_c
  G_s = X∪V_c //Derive the actual k-clique-star
  S = S∪{G_s} //Add an identified k-clique-star to S
//At the end, remove duplicates from S
```

Algorithm 2: k-clique-star listing [84].

Vertex Similarity & Clustering [Graph Learning] Various measures assess how similar two vertices v and u are, see Listing 3. They can be used on their own, or as a main building block of more complex algorithms such as clustering. In clustering, one iterates over all adjacent vertex pairs, and uses their similarity to decide if the pair belongs to a cluster.

```
/* Input: A graph G. Output: Similarity S∈ℝ of neighborhoods
 * N(u) and N(v) of some vertices u and v. */
S_J(v,u) = |N(v)∩N(u)| / |N(v)∪N(u)| /* Jaccard Similarity */
S_O(v,u) = |N(v)∩N(u)| / min(|N(v)|, |N(u)|) //Overlap Similarity
```

Algorithm 3: Vertex similarity measures.

"Low-Complexity" Graph Algorithms: Discussion SISA does not target the "low-complexity" algorithms such as PageRank, as these algorithms offer few straightforward opportunities for set-centric acceleration. For example, in PageRank, one iterates over two nested loops, and updates vertex ranks, which is not easily expressible with set operations. We analyzed many other such algorithms, including Dijkstra's SSSP [160], Δ–Stepping [124], Bellman-Ford [47], Betweenness Centrality schemes [161], traversals [24], Connected Components algorithms [71, 156, 165, 187], Low-Diameter Decomposition [125], or Boruvka's Minimum Spanning Tree [32]. Some of them use set notation, but *these are not "PIM-friendly" operations such as bulk set intersections of vertex sets, and are thus not easily accelerated with SISA (we list such operations in Table 3).* While this may be possible, we leave this direction for future work.

5.1 Deriving a Set-Centric Formulation

Often, algorithms use set notation, and one may simply pick operations for memory acceleration. This is the case with, for example, Jarvis-Patrick clustering (Section 5). Yet, sometimes one may need to apply more complex changes to "expose" set instructions. The general rule is to associate used data structures with sets, and then identify respective set operations. As an example, we compare a traditional

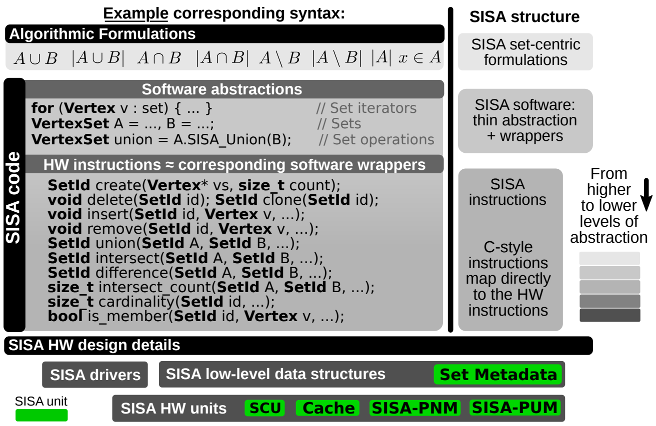


Figure 3: Overview of SISA instructions and syntax at different levels of abstraction.

snippet for deriving the count of all 4-cliques `cnt`, a derived set-centric algorithmic formulation, and the corresponding SISA snippet in Table 4. The key algorithmic change is using set intersections instead of explicitly verifying if vertices are connected. For example, instead of iterating over all neighbors of v_1-v_3 (Lines 4-6, the top snippet), in SISA, we intersect neighborhoods of v_1-v_3 (Line 4 & 6, the middle snippet) to filter 4-cliques.

```
1 //Non set-centric code:
2 CSR_Graph g(G); //Standard codes often use some form of CSR
3 #pragma omp parallel for
4 for (auto v1: g.V()) //For all vertices in parallel.
5  for (auto v2: g.N_out(v1)) //Explore neighborhoods of v1-v4...
6   for (auto v3: g.N_out(v2)) //...searching for a 4-clique
7    for (auto v4: g.N_out(v3)) //If v1-v4 are connected pairwise
8      if(g.edge(v1,v3) && g.edge(v1,v4) && g.edge(v2,v4)) ++cnt;
```

```
1 //A set-centric algorithmic formulation:
2 for v1 ∈ V in parallel do: //For all vertices in parallel.
3   for v2 ∈ N+(v1) do: //For each neighbor of v1...
4     S1 = N+(v1)∩N+(v2) //Find common neighbors of v1 and v2.
5     for v3 ∈ S1 do: //Narrow further search to S1.
6       cnt += |S1∩N+(v3)| //Common neighbors of v1, v2, and v3
```

```
1 //SISA (simplified) set-centric code:
2 SetGraph g = SetGraph(G);
3 #pragma omp parallel for
4 for (auto v1: g.V()) for (auto v2: g.N_out(v1)) {
5     auto S1 = intersect(g.N_out(v1), g.N_out(v2));
6     for (auto v3: S1) cnt += intersect_card(S1, g.N_out(v3)); }
```

Table 4: Listing all 4-cliques: a traditional (non-set-centric) snippet, a set-centric algorithmic formulation derived in this work, and a SISA set-centric snippet.

6. SISA: DESIGN, SYNTAX, SEMANTICS

We now present the details of representing and processing sets used in set-centric formulations. This constitutes core parts of SISA's design. We summarize SISA in Figure 3 and we detail key SISA instructions in Table 5.

6.1 Representation of Sets

The first key question is how to represent sets: SISA's "first-class citizens". We observe that – in each graph algorithm – there are two fundamentally different classes of data structures. One class are **(1) vertex neighborhoods** $N(v)$ that maintain the structure of the input graph. There are n such sets, their total size is $O(m)$, and each single neighborhood is static (we currently focus on static graphs) and sorted (following the established practice in graph processing [118]). Another class are **(2) auxiliary structures**, for example P in Bron-Kerbosch (Listing 1). These sets are used to maintain some algorithmic state. They are usually dynamic, they may

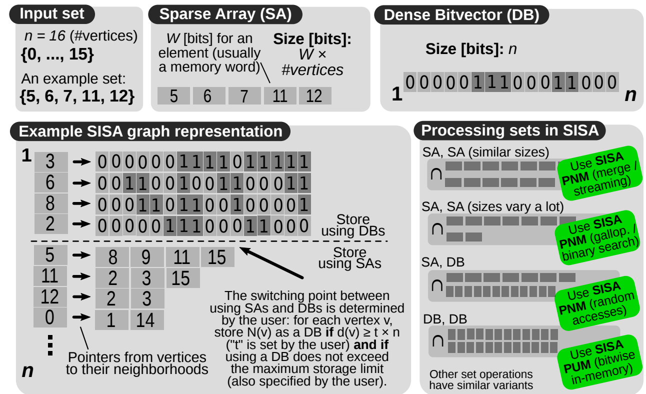


Figure 4: SISA representations of sets and graphs, and processing SISA sets.

be unsorted, their number (in a given algorithm) is usually a (small) constant, and their total size is $O(n)$. While SISA enables using any set representation for any specific set, we offer certain recommendations to maximize performance.

SAs should be used for small neighborhoods and DBs for the large ones (in the evaluation, we vary the threshold so that 5%-30% largest neighborhoods use DBs). This approach is memory efficient. For example, for $|N(v)| = n/2$, a DB takes only n bits while an SA uses $16n$ bits (for a 32-bit word size).

Auxiliary sets benefit from being stored as dense bitvectors. This is because such sets are often dynamic, and updates or removals take $O(1)$ time. Simultaneously, as in practice there is usually a small constant number of such sets in considered algorithms, the needed storage is not excessive (e.g., less than 3% of the total storage needed for a graph with the average degree 100 (such as orkut), assuming using 32 threads and the Bron-Kerbosch algorithm, with auxiliary sets P, X, and R). We analyze and confirm it for other algorithms and datasets.

The user controls selecting a set representation. For programmability, SISA offers a predefined graph structure, where small and large neighborhoods are **automatically** stored as sparse arrays and dense bitvectors, respectively. This is guided by a simple model: a given neighborhood $N(v)$ is stored as a DB whenever $|N(v)| \geq t \cdot n$ ($t \in (0;1)$ is a user parameter that controls a "bias" towards using DBs or SAs) and it does not exceed a storage budget limit set by the user (SISA by default uses a limit of 10% of the additional storage on top of the graph size when stored only with SAs). For example, $t = 0.5$ indicates that each vertex connected to at least 50% of all vertices has its neighborhood stored as a DB.

Figure 4 shows an SA and a DB built from the same vertex set. Then, it illustrates an example SISA graph representation where some neighborhoods are DBs and some are SAs.

6.2 High-Performance Set Operations

The second key challenge in SISA is how to apply set

ins	Set op.	A and B represent.	Set algorithm	S?	Time complexity	Input size [bits]	Main form of data transfer (§ 8.3)
`0x0`	$A \cap B$	SA ∩ SA	Merge	👍, 👍	$O(\|A\| + \|B\|)$	$W\|A\| + W\|B\|$	Streaming
`0x1`	$A \cap B$	SA ∩ SA	Galloping	👍, 👍	$O(\|A\| \log \|B\|)$	$W\|A\| + W\|B\|$	Random accesses
`0x2`	$A \cap B$	SA ∩ SA	Merge / gallop.	👍, 👍	cf. `0x0` and `0x1`	$W\|A\| + W\|B\|$	cf. `0x0` and `0x1`
`0x3`	$A \cap B$	SA ∩ DB	Galloping	👍, na	$O(\|A\|)$	$W\|A\| + n$	Random accesses
`0x4`	$A \cap B$	DB ∩ DB	Bitwise AND	na, na	$O(n/(qS))$	$n + n$	In-situ row copies
`0x5`	$A \cup \{x\}$	DB ∪ $\{x\}$	Set bit	na, na	$O(1)$	$n + W$	Random access
`0x6`	$A \setminus \{x\}$	DB \ $\{x\}$	Clear bit	na, na	$O(1)$	$n + W$	Random access

Table 5: **Overview of SISA instructions**, one row describes one specific set operation variant. Set elements are vertices ($A, B \subseteq V, x \in V$). "👍" means "yes". **"na"** means "not applicable". **"ins"** is a proposed instruction opcode. **"S (Sorted)"** indicates if an instruction assumes set representations of A and B to be sorted (thus two columns).

operations for highest performance. For this, we detail the algorithmic aspects, a summary is in Table 5. HW details (used PIM and a performance model) are discussed in Section 8. An overview of the structure of SISA is in Figure 3.

Set Intersection $A \cap B$ is a key operation in SISA, because our analysis illustrates that it is used in essentially all considered graph algorithms. We now briefly discuss the most relevant variants of $\cap$, a summary is in Figure 4.

- **SA [sorted]** $A \cap$ **SA [sorted]** B The intersection of two sorted SAs is commonly used when processing two neighborhoods. It comes in two "flavors". If A and B have similar sizes ($|A| \approx |B|$), one prefers the **merge** scheme where one simply iterates through A and B, identifying common elements (time $O(|A|+|B|)$). If one set is much smaller than the other ($|A| \ll |B|$), it is better to use the **galloping** scheme [1], in which one iterates over the elements of a smaller set and uses a binary search to check if each element is in the bigger set (time $O(|A| \log |B|)$). SISA offers both variants, and a variant that automatically selects the best variant with a performance model (described in § 8.3).
- **SA [unsorted or sorted]** $A \cap$ **DB** B Iterate over A ($O(|A|)$) and check if each element is in B ($O(1)$). This variant is often used to intersect a neighborhood with an auxiliary set represented as a bitvector, for example $X \cap N(v)$ in Listing 1.
- **DB** $A \cap$ **DB** B Apply bitwise AND over both input DBs (they both have sizes of n bits, giving $O(n/C)$ time, where C is the maximum chunk of bits that can be processed in $O(1)$ time using bit-level parallelism). This variant is used for example when intersecting two dense neighborhoods.

Set Union $A \cup B$ and Set Difference $A \setminus B$ $A \setminus B$ and $A \cup B$ have variants similar to those for $\cap$, there are also corresponding merge and galloping variants.

Set Membership $x \in A$ and Set Cardinality $|A|$ Set membership takes $O(|A|)$ time for an unsorted SA (linear scan), $O(\log |A|)$ time for a sorted SA (binary search), and $O(1)$ for a DB (a single access to verify if x-th bit is set). As for set cardinality, we maintain this information for any set. This incurs only $O(1)$ storage overhead for any set as well $O(1)$ time overhead needed to update the size, but it enables $O(1)$ time to resolve any set cardinality operation. Finally, we note that SISA provides dedicated instructions for computing cardinalities of the results of set operations, for example $|A \cap B|$. This enables speedups as SISA avoids creating any intermediate structures needed for keeping the results of operations such as intersection.

Adding and Removing Elements Auxiliary sets often grow and shrink by one element. Both add and remove straightforwardly take $O(1)$ time for a DB (setting or zeroing a corresponding bit) and $O(|A|)$ for an SA (moving data in case the SA is sorted). Thus, in general, we advocate using DBs for auxiliary sets; the size is n bits.

6.3 Additional Details of SISA Design

We detail several aspects of SISA's design; cf. Figure 3.

SISA Instructions SISA offers instructions that package the described set operations in all the considered variants, including instructions that automatically select merge or galloping set algorithms (cf. § 6.2). Finally, SISA also provides instructions for creating and deleting sets.

Programming Interface (Set Iterators & Wrappers) For programmability, SISA offers a thin software layer on top of high-level instructions that consists of abstractions and wrappers. In the former, we provide an opaque type `Set` that is a reference to a SISA set; this enables using C++ iterators over sets, see left side of Figure 3. In the latter, SISA provides functions that directly map to SISA set instructions.

RISC-V Compliant Encoding SISA can be integrated with the RISC-V ISA [181]. To enable modularity and flexibility, SISA's new instructions are encoded using the custom opcode set [180]. We encode the opcode and functionality of custom RISC-V instructions using bits [31..25] and [6..0], see Figure 5. The former represent the different SISA instructions (up to 128). The latter are set to 0x16 to represent the custom characteristic of the instruction. Fields `rs1`, `rs2`, and `rd` indicate registers with IDs of input sets and the output set, respectively. In Table 5, we assign ISA codes (bits [31..25]) to respective instructions. The number of SISA instructions is less than 20, leaving space for potential new variants.

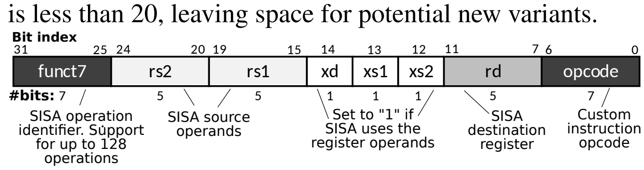


Figure 5: Encoding of SISA instructions.

7. THEORETICAL ANALYSIS

We now support our claim that SISA-enhanced algorithms are *theoretically efficient*, i.e., their time complexities match those of hand-tuned graph mining algorithms. In general, this is enabled by SISA's ability to control used set representations and set operations, facilitates tuning performance and storage tradeoffs. To show this, we analyze how varying a used set intersection variant (merge vs. galloping) impacts the runtime of set-centric algorithms. We focus on intersection as it is prevalent in considered algorithms. The analysis results are in Table 6 (proofs are in the report). Crucially, *all set-centric variants are able to match the competitive time complexities of considered tuned graph mining algorithms*.

We parametrize complexities with degeneracy c, a well-known measure of graph sparsity [120]. The degeneracy c of a graph G is the smallest number x such that every subgraph in G has a vertex of degree at most x (i.e., every subgraph has at least one sparsely connected vertex). We consider degeneracy as it is used by many recent graph mining algorithms to enhance their time complexities [53, 62, 206].

8. HARDWARE IMPLEMENTATION

We now discuss details of SISA hardware implementation.

8.1 Processing-In-Memory for Sets

We start with how SISA uses PIM for set operations.

SISA-PUM First, the intersection, union, and difference of sets represented as DBs are processed with SISA-PUM that relies on in-situ DRAM bulk bitwise schemes. For concreteness, we pick Ambit [153], a recent design that enables energy-efficient bulk bitwise operations fully inside DRAM, by small extensions to the DRAM circuitry but without any changes to the DRAM interface. However, SISA is generic and other designs could also be used (e.g., ELP2IM [183], DRISA [107], ComputeDRAM [65], PCM (Pinatubo) [108]). The key extension in Ambit (for in-situ processing) is to modify a decoder for three selected DRAM

	Triangle Counting [158]	k-Clique Listing [53]	k-Star-Clique Listing [84]	Maximal Cliques Listing [33, 62]	Link Prediction†	Link Prediction‡	Link Prediction§	Jarvis-Patrick Clustering [86]
SISA + merging intersection	$O(mc)^{\bigstar}$	$O\left(km\left(\frac{c}{2}\right)^{k-2}\right)^{\bigstar}$	$O\left(k^2m\left(\frac{c}{2}\right)^{k-1}\right)^{\bigstar}$	$O\left(cdn3^{c/3}\right)$	$O(md)$	$O\left(n^2+md\right)$	$O\left(n^2\right)^{\bigstar}$	$O(md)$
SISA + galloping intersection	$O(mc\log c)$	$O\left(km\left(\frac{c}{2}\right)^{k-2}\log c\right)$	$O\left(k^2m\left(\frac{c}{2}\right)^{k-1}\log c\right)$	$O\left(cn3^{c/3}\right)^{\bigstar}$	$O(mc\log c)^{\bigstar}$	$O\left(n^2+mc\log c\right)^{\bigstar}$	$O\left(n^2\right)^{\bigstar}$	$O(mc\log d)^{\bigstar}$

Table 6: **The impact of set intersection schemes (merging vs. galloping) on the runtime of graph mining algorithms.**. "$\bigstar$" means that a given SISA variant matches asymptotically the best known non-set-centric baseline, referenced in the top row. k, c, and d denote the size of the mined pattern, the graph degeneracy (a popular measure of graph sparsity) and the maximum vertex degree, respectively (other symbols are described in Section 2 and Table 2). Link prediction complexities are valid for the following vertex similarity measures: †Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors; ‡Total Neighbors; §Preferential Attachment [104, 132].

rows (that share the same set of sense amplifiers) in such a way that one amplifier connects directly to three DRAM cells. This enables logical AND and OR over two of such three rows, immediately computing the result in the third row (NOT is provided by including a single row of dual-contact DRAM cells [153]). *Importantly for SISA-PUM*, only three selected designated DRAM rows (per single DRAM subarray) are modified this way. Whenever the running code requests an in-situ memory operation, Ambit uses a recent RowClone technology [152] to copy (also in-situ) the rows that store input sets to these two designated rows, compute the result in-situ, and again use RowClone to copy the result to the destination (unmodified) DRAM row. Now, SISA-PUM uses Ambit's execution model and interface without any modifications: set intersection and union are processed with an in-situ AND and OR, respectively. Set difference is processed using set intersection, along with the well-known set algebra rule: $A \setminus B = A \cap B'$ [87].

SISA-PNM A set operation with no bulk bitwise processing uses SISA-PNM that relies on high bandwidth between processing units and DRAM (as in UPMEM [101], HMC [88], or Tesseract [5]). Adding or removing an element from a set stored as a DB ($A \cup \{x\}, A \setminus \{x\}$) is conducted with a single DRAM access to a specific memory cell. Other set operations on SAs that employ streaming or random accesses are also executed using small in-order cores.

8.2 Automatizing SISA Decisions

We use a small SISA Control Unit (SCU), cf. Section 3, to automatically decide which SISA instructions to run, and how. SCU could either be added to the CPU or to the DRAM circuitry (see the feasibility discussion later in this section), or – to avoid any HW modifications – it can also be emulated by a single designated in-order logic layer core.

Automatic Selection of SISA-PUM & SISA-PNM First, SCU decides whether to use SISA-PUM or SISA-PNM for given two sets. This decision is simple and is based on how sets are represented (this information is stored in the SISA metadata structure and possibly cached in SCU's cache).

Automatic Selection of Variants of Set Operations Second, SCU automatically detects if it is best to use merge or galloping, and processes input sets using the corresponding variant. This decision is guided by our performance models.

8.3 Performance Models for Set Operations

The runtime of each SISA instruction variant is dominated by either streaming or random accesses.

Streaming takes place when two sets A and B stored as SAs are processed using merging. We model the runtime as $l_M + W \cdot \max\{|A|, |B|\} \cdot \min\{b_M, b_L\}$. l_M and b_M are latency and bandwidth of accessing DRAM, and b_L is bandwidth between cores. The model conservatively assumes that A and B may be located in memory locations attached to different cores (e.g., in different vaults), and thus the overall bandwidth is bottlenecked by $\min\{b_M, b_L\}$.

To model **random accesses**, we simply count the number of performed operations and multiply it by the memory access latency. This gives $l_M \cdot \min\{|A|, |B|\} \cdot \log(\max\{|A|, |B|\})$ for a binary search over the larger of input sets, used when processing two SAs with galloping.

Then, a specific variant is **selected automatically** to minimize the predicted runtime. To **parametrize** these models, SISA needs (1) the sizes of processed sets, (2) their representation types, and (3) b_M, b_L, l_M. (1) and (2) are maintained (for each set) in a simple in-memory SM ("set metadata") structure. (3) describe the execution environment and are thus identical for each set; they are stored directly in the SCU. We instantiate (3) to reflect logic layers in Tesseract [5].

8.4 Details of SISA Hardware

Life Cycle of a Set A set is allocated with a standard malloc, augmented with setting the appropriate set information in the set metadata (SM) structure. Loading, processing, and storing sets is conducted by the respective existing elements such as logic layer cores; the SCU is only responsible for selecting the appropriate instruction variant to be executed. Once a set is deleted, the standard free call is used, together with removing a respecting entry from the SM structure.

Set Metadata The SM structure is a simple associative structure that holds constant amount of data per set (set representation, set size). The total SM size is $O(n)$ as there are n neighborhoods and a constant number of auxiliary sets. Thus, while we conservatively assume that SM is an in-memory structure, in practice it fits completely in cache or a small scratchpad. This is because many datasets processed by graph mining algorithms have small n, in the order of hundreds or thousands [143]. These graphs pose computational challenges, but these challenges come from high computational complexities (e.g., listing maximal cliques is NP-hard) or from relatively high edge counts m (as some vertices may have high degrees [143]), but *not* (or to a smaller extend) from n. Each SM entry describing one set also contains the set location. Now, entries in the SM structure are indexed by set IDs. A set ID is returned by a function creating a set, cf. Figure 3. Set IDs and set creation (and destruction) calls are used by a developer analogously to pointers and malloc/free calls.

Caching Set Metadata Depending on how SISA HW is deployed, the SM information can be cached in either a small dedicated scratchpad or cache (if the SCU is implemented as an additional circuitry), or in the standard cache of a logic layer core (if the SCU is emulated by a designated such core).

Using SISA-PNM and SISA-PUM Together We rely on Ambit's full compatibility with DRAM, as described in the original publication [153]. Specifically, Ambit fully preserves the DRAM interface: the sets are always stored in standard DRAM rows, and moved to the designated rows *only* for bulk bitwise processing. Thus, one can freely use standard DRAM accesses for any non-SISA-PUM set modifications.

Harnessing Parallelism SISA HW harnesses memory parallelism at different levels, enabling parallel execution of both a single set operation and different set operations. First, bit-level parallelism is enabled by using Ambit's bulk bitwise operations: bits in a row are ANDed or ORed in parallel. Second, pairs of bitvectors placed in different subarrays can be processed in parallel. This applies to other parts of the DRAM hierarchy, for example banks. Third, processing pairs of sets stored as integer arrays in different vaults can also be parallelized. Here, SISA benefits from the same effect of bandwidth scalability as the Tesseract graph accelerator [5].

Managing Concurrency SISA relies on established techniques (locks, lock-free protocols, and general parallel programming principles [78] and libraries such as OpenMP [39]) to manage concurrent accesses to the same set.

Memory Layout and Storage of Sets We ensure that storing SISA sets is feasible (i.e., a maximum-size neighborhood, represented as SA or DB, fits into a single vault).

8.5 SISA Hardware Cost and Feasibility

We also briefly discuss the hardware cost. First, the needed **DRAM chip modifications** are minimal and identical to those already discussed in Ambit. Second, as the **logic to be implemented in SCU** is straightforward decision making on what instruction variant to use, its costs are not prohibitive, as shown by many designs proposed in the past, for example in HyVE [81] (a hybrid vertex-edge memory hierarchy that uses ReRAM and DRAM) or in GraphH [51] (an accelerator that combines HMC with SRAM). Third, the code of all SISA instructions is also straightforward: a simple binary search (galloping), merging of two arrays (merge), or setting/clearing a DRAM cell (set element add/remove). Thus, they can be trivially deployed in in-order cores in the logic layer of 3D stacked DRAM, as shown by other designs [51].

9. EVALUATION

We illustrate example performance advantages from SISA.

9.1 Methodology, Setup, Parameters

Simulation Infrastructure We use Sniper [77] with the Pin frontend [114]. Sniper is a popular cycle-level simulator used in many works proposing various architectural extensions for both CPUs and memory subsystem [126, 174].

SISA Implementation We simulate the SISA HW design and the ISA, instrumenting the code so that the simulation toolchain can distinguish between SISA and non-SISA instructions. To model each component of SISA, we add the respective set instructions and simulate the SCU (a small fixed delay), the cache in SCU (with the LRU policy), the SM structure (random memory accesses whenever the SCU cache is not hit), and the execution of all used set operations by appropriate delays in the simulation execution. For operations based on streaming and random memory accesses, we use the performance models described in § 8.3. To simulate SISA-PUM, we model a run-time of in-situ operations with a delay $l_M + l_I \cdot \lceil n/(qS) \rceil$, where l_M is the latency of accessing DRAM (to initiate the operation) and l_I is the latency of executing a single in-situ instruction. $\lceil n/(qR) \rceil$ models a scenario when the bitvector size n exceeds the size of all DRAM rows that can be processed in parallel (cf. Table 2).

Simulated Platform for SISA & SISA Parametrization For concreteness, we set the platform for executing SISA instructions to match Tesseract [5] (for SISA-PNM) and Ambit [153] (for SISA-PUM). The former has simple in-order cores (1 core/vault in its logic layer) with 32 KB L1 instruction/data caches, no L2, 16 8GB HMCs (128 GB in total), 32 vaults/cube, 16 banks/vault. Each vault offers 16 GB/s of memory bandwidth to its core. Thus, we assume scalable bandwidth as proposed by Tesseract: using more vaults increases the total memory bandwidth. In the latter, we set the DRAM row rank size to 8 KB, following Ambit [153]. Next, we set the parameter $t \in [0;1]$ (that controls the bias towards using DBs or SAs to store neighborhoods) to 0.4 (i.e., 40% of neighborhoods are stored as DBs); we also analyze other values. We ensure that the total storage used for neighborhoods does not exceed the size of the simple CSR graph storage by more than 10%. Finally, we set the size of SISA SCU's cache to be 32 KB (matching Tesseract's L1).

Simulated Platform for non-SISA Instructions & Baselines For any non-SISA instructions and comparison baselines, we use a high-performance Out-of-Order manycore CPU. Each core has a 128-entry instruction window, a branch predictor, 32 KB L1 instruction/data caches, a 256 KB L2 cache. All cores share an 8 MB L3 cache. There is also a four-way associative 64-entry D-TLB, a 128-entry I-TLB, and a 512-entry S-TLB. For fair comparison, *we also use bandwidth scalability in this configuration, i.e., we increase the memory bandwidth with the number of cores, matching it with that of SISA-PNM.*

Considered Mining Problems The graph mining problems that we consider are clustering with the Jaccard (`cl-jac`), overlap (`cl-ovr`), and total neighbors (`cl-tot`) coefficients, listing k-cliques (`kcc-`k, $k \in \{4,5,6\}$), k-clique-stars (`ksc-`k, $k \in \{4,5,6\}$), maximal cliques (`mc`), triangles (`tc`), and subgraph isomorphism (`si-`k`s` for k-stars).

Comparison Targets: Hand-Tuned Algorithms Our most important (the most challenging to outperform) baselines are hand-optimized parallel algorithms for each graph mining problem. Specifically, we use a tuned version from the GAP Benchmark Suite [14] for `tc`, Eppstein's version of BK for `mc` [62], Danisch' scheme for `kcc-`k [53], enhanced Jabbour's scheme for `ksc-`k [84], parallel VF2 for `si-`k`s` [46], and `cl-jac` based on counting triangles in the GAP suite [14]. All used baselines have competitive work and depth complexities, cf. Table 6. For fair comparison, all baselines benefit from the high bandwidth of PIM. We consider algorithms that do not explicitly use set algebra (denoted with `_non-set`) and their set-centric variants (denoted with `_set-based`).

Comparison Targets: Pattern Matching Frameworks When possible, we compare to graph pattern matching frameworks: Peregrine [85] (a very recent and fast design that represents accelerators such as Gramer [194], cf. "pattern matching" in Table 1), and RStream [176] which represents accelerators such as TrieJax [94] based on relation algebra (cf. "joins" in Table 1). We stress that we focus on comparing to (much faster) hand-tuned parallel algorithms.

Graphs We select a broad set of input datasets from Network Repository [144], considering biological (`bio-`), interaction (`int-`), social (`soc-`), brain (`bn-`), dynamic (`D`), web

(web-), economical (econ-), ecological (eco-), and structural (str-) networks. We pick graphs with different structural properties (low/high density, small/large maximum degree, low/high degree distribution skew, etc.).

Tackling Long Simulation Runtimes Most benchmarks use relatively small graphs because (1) we run cycle accurate simulations, tracing all memory accesses, which is very time-consuming, and (2) the considered algorithms are computationally hard and even software codes use graphs much smaller than those used with algorithms such as PageRank [53,62]. However, even this is often not enough to enable finishing simulations of algorithms such as Bron-Kerbosch. Thus, we usually also pre-specify a number of graph patterns to be found. Past work analogously handled long simulations graph algorithms [5] such as PageRank (limiting #iteration).

Performance Measures & Summaries: We focus on plain runtimes; this is recommended when measuring performance of parallel codes [79] as speedup may be misleading because it is higher on unoptimized baselines. However, for overview, we also *summarize speedups* (following [79]), i.e., we provide (1) speedups of *average runtimes* ("arithm"), and (2) *geometric means of speedups* of all data points ("geom").

9.2 Discussion of Results

Comparison to Hand-Tuned Algorithms We first analyze run-times with all available cores, comparing SISA set-centric variants to non-set-based and set-based hand-tuned parallel baselines that all benefit from high-bandwidth storage. The results are in Figure 6. SISA is almost always the fastest by a large margin of at least 2×, often more than 10× (than non-set schemes). The differences vary depending on the processed graphs and the considered problem. Gains are usually larger on graphs with large maximum degrees, such as brain graphs, where SISA-PUM is used more often to directly process sets inside DRAM, reducing the latency. Such graphs are prevalent in many computational domains [144], and this is the case for the majority of considered datasets.

Algorithmic vs. Architectural Speedups We also observe speedups from using only set-centric formulations (over non-set-based variants). Namely, speedups of "_set-based" schemes over the "_non-set" ones indicate gains from purely *algorithmic* (set-centric) changes, while speedups of "_sisa" schemes over the "_set-based" indicate gains only from *architectural* changes (i.e., from using PIM). First, the differences between _set-based and _non-set heavily depend on the targeted mining algorithm. These speedups are particularly visible for more complex algorithms such as mc, with multiple nested loops and/or recursion. Packaging different parts of such algorithms into, e.g., set intersections, and being able to control the used operation variant (e.g., merging based on streaming) helps to utilize features such as high sequential bandwidth. Contrarily, for certain simpler schemes such as clustering, the very tuned _non-set baseline outperforms _set-based (while still falling short of _sisa). Second, the difference between _set-based and _sisa depend more on the used graph. Here, in many cases, _sisa is only marginally faster than _set-based, because the graph structure (e.g., sizes of neighborhoods) favor using SAs rather than DBs, diminishing benefits from SISA-PUM (e.g., for econ- graphs) and equalizing the differences because both _set-based and _non-set take advantage from

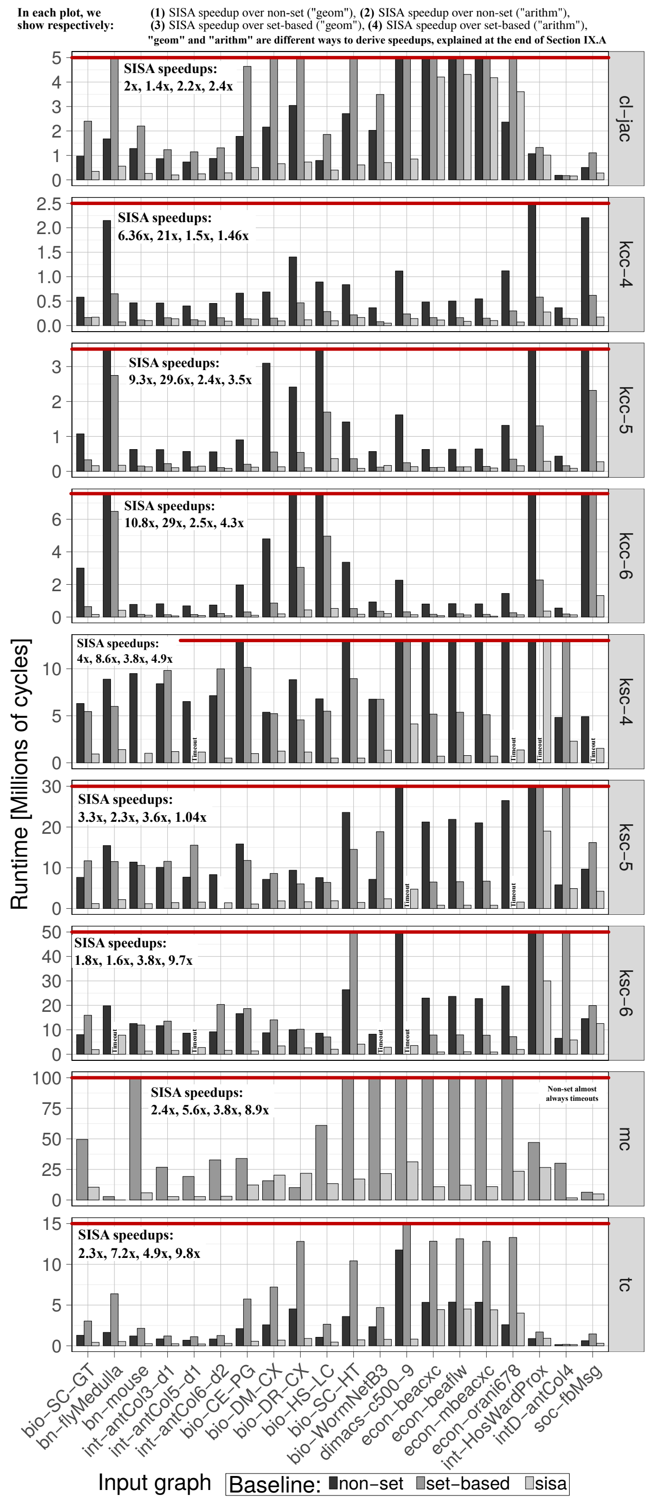
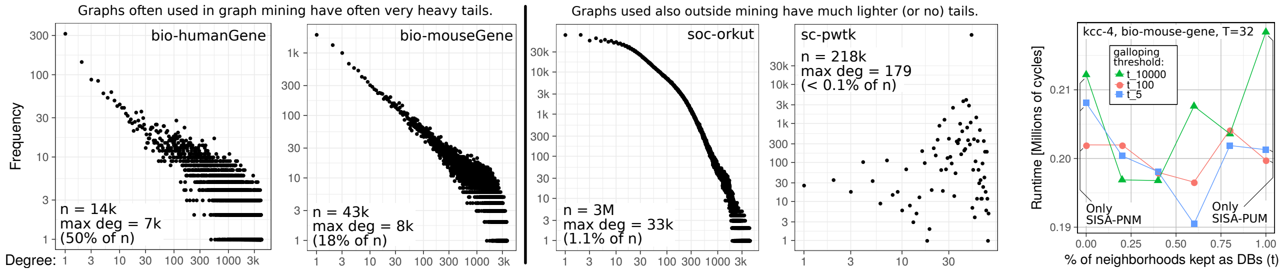


Figure 6: **Run-times with full parallelism**. The bold red line indicates the cutoff of long simulation runtimes, used for readability (the bars reaching the line have much larger runtimes). No bar indicates the timeout of the respective baseline (>24h). The results for cl-jac (clustering based on the Jaccard coefficient) are very similar to those that use other coefficients and for link prediction as well as vertex similarity. All 32 cores are used. **Acronyms are stated in "Comparison Targets: Hand-Tuned Algorithms".**



(a) Degree distribution analysis

(b) Sensitivity analysis.

Figure 7: Figure 7a: Differences between degree distributions in graphs used mostly in graph mining and the ones used also outside graph mining (on the right). Figure 7b: Sensitivity analysis: the percentage of neighborhoods stored as dense bitvectors vs. different thresholds for using the galloping or the merging intersection.

the high bandwidth setting. In other cases (e.g., bio-HS-LC), more vertices have large enough degrees to benefit from DBs and low latencies of SISA-PUM.

Scalability We also analyze how run-times change when varying numbers of threads T, for a fixed graph size ("strong scaling"), and when increasing T proportionally to the graph size ("weak scalability"). To fix the used graph model, we use Kronecker graphs [105] and we vary the number of edges/vertex. SISA maintains its speedups, but they become less distinctive when T is small. This is expected because fewer threads exert less pressure on the memory subsystem, and there is overall smaller potential from using PIM in SISA.

Large Graphs We execute SISA on several large graphs, see Figure 8. Runtime benefits from SISA and the set-centric formulations are similar to those in smaller graphs in Figure 6. The only two graphs where SISA and non-SISA set baselines are comparable, are sc-pwtk and soc-orkut. This is because these networks, due to their origin (social and scientific) do not have large cliques or very dense clusters (unlike, e.g., genome graphs), somewhat lowering SISA benefits.

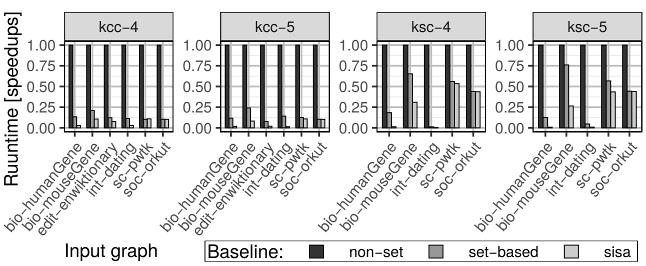


Figure 8: **Run-times for large graphs**. The bold red line indicates the cutoff of long simulation runtimes, used for readability (the bars reaching the line have much larger runtimes). 8 cores are used.

Comparison to Pattern Matching Frameworks We compare SISA set-centric algorithms to Peregrine and RStream. Peregrine is able to express only listing k-cliques and subgraph isomorphism, and maximal clique listing in a limited way (i.e., it does not offer a native scheme for MC and we implemented it by iterating over possible clique sizes and listing maximal cliques of each size). RStream is only able to find k-cliques. In each case, SISA baselines are *much* faster: 10-100× than Peregrine (and more than 1,000× for mc due to Peregrine's inability to natively support mc), and more than 100× for RStream. This is because these frameworks focus on programmability in the first place, sacrificing performance, while in SISA we start with tuned graph algorithms and only then restructure them with the set-centric paradigm.

Sensitivity Analysis & Design Exploration We investigate the impact from varying SISA parameters.

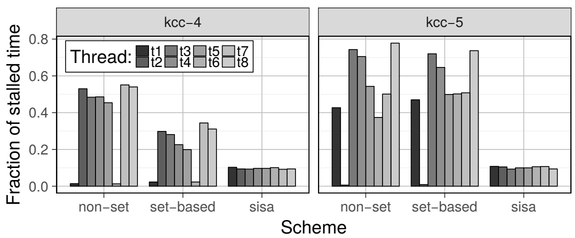
SCU cache First, not using the SCU cache results in the loss of performance of ≈1.5× for $T = 1$ and ≈0.05-0.1× for $T = 32$. The lower performance loss for high T is because, with more threads executing set operations, it becomes increasingly more difficult to ensure high hit ratio. Overall, the behavior of the SCU cache is similar to that of other such units such as L1, including varying cache parameters such as size.

Varying Fraction of Dense/Sparse Neighb Second, we observe that – while using SISA-PUM is beneficial for the overall performance – too many neighborhoods stored as DBs result in slowdowns. This is because when sparse neighborhoods are also stored as DBs, processing such sets (which have always size n bits) with SISA-PUM begins to take more time than processing them with SISA-PNM. Thus, it is relevant to not choose the bias parameter to be too high. We find that 0.4 works well for most processed graphs. We illustrate this in Figure 7b, where we analyze how the performance changes when varying the fraction of largest neighborhoods stored as DBs. Smallest and largest fractions that correspond to *using only SISA-PNM or only SISA-PUM* give slowest runtimes. We also vary the "galloping threshold", i.e., the relative difference between two sets that causes the set operation to switch to the galloping variant. For example, the value of 5 indicates that galloping is used if any of the two sets is at least 5× larger than the other one. While this threshold influences performance, the general pattern stays the same.

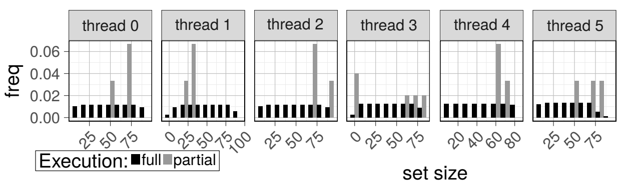
We also analyze the **impact from the degree distributions of datasets**, see Figure 7a. Graphs often used in graph mining, such as biological networks, that SISA focuses on, have often *very* heavy tails. This implies *many large neighborhoods and very dense large clusters, benefiting from SISA-PUM*. For example, the human genome graph has many vertices connected to more than 30% of all other vertices. Other graphs such as social networks have *much lighter tails*, cf. soc-orkut and sc-pwtk in Figure 7a. This is because these networks, due to their origin (social, scientific) do not have large cliques or very dense clusters. Such graphs benefit less from SISA-PUM. Still, using SISA-PNM enables high performance, outperforming tuned non-set-based baselines, cf. Figure 8.

We also analyze **load balancing**. Figure 9a illustrates total fractions of time during which each parallel thread is stalled when executing a given algorithm. SISA stall times are low because its design implicitly tackles two types of load imbalance. First, SISA's performance models enable adaptive selection of the best variant of a set algorithm to be executed for any two sets. This minimizes load imbalance from processing two sizes that differ a lot in sizes. Second, load imbalance due to processing imbalanced *pairs* of sets (i.e., two very small and two very large sets) is alleviated by the fact that very large pairs of sets are processed with very fast SISA-PUM.

We also show that the reduced simulation runtimes do not artificially eliminate load imbalance. We gather traces of executed set operations in full vs. partial simulation executions, and we plot histograms of the sizes of processed sets, see Figure 9b. In both types of executions, we encounter large sets which are the primary source of load imbalance.



(a) Total amounts of stall times of different 8 parallel threads.



(b) Histograms of sizes of processed sets of full vs. partial executions, for 6 parallel threads (the remainder of threads behave similarly). Graph: int-antCol3-d1. Problem: kcc-4.

Figure 9: Load balancing analysis.

SISA Limitations For some graphs with small maximum degrees (e.g., `soc-fbMsg`) in Figure 6, SISA speedups are smaller, or even (in the extreme cases) result in slowdowns. This is because the benefits from SISA-PUM, or from the automatic selection of the most beneficial set operation variant, are out-weighted by having to process too many large bitvectors. This effect rare, and it can be alleviated by reducing the number of neighborhoods stored as DBs. In this case, the performance of SISA variants gradually converges towards that of standard CSR based set-centric algorithms. We plan on addressing it with advanced bitvector representations.

10. RELATED WORK & DISCUSSION

We already extensively described related graph processing paradigms (Table 1) and various software related graph processing efforts (Section 1) [20, 20, 25, 115, 146]. We now briefly summarize other related areas. First, we conducted an exhaustive analysis of existing hardware accelerators as well as ISA designs for graph processing, see Table 7. The analysis indicates that SISA offers the *only* hardware acceleration for a broad family of problems such as maximal clique listing or clustering. Additionally, this work is an example of how to seamlessly integrate PUM and PNM capabilities in a single system. They work synergistically and produce significantly better results than working separately. Works orthogonal to SISA include HW accelerated dynamic (time-evolving) graph analytics [34, 35, 76], or external memory HW accelerated graph processing [57, 92, 119]. *One could use the latter as a SISA backend for external memory set instructions; we leave details for future work.*

While in the current SISA version we focus on implementing and executing set operations in set-centric algorithm

Reference / Accelerator	Prob.	Key memory mechanism	Pattern M. mc	Pattern M. kc	Pattern M. ds	Pattern M. si	Learning vs	Learning lp	Learning cl	"Low-c." av	"Low-c." bf	"Low-c." pr	"Low-c." cc	is	xl	ab
[Pi] GaaS-X [38]	SpMV	**[e]** CAM/MAC	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	◧
[Pi] GraphSAR [52]	ver-c	**[e]** ReRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	✗
[Pi] GraphiDe [10]	low-c	**[e]** DRAM	✗	✗	✗	✗	■	✗	✗	✗	■	■	■	◧	◧	✗
[Pi] GraphIA [106]	edge-c	**[e]** DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	✗
[Pc] Spara [200]	ver-c	**[e]** ReRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[Pc] GraphQ [209]	ver-c	**[e]** HMC	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[Pc] GraphS [11]	low-c	**[e]** SOT-MRAM	✗	✗	✗	✗	■	✗	✗	✗	■	■	■	✗	✗	✗
[Pc] RAGra [82]	ver-c	**[e]** 3D ReRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[Pc] GRAM [201]	ver-c	**[e]** ReRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[Pc] GraphR [162]	SpMV	**[e]** ReRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	◧
[Pc] GraphP [196]	ver-c	**[e]** HMC	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	■
[Pc] Tesseract [5]	low-c	**[e]** HMC	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	◧
[Pc] PIM-Enabled [6]	low-c	**[e]** HMC	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	◧	■	◧
[Pc] Gao et al. [66]	low-c	3D DRAM	✗	✗	◧	✗	✗	✗	✗	✗	■	■	■	✗	■	✗
[Pc] LiM [207, 208]	SpMSpM	**[e]** 3D DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	✗
[A] Gramer [94]	pattern m.	DRAM, cache	✗	■	■	◧	✗	✗	✗	✗	✗	✗	✗	✗	◧	✗
[A] TrieJax [94]	joins	DRAM, LLC	✗	■	■	✗	✗	✗	✗	✗	✗	✗	✗	✗	■	◧
[A] HyGCN [189]	GCN	eDRAM	✗	✗	✗	✗	◧	■	■	✗	✗	✗	◧	✗	◧	■
[A] Outerspace [136]	SpMSpM	HBM	✗	◧	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	✗
[A] Domino [184]	low-c	on-chip buffers	✗	◧	✗	✗	✗	✗	✗	■	■	■	✗	✗	✗	
[A] GraphPIM [131]	low-c	**[e]** HMC	✗	◧	◧	✗	✗	✗	✗	✗	■	■	■	■	■	✗
[A] Graphicionado [73]	ver-c	**[e]** eDRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	◧
[A] Ozdal et al. [135]	ver-c	**[e]** caches	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	◧
[M] GraphSSD [119]	low-c	**[e]** SSD	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	◧
[M] GRASP [63]	low-c	**[e]** LLC	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[M] DROPLET [12]	edge-c	**[e]** DRAM pref.	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	✗
[M] Ainsworth [7]	low-c	**[e]** DRAM pref.	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	✗
[M] HyVE [81]	ver-c	ReRAM, SRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	✗
[M] HATS [127]	low-c	**[e]** caches	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	■
[M] OSCAR [159]	edge-c	**[e]** scratchpads	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	◧
[M] IMP [195]	low-c	**[e]** caches	✗	◧	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	◧
[F] GraphABCD [191]	low-c	DRAM	✗	✗	✗	✗	◧	◧	◧	✗	■	■	■	✗	■	■
[F] Wang et al. [175]	clustering	BRAM	✗	✗	✗	✗	✗	✗	◧	✗	✗	✗	✗	■	✗	◧
[F] ForeGraph [49, 50]	low-c	BRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	◧
[F] Yang [190]	ver-c	DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	◧
[F] Yao [192]	low-c	DRAM	✗	◧	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[F] Zhou [204]	edge-c	DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	◧
[F] ExtraV [102]	low-c	DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	■
[F] Ma [116]	low-c	DRAM	✗	◧	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[F] Zhou [205]	ver-c, edge-c	DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	■
[F] GraVF [61]	ver-c	BRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	■
[F] Zhou [202, 203]	edge-c	DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	✗	◧
[F] GraphOps [134]	low-c	BRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	■
[F] FPGP [48]	ver-c	DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[F] GraphSoC [95]	low-c, SpMV	BRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	■	◧	◧
[F] GraphGen [133]	ver-c	DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	◧
[F] GraphStep [96]	low-c	BRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	■
[F] Betkaoui et al. [30]	low-c	DRAM	✗	◧	◧	✗	✗	✗	✗	✗	■	■	■	✗	■	◧
[A+Pc] EnGN [75]	GNN	**[e]** HBM	✗	✗	✗	✗	◧	■	■	✗	✗	◧	✗	✗	✗	✗
[A+Pc] OMEGA [2]	low-c	**[e]** Scratchpads	✗	◧	✗	✗	✗	✗	◧	✗	■	■	■	✗	✗	◧
[A+Pc+M] GraphH [51]	ver-c	**[e]** HMC	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	■	✗
[F+Pc] HRL [67]	ver-c	**[e]** 3D DRAM	✗	✗	✗	✗	✗	✗	✗	✗	■	■	■	✗	◧	✗
[Pc+Pi] SISA [This work]	Graph mining	PIM	■	■	■	■	■	■	■	■	✗	✗	✗	■	■	■

Table 7: **Comparison of SISA to graph-related accelerators, focusing on supported graph mining problems and offered architecture elements**. "■": Support / significant focus. "◧": Partial support / some focus. "✗": no support / no focus. **Addressed problems:** see Table 1. **Graph problems and algorithms: as in Table 1**. **Architecture and stack elements that are considered and discussed:** **is**: an ISA, or its extensions, **xl**: a cross-layer design, **ab**: a programming paradigm **Classes of accelerators:** **[Pi]**: *in-situ* PIM, **[Pc]**: *near memory* PIM (e.g., logic layers), **[A]**: ASIC, **[M]**: focus on memory hierarchy enhancements, **[F]**: FPGA, **[e]** focus on extensions and modifications to the established (already proposed) HW technology,

formulations using PIM, SISA could be extended into different directions. This includes parallel and distributed execution of set operations, and implementing them using high-performance techniques such as Remote Direct Memory Access [19, 21, 68, 150]. One could also enable more efficient execution of set-centric graph mining algorithms in the context of modern complex heterogeneous architectures that may host massively parallel on-chip networks [18], NUMA and systems with locality effects [151,169], or FPGAs [17,26,55]. One could also incorporate various forms of graph compression and summarization [22, 27, 29, 111].

Sets are used in different graph algorithms, to simplify operations on selected data structures [25, 33, 99, 124, 137, 149, 155]. For example, the BFS frontier can be modeled as a set. Here, SISA's main contribution is *not* to simply use set notation. Instead, from the algorithmic perspective, SISA is the first design that (1) uses set operations as the *primary building blocks*, which break down complex graph mining algorithms into simple units of parallel execution, and (2) identifies the "appropriate" set operations (i.e., operations that are easily accelerated with PIM) and reformulates selected algorithms so that they use such operations, cf. Table 3.

11. DISCUSSION AND CONCLUSION

We develop the first hardware acceleration approach for general graph pattern matching and learning. First, we offer a set-centric programming paradigm, where one identifies and exposes set operations in graph mining algorithms. This enables competitive time complexities and succinct formulations. Second, the set-centric algorithms are mapped to SISA, a small yet expressive "set-centric" ISA extension for graph mining. SISA could be extended with CISC-style set instructions that accept multiple arguments (e.g., $A_1 \cap ... \cap A_l$) to facilitate optimizations such as vectorization with loop unrolling. Due to the generality of set algebra, SISA can be used for problems beyond graph mining. Third, while we pick in-situ and logic layer PIM for hardware acceleration, SISA's set algebra interface could easily use other hardware backends, for example a GPU backend for fast SIMD-based set intersections [74], FPGAs [26], or even execution in caches [3, 130]. Our cross-layer architecture could also be extended in other directions, for example by providing compiler support for generating SISA programs from set-centric formulations.

REFERENCES

[1] C. R. Aberger, A. Lamb, S. Tu, A. Nötzli, K. Olukotun, and C. Ré, "Emptyheaded: A relational engine for graph processing," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 4, pp. 1–44, 2017.

[2] A. Addisie, H. Kassa, O. Matthews, and V. Bertacco, "Heterogeneous memory subsystem for natural graph analytics," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 134–145.

[3] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 481–492.

[4] C. C. Aggarwal and H. Wang, *Managing and mining graph data*. Springer, 2010, vol. 40.

[5] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[6] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*. IEEE, 2015, pp. 336–348.

[7] S. Ainsworth and T. M. Jones, "An event-triggered programmable prefetcher for irregular workloads," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 578–592, 2018.

[8] M. Al Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," in *SDM06: workshop on link analysis, counter-terrorism and security*, 2006.

[9] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*. Springer, 2011, pp. 243–275.

[10] S. Angizi and D. Fan, "Graphide: A graph processing accelerator leveraging in-dram-computing," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 45–50.

[11] S. Angizi, J. Sun, W. Zhang, and D. Fan, "Graphs: A graph processing accelerator leveraging sot-mram," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 378–383.

[12] A. Basak, S. Li, X. Hu, S. M. Oh, X. Xie, L. Zhao, X. Jiang, and Y. Xie, "Analysis and optimization of the memory hierarchy for graph processing workloads," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 373–386.

[13] O. Batarfi, R. El Shawi, A. G. Fayoumi, R. Nouri, A. Barnawi, and S. Sakr, "Large scale graph processing systems: survey and an experimental evaluation," *Cluster Computing*, vol. 18, no. 3, pp. 1189–1213, 2015.

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

[15] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, "A modular benchmarking infrastructure for high-performance and reproducible deep learning," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 66–77.

[16] M. Besta, A. Carigiet, Z. Vonarburg-Shmaria, K. Janda, L. Gianinazzi, and T. Hoefler, "High-performance parallel graph coloring with strong guarantees on work, depth, and quality," *arXiv preprint arXiv:2008.11321*, 2020.

[17] M. Besta, M. Fischer, T. Ben-Nun, D. Stanojevic, J. D. F. Licht, and T. Hoefler, "Substream-centric maximum matchings on fpga," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 2, pp. 1–33, 2020.

[18] M. Besta, S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler, "Slim noc: A low-diameter on-chip network topology for high energy efficiency and scalability," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 43–55, 2018.

[19] M. Besta and T. Hoefler, "Fault tolerance for remote memory access programming models," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, 2014, pp. 37–48.

[20] M. Besta and T. Hoefler, "Accelerating irregular computations with hardware transactional memory and active messages," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, 2015, pp. 161–172.

[21] M. Besta and T. Hoefler, "Active access: A mechanism for high-performance distributed data-centric computations," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 155–164.

[22] M. Besta and T. Hoefler, "Survey and taxonomy of lossless graph compression and space-efficient graph representations," *arXiv preprint arXiv:1806.01799*, 2018.

[23] M. Besta, R. Kanakagiri, H. Mustafa, M. Karasikov, G. Rätsch, T. Hoefler, and E. Solomonik, "Communication-efficient jaccard similarity for high-performance distributed genome comparisons," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 1122–1132.

[24] M. Besta, F. Marending, E. Solomonik, and T. Hoefler, "Slimsell: A

vectorizable graph representation for breadth-first search," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 32–41.

[25] M. Besta, M. Podstawski, L. Groner, E. Solomonik, and T. Hoefler, "To push or to pull: On reducing communication and synchronization in graph computations," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2017, pp. 93–104.

[26] M. Besta, D. Stanojevic, J. D. F. Licht, T. Ben-Nun, and T. Hoefler, "Graph processing on fpgas: Taxonomy, survey, challenges," *arXiv preprint arXiv:1903.06697*, 2019.

[27] M. Besta, D. Stanojevic, T. Zivic, J. Singh, M. Hoerold, and T. Hoefler, "Log (graph): a near-optimal high-performance graph representation," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. ACM, 2018, p. 7.

[28] M. Besta, Z. Vonarburg-Shmaria, Y. Schaffner, L. Schwarz, G. Kwasniewski, L. Gianinazzi, J. Beranek, K. Janda, T. Holenstein, S. Leisinger *et al.*, "Graphminesuite: Enabling high-performance and programmable graph mining algorithms with set algebra," *VLDB*, 2021.

[29] M. Besta, S. Weber, L. Gianinazzi, R. Gerstenberger, A. Ivanov, Y. Oltchik, and T. Hoefler, "Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–25.

[30] B. Betkaoui, D. B. Thomas, W. Luk, and N. Przulj, "A framework for fpga acceleration of large graph problems: Graphlet counting case study," in *Field-Programmable Technology (FPT), 2011 International Conference on*. IEEE, 2011, pp. 1–8.

[31] G. E. Blelloch and B. M. Maggs, *Parallel Algorithms*, 2nd ed. Chapman & Hall/CRC, 2010, p. 25.

[32] O. Boruvka, "O jistém problému minimálním," 1926.

[33] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[34] L. Bustio, R. Cumplido, R. Hernández, J. M. Bande, and C. Feregrino, "Frequent itemsets mining in data streams using reconfigurable hardware," in *International Workshop on New Frontiers in Mining Complex Patterns*. Springer, 2015, pp. 32–45.

[35] L. Bustio-Martínez, R. Cumplido, M. Letras-Luna, C. F. Uribe, R. Hernández-León, and J. M. Bande-Serrano, "Approximate frequent itemsets mining on data streams using hashing and lexicographie order in hardware," in *2017 IEEE 8th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE, 2017, pp. 1–4.

[36] F. Cazals and C. Karande, "A note on the problem of reporting maximal cliques," *Theoretical Computer Science*, vol. 407, no. 1-3, pp. 564–568, 2008.

[37] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, generators, and algorithms," *ACM computing surveys (CSUR)*, vol. 38, no. 1, p. 2, 2006.

[38] N. Challapalle, S. Rampalli, L. Song, N. Chandramoorthy, K. Swaminathan, J. Sampson, Y. Chen, and V. Narayanan, "Gaas-x: Graph analytics accelerator supporting sparse data representation using crossbar architectures," *ISCA*, 2020.

[39] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[40] H. Chen, M. Liu, Y. Zhao, X. Yan, D. Yan, and J. Cheng, "G-miner: an efficient task-oriented graph mining system," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 32.

[41] X. Chen, R. Dathathri, G. Gill, and K. Pingali, "Pangolin: An efficient and flexible graph mining system on cpu and gpu," *arXiv preprint arXiv:1911.06969*, 2019.

[42] J. Cheng, L. Zhu, Y. Ke, and S. Chu, "Fast algorithms for maximal clique enumeration with limited memory," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 1240–1248.

[43] J. Cheng, J. X. Yu, B. Ding, S. Y. Philip, and H. Wang, "Fast graph pattern matching," in *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 2008, pp. 913–922.

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

[45] D. J. Cook and L. B. Holder, *Mining graph data*. John Wiley & Sons, 2006.

[46] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[47] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[48] G. Dai, Y. Chi, Y. Wang, and H. Yang, "Fpgp: Graph processing framework on fpga a case study of breadth-first search," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 105–110. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847339

[49] G. Dai, T. Huang, Y. Chi, N. Xu, Y. Wang, and H. Yang, "Foregraph: Exploring large-scale graph processing on multi-fpga architecture," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 217–226.

[50] G. Dai, T. Huang, Y. Chi, N. Xu, Y. Wang, and H. Yang, "Foregraph: Exploring large-scale graph processing on multi-fpga architecture," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 217–226. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021739

[51] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "Graphh: A processing-in-memory architecture for large-scale graph processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 640–653, 2018.

[52] G. Dai, T. Huang, Y. Wang, H. Yang, and J. Wawrzynek, "Graphsar: a sparsity-aware processing-in-memory architecture for large-scale graph processing on rerams," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 120–126.

[53] M. Danisch, O. Balalau, and M. Sozio, "Listing k-cliques in sparse real-world graphs," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2018, pp. 589–598.

[54] W. H. Day and D. Sankoff, "Computational complexity of inferring phylogenies by compatibility," *Systematic Biology*, vol. 35, no. 2, pp. 224–229, 1986.

[55] J. de Fine Licht, M. Besta, S. Meierhans, and T. Hoefler, "Transformations of high-level synthesis codes for high-performance computing," *arXiv e-prints*, pp. arXiv–1805, 2018.

[56] L. Dhulipala, G. E. Blelloch, and J. Shun, "Theoretically efficient parallel graph algorithms can be fast and scalable," in *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, 2018, pp. 393–404.

[57] L. Dhulipala, C. McGuffey, H. Kang, Y. Gu, G. Blelloch, P. Gibbons, and J. Shun, "Sage: Parallel semi-asymmetric graph algorithms for nvrams," *PVLDB*, 2020.

[58] V. Dias, C. H. Teixeira, D. Guedes, W. Meira, and S. Parthasarathy, "Fractal: A general-purpose graph pattern mining system," in *Proceedings of the 2019 International Conference on Management of Data*. ACM, 2019, pp. 1357–1374.

[59] S. Dua and X. Du, *Data mining and machine learning in cybersecurity*. CRC press, 2016.

[60] J. D. Eblen, C. A. Phillips, G. L. Rogers, and M. A. Langston, "The maximum clique enumeration problem: algorithms, applications, and implementations," in *BMC bioinformatics*, vol. 13, no. S10. Springer, 2012, p. S5.

[61] N. Engelhardt and H. K.-H. So, "Gravf: A vertex-centric distributed graph processing framework on fpgas," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 2016, pp. 1–4.

[62] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju*

Island, Korea, December 15-17, 2010, Proceedings, Part I, 2010, pp. 403–414. [Online]. Available: https://doi.org/10.1007/978-3-642-17517-6_36

[63] P. Faldu, J. Diamond, and B. Grot, "Poster: Domain-specialized cache management for graph analytics," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2019, pp. 473–474.

[64] B. Gallagher, "Matching structure and semantics: A survey on graph-based pattern matching." in *AAAI Fall Symposium: Capturing and Using Patterns for Evidence Detection*, 2006, pp. 45–53.

[65] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "Computedram: In-memory compute using off-the-shelf drams," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 100–113.

[66] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 2015, pp. 113–124.

[67] M. Gao and C. Kozyrakis, "Hrl: Efficient and flexible reconfigurable logic for near-data processing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Ieee, 2016, pp. 126–137.

[68] R. Gerstenberger, M. Besta, and T. Hoefler, "Enabling highly-scalable remote memory access programming with mpi-3 one sided," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[69] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-Memory: A Workload-driven Perspective," *IBM JRD*, 2019.

[70] S. Ghose, K. Hsieh, A. Boroumand, R. Ausavarungnirun, and O. Mutlu, "The processing-in-memory paradigm: Mechanisms to enable adoption," in *Beyond-CMOS Technologies for Next Generation Computer Design*. Springer, 2019, pp. 133–194.

[71] L. Gianinazzi, P. Kalvoda, A. De Palma, M. Besta, and T. Hoefler, "Communication-avoiding parallel minimum cuts and connected components," *ACM SIGPLAN Notices*, vol. 53, no. 1, pp. 219–232, 2018.

[72] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 721–732.

[73] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–13.

[74] S. Han, L. Zou, and J. X. Yu, "Speeding up set intersections in graph algorithms using simd instructions," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 1587–1602.

[75] L. He, "Engn: A high-throughput and energy-efficient accelerator for large graph neural networks," *arXiv preprint arXiv:1909.00155*, 2019.

[76] E. R. Hein, "Near-data processing for dynamic graph analytics," Ph.D. dissertation, Georgia Institute of Technology, 2018.

[77] W. Heirman, T. Carlson, and L. Eeckhout, "Sniper: Scalable and accurate parallel multi-core simulation," in *8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES-2012)*. High-Performance and Embedded Architecture and Compilation Network of …, 2012, pp. 91–94.

[78] M. Herlihy, N. Shavit, V. Luchangco, and M. Spear, *The art of multiprocessor programming*. Newnes, 2020.

[79] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2015, pp. 1–12.

[80] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 158–167.

[81] T. Huang, G. Dai, Y. Wang, and H. Yang, "Hyve: Hybrid vertex-edge memory hierarchy for energy-efficient graph processing," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 973–978.

[82] Y. Huang, L. Zheng, X. Liao, H. Jin, P. Yao, and C. Gui, "Ragra: Leveraging monolithic 3d reram for massively-parallel graph processing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1273–1276.

[83] A. P. Iyer, Z. Liu, X. Jin, S. Venkataraman, V. Braverman, and I. Stoica, "{ASAP}: Fast, approximate graph pattern mining at scale," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 745–761.

[84] S. Jabbour, N. Mhadhbi, B. Raddaoui, and L. Sais, "Pushing the envelope in overlapping communities detection," in *International Symposium on Intelligent Data Analysis*. Springer, 2018, pp. 151–163.

[85] K. Jamshidi, R. Mahadasa, and K. Vora, "Peregrine: a pattern-aware graph mining system," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[86] R. A. Jarvis and E. A. Patrick, "Clustering using a similarity measure based on shared near neighbors," *IEEE Transactions on computers*, vol. 100, no. 11, pp. 1025–1034, 1973.

[87] T. Jech, *Set theory*. Springer Science & Business Media, 2013.

[88] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 2012, pp. 87–88.

[89] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *The Knowledge Engineering Review*, vol. 28, no. 1, pp. 75–105, 2013.

[90] D. Jiang and J. Pei, "Mining frequent cross-graph quasi-cliques," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2, no. 4, pp. 1–42, 2009.

[91] A. Joshi, Y. Zhang, P. Bogdanov, and J.-H. Hwang, "An efficient system for subgraph discovery," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 703–712.

[92] S.-W. Jun, A. Wright, S. Zhang, and S. Xu, "Grafboost: Using accelerated flash storage for external graph analytics," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 411–424.

[93] V. Kalavri, V. Vlassov, and S. Haridi, "High-level programming abstractions for distributed graph processing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 2, pp. 305–324, 2017.

[94] O. Kalinsky, B. Kimelfeld, and Y. Etsion, "The triejax architecture: Accelerating graph operations through relational joins," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1217–1231.

[95] N. Kapre, "Custom fpga-based soft-processors for sparse graph acceleration," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2015, pp. 9–16.

[96] N. Kapre, N. Mehta, D. Rizzo, I. Eslick, R. Rubin, T. E. Uribe, F. Thomas Jr, and A. DeHon, "Graphstep: A system architecture for sparse-graph algorithms," in *Field-Programmable Custom Computing Machines, 2006. FCCM'06. 14th Annual IEEE Symposium on*. IEEE, 2006, pp. 143–151.

[97] J. Kepner, P. Aaltonen, D. Bader, A. Buluç, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, and H. Meyerhenke, "Mathematical foundations of the graphblas," in *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*. IEEE, 2016, pp. 1–9.

[98] A. Khan, "Vertex-centric graph processing: The good, the bad, and the ugly," *arXiv preprint arXiv:1612.07404*, 2016.

[99] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, "K-core decomposition of large networks on a single pc," *Proceedings of the VLDB Endowment*, vol. 9, no. 1, pp. 13–23, 2015.

[100] S. Ko and W.-S. Han, "Turbograph++: A scalable and fast graph analytics system," in *Proceedings of the 2018 International*

Conference on Management of Data. ACM, 2018, pp. 395–410.

[101] D. Lavenier, J.-F. Roy, and D. Furodet, "Dna mapping using processor-in-memory architecture," in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2016, pp. 1429–1435.

[102] J. Lee, H. Kim, S. Yoo, K. Choi, H. P. Hofstee, G.-J. Nam, M. R. Nutter, and D. Jamsek, "Extrav: boosting graph processing near storage with a coherent accelerator," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1706–1717, 2017.

[103] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal, "A survey of algorithms for dense subgraph discovery," in *Managing and Mining Graph Data*. Springer, 2010, pp. 303–336.

[104] E. A. Leicht, P. Holme, and M. E. Newman, "Vertex similarity in networks," *Physical Review E*, vol. 73, no. 2, p. 026120, 2006.

[105] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 985–1042, 2010.

[106] G. Li, G. Dai, S. Li, Y. Wang, and Y. Xie, "Graphia: an in-situ accelerator for large-scale graph processing," in *Proceedings of the International Symposium on Memory Systems*, 2018, pp. 79–84.

[107] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 288–301.

[108] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[109] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[110] S. Liu and A. Khan, "An empirical analysis on expressibility of vertex centric graph processing paradigm," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 242–251.

[111] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–34, 2018.

[112] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *ACM SIGARCH computer architecture news*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 453–464.

[113] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[114] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *Acm sigplan notices*, vol. 40, no. 6, pp. 190–200, 2005.

[115] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. W. Berry, "Challenges in Parallel Graph Processing," *Par. Proc. Let.*, vol. 17, no. 1, pp. 5–20, 2007.

[116] X. Ma, D. Zhang, and D. Chiou, "Fpga-accelerated transactional execution of graph workloads," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 227–236.

[117] 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 *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[118] J. Malicevic, B. Lepers, and W. Zwaenepoel, "Everything you always wanted to know about multicore graph processing but were afraid to ask," in *2017 USENIX Annual Technical Conference (USENIX ATC'17)*, 2017, pp. 631–643.

[119] K. K. Matam, G. Koo, H. Zha, H.-W. Tseng, and M. Annavaram, "Graphssd: graph semantics aware ssd," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 116–128.

[120] D. W. Matula and L. L. Beck, "Smallest-last ordering and clustering and graph coloring algorithms," *JACM*, 1983.

[121] D. Mawhirter, S. Reinehr, C. Holmes, T. Liu, and B. Wu, "Graphzero: Breaking symmetry for efficient graph mining," *arXiv preprint arXiv:1911.12877*, 2019.

[122] D. Mawhirter and B. Wu, "Automine: harmonizing high-level abstraction and high performance for graph mining," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. ACM, 2019, pp. 509–523.

[123] R. R. McCune, T. Weninger, and G. Madey, "Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, p. 25, 2015.

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

[125] G. L. Miller, R. Peng, A. Vladu, and S. C. Xu, "Improved parallel algorithms for spanners and hopsets," in *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 2015, pp. 192–201.

[126] S. Mittal, J. S. Vetter, and D. Li, "Improving energy efficiency of embedded dram caches for high-end computing systems," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, 2014, pp. 99–110.

[127] A. Mukkara, N. Beckmann, M. Abeydeera, X. Ma, and D. Sanchez, "Exploiting locality in graph analytics through hardware-accelerated traversal scheduling," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 1–14.

[128] O. Mutlu *et al.*, "Processing Data Where It Makes Sense: Enabling In-Memory Computation," *MicPro*, 2019.

[129] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A modern primer on processing in memory," *arXiv preprint arXiv:2012.03112*, 2020.

[130] A. Nag, C. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, "Gencache: Leveraging in-cache operators for efficient sequence alignment," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 334–346.

[131] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "Graphpim: Enabling instruction-level pim offloading in graph computing frameworks," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 457–468.

[132] Neo4j, Inc., "The neo4j graph algorithms user guide v3.5," https://neo4j.com/docs/graph-algorithms/current, 2019.

[133] E. Nurvitadhi, G. Weisz, Y. Wang, S. Hurkat, M. Nguyen, J. C. Hoe, J. F. Martínez, and C. Guestrin, "Graphgen: An fpga framework for vertex-centric graph computation," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2014, pp. 25–28.

[134] T. Oguntebi and K. Olukotun, "Graphops: A dataflow library for graph analytics acceleration," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 111–117. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847337

[135] M. M. Ozdal, S. Yesil, T. Kim, A. Ayupov, J. Greth, S. Burns, and O. Ozturk, "Energy efficient architecture for graph analytics accelerators," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 166–177.

[136] S. Pal, J. Beaumont, D.-H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H.-S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "Outerspace: An outer product based sparse matrix multiplication accelerator," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 724–736.

[137] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, and M. Méndez-Lojo, "The tao of parallelism in algorithms," in *ACM Sigplan Notices*, vol. 46, no. 6. ACM, 2011, pp. 12–25.

[138] T. Ramraj and R. Prabhakar, "Frequent subgraph mining algorithms-a survey," *Procedia Computer Science*, vol. 47, pp. 197–204, 2015.

[139] S. U. Rehman, A. U. Khan, and S. Fong, "Graph mining: A survey of

graph mining techniques," in *Seventh International Conference on Digital Information Management (ICDIM 2012)*. IEEE, 2012, pp. 88–92.

[140] N. Rhodes, P. Willett, A. Calvet, J. B. Dunbar, and C. Humblet, "Clip: similarity searching of 3d databases using clique detection," *Journal of chemical information and computer sciences*, vol. 43, no. 2, pp. 443–448, 2003.

[141] P. Ribeiro, P. Paredes, M. E. Silva, D. Aparicio, and F. Silva, "A survey on subgraph counting: Concepts, algorithms and applications to network motifs and graphlets," *arXiv preprint arXiv:1910.13011*, 2019.

[142] I. Robinson, J. Webber, and E. Eifrem, *Graph databases*. " O'Reilly Media, Inc.", 2013.

[143] R. A. Rossi and N. K. Ahmed, "An interactive data repository with visual analytics," *SIGKDD Explor.*, vol. 17, no. 2, pp. 37–41, 2016. [Online]. Available: http://networkrepository.com

[144] R. A. Rossi and N. K. Ahmed, "An interactive data repository with visual analytics," *ACM SIGKDD Explorations Newsletter*, vol. 17, no. 2, pp. 37–41, 2016.

[145] 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*. ACM, 2013, pp. 472–488.

[146] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. Aref, M. Arenas, M. Besta, P. A. Boncz *et al.*, "The future is big graphs! a community view on graph processing systems," *arXiv preprint arXiv:2012.06171*, 2020.

[147] S. Salihoglu and J. Widom, "Optimizing graph algorithms on Pregel-like systems," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 577–588, 2014.

[148] S. E. Schaeffer, "Graph clustering," *Computer science review*, vol. 1, no. 1, pp. 27–64, 2007.

[149] T. Schank, "Algorithmic aspects of triangle-based network analysis," *Phd in computer science, University Karlsruhe*, vol. 3, 2007.

[150] P. Schmid, M. Besta, and T. Hoefler, "High-performance distributed rma locks," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, 2016, pp. 19–30.

[151] H. Schweizer, M. Besta, and T. Hoefler, "Evaluating the cost of atomic operations on modern architectures," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 2015, pp. 445–456.

[152] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, and M. A. Kozuch, "Rowclone: fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 185–197.

[153] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 273–287.

[154] X. Shi, Z. Zheng, Y. Zhou, H. Jin, L. He, B. Liu, and Q.-S. Hua, "Graph processing on gpus: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 81, 2018.

[155] Y. Shiloach and U. Vishkin, "An o (log n) parallel connectivity algorithm," Computer Science Department, Technion, Tech. Rep., 1980.

[156] Y. Shiloach and U. Vishkin, "An o (logn) parallel connectivity algorithm," *Journal of Algorithms*, vol. 3, no. 1, pp. 57–67, 1982.

[157] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *ACM SIGPLAN Notices*, vol. 48, no. 8, 2013, pp. 135–146.

[158] J. Shun and K. Tangwongsan, "Multicore triangle computations without tuning," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 2015, pp. 149–160.

[159] S. G. Singapura, A. Srivastava, R. Kannan, and V. K. Prasanna, "Oscar: Optimizing scratchpad reuse for graph processing," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–7.

[160] S. Skiena, "Dijkstra's algorithm," *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pp. 225–227, 1990.

[161] E. Solomonik, M. Besta, F. Vella, and T. Hoefler, "Scaling betweenness centrality using communication-efficient sparse matrix multiplication," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 47.

[162] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating graph processing using ReRAM," in *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 531–543.

[163] V. Spirin and L. A. Mirny, "Protein complexes and functional modules in molecular networks," *Proceedings of the National Academy of Sciences*, vol. 100, no. 21, pp. 12 123–12 128, 2003.

[164] 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.

[165] M. Sutton, T. Ben-Nun, and A. Barak, "Optimizing parallel graph connectivity computation via subgraph sampling."

[166] I. Takigawa and H. Mamitsuka, "Graph mining: procedure, application to drug discovery and recent advances," *Drug discovery today*, vol. 18, no. 1-2, pp. 50–57, 2013.

[167] L. Tang and H. Liu, "Graph mining applications to social network analysis," in *Managing and Mining Graph Data*. Springer, 2010, pp. 487–513.

[168] B. Taskar, M.-F. Wong, P. Abbeel, and D. Koller, "Link prediction in relational data," in *Advances in neural information processing systems*, 2004, pp. 659–666.

[169] A. Tate, A. Kamil, A. Dubey, A. Größlinger, B. Chamberlain, B. Goglin, C. Edwards, C. J. Newburn, D. Padua, D. Unat *et al.*, "Programming abstractions for data locality." PADAL Workshop 2014, April 28–29, Swiss National Supercomputing Center . . . , 2014.

[170] C. H. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: a system for distributed graph mining," in *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015, pp. 425–440.

[171] S. Thiprungsri and M. A. Vasarhelyi, "Cluster analysis for anomaly detection in accounting data: An audit approach." *International Journal of Digital Accounting Research*, vol. 11, 2011.

[172] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theor. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, 2006. [Online]. Available: https://doi.org/10.1016/j.tcs.2006.06.015

[173] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, pp. 31–42, 1976.

[174] K. Van Craeynest, S. Akram, W. Heirman, A. Jaleel, and L. Eeckhout, "Fairness-aware scheduling on single-isa heterogeneous multi-cores," in *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*. IEEE, 2013, pp. 177–187.

[175] C. Wang, L. Gong, F. Jia, and Z. Xuehai, "An fpga based accelerator for ubiquitous clustering applications with custom instructions," *IEEE Transactions on Computers*, 2020.

[176] K. Wang, Z. Zuo, J. Thorpe, T. Q. Nguyen, and G. H. Xu, "Rstream: marrying relational algebra with streaming for efficient graph mining on a single machine," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 763–782.

[177] L. Wang, K. Hu, and Y. Tang, "Robustness of link-prediction algorithm based on similarity and application to biological networks," *Current Bioinformatics*, vol. 9, no. 3, pp. 246–252, 2014.

[178] T. Washio and H. Motoda, "State of the art of graph-based data mining," *Acm Sigkdd Explorations Newsletter*, vol. 5, no. 1, pp. 59–68, 2003.

[179] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

[180] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The risc-v instruction set manual, volume i: Base user-level isa," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011.

[181] A. S. Waterman, "Design of the risc-v instruction set architecture," Ph.D. dissertation, UC Berkeley, 2016.

[182] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[183] X. Xin, Y. Zhang, and J. Yang, "Elp2im: Efficient and low power bitwise operation processing in dram," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 303–314.

[184] C. Xu, C. Wang, L. Gong, L. Jin, X. Li, and X. Zhou, "Domino: Graph processing services on energy-efficient hardware accelerator," in *2018 IEEE International Conference on Web Services (ICWS)*. IEEE, 2018, pp. 274–281.

[185] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[186] D. Yan, H. Chen, J. Cheng, M. T. Özsu, Q. Zhang, and J. Lui, "G-thinker: big graph mining made easier and faster," *arXiv preprint arXiv:1709.03110*, 2017.

[187] D. Yan, J. Cheng, K. Xing, Y. Lu, W. Ng, and Y. Bu, "Pregel algorithms for graph connectivity problems with performance guarantees," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 1821–1832, 2014.

[188] D. Yan, W. Qu, G. Guo, and X. Wang, "Prefixfpm: A parallel framework for general-purpose frequent pattern mining," in *Proceedings of the 36th IEEE International Conference on Data Engineering (ICDE) 2020*, 2020.

[189] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 15–29.

[190] C. Yang, "An efficient dispatcher for large scale graphprocessing on opencl-based fpgas," *arXiv preprint arXiv:1806.11509*, 2018.

[191] Y. Yang, Z. Li, Y. Deng, Z. Liu, S. Yin, S. Wei, and L. Liu, "Graphabcd: Scaling out graph analytics with asynchronous block coordinate descent," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 419–432.

[192] P. Yao, "An efficient graph accelerator with parallel data conflict management," *arXiv preprint arXiv:1806.00751*, 2018.

[193] P. Yao, L. Zheng, Z. Zeng, Y. Huang, C. Gui, X. Liao, H. Jin, and J. Xue, "A locality-aware energy-efficient accelerator for graph mining applications."

[194] P. Yao, L. Zheng, Z. Zeng, Y. Huang, C. Gui, X. Liao, H. Jin, and J. Xue, "A locality-aware energy-efficient accelerator for graph mining applications," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 895–907.

[195] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "Imp: Indirect memory prefetcher," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 178–190.

[196] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "GraphP: Reducing Communication for PIM-based Graph Processing with Efficient Data Partition," in *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 544–557.

[197] Y. Zhang, F. N. Abu-Khzam, N. E. Baldwin, E. J. Chesler, M. A. Langston, and N. F. Samatova, "Genome-scale computational approaches to memory-intensive applications in systems biology," in *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. IEEE, 2005, pp. 12–12.

[198] C. Zhao, Z. Zhang, P. Xu, T. Zheng, and X. Cheng, "Kaleido: An efficient out-of-core graph mining system on a single machine," *arXiv preprint arXiv:1905.09572*, 2019.

[199] K. Zhao and J. X. Yu, "All-in-one: Graph processing in rdbmss revisited," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1165–1180.

[200] L. Zheng, J. Zhao, Y. Huang, Q. Wang, Z. Zeng, J. Xue, X. Liao, and H. Jin, "Spara: An energy-efficient reram-based accelerator for sparse graph analytics applications," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 696–707.

[201] M. Zhou, M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Gram: graph processing in a reram-based computational memory." in *ASP-DAC*, 2019, pp. 591–596.

[202] S. Zhou, C. Chelmis, and V. K. Prasanna, "Optimizing memory performance for fpga implementation of pagerank." in *ReConFig*, 2015, pp. 1–6.

[203] S. Zhou, C. Chelmis, and V. K. Prasanna, "High-throughput and energy-efficient graph processing on fpga," in *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*. IEEE, 2016, pp. 103–110.

[204] S. Zhou, R. Kannan, H. Zeng, and V. K. Prasanna, "An fpga framework for edge-centric graph processing," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*. ACM, 2018, pp. 69–77.

[205] S. Zhou and V. K. Prasanna, "Accelerating graph analytics on cpu-fpga heterogeneous platform," in *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2017, pp. 137–144.

[206] X. Zhou and T. Nishizeki, "Edge-coloring and f-coloring for various classes of graphs," *J. Graph Algorithms Appl.*, vol. 3, no. 1, pp. 1–18, 1999. [Online]. Available: https://doi.org/10.7155/jgaa.00012

[207] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti, "A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing," in *2013 IEEE international 3D systems integration conference (3DIC)*. IEEE, 2013, pp. 1–7.

[208] Q. Zhu, T. Graf, H. E. Sumbul, L. Pileggi, and F. Franchetti, "Accelerating sparse matrix-matrix multiplication with 3d-stacked logic-in-memory hardware," in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2013, pp. 1–6.

[209] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian, "Graphq: Scalable pim-based graph processing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 712–725.