SHARP: Software Hint-Assisted Memory Access Prediction for Graph Analytics

Pengmiao Zhang
University of Southern California
Los Angeles, USA
pengmiao@usc.edu

Rajgopal Kannan
US Army Research Lab-West
Los Angeles, USA
rajgopal.kannan.civ@army.mil

Xiangzhi Tong
Xi'an Jiaotong-Liverpool University
Suzhou, China
Xiangzhi.Tong18@student.xjtlu.edu.cn

Anant V. Nori
Processor Architecture Research Lab
Intel Labs
Bangalore, India
anant.v.nori@intel.com

Viktor K. Prasanna
University of Southern California
Los Angeles, USA
prasanna@usc.edu

***Abstract*—With the rise of large-scale graph analytics, graph processing is increasingly bottlenecked by memory performance. Data prefetching is a technique that can hide memory latency, which relies on accurate prediction of memory accesses. While recent machine learning approaches have performed well on memory access prediction, they are restricted to building general models, ignoring the shift of memory access patterns following the change of processing phases in software. Therefore, we propose SHARP: a novel Software Hint-Assisted memoRy access Prediction approach for graph analytics under Scatter-Gather paradigm on multi-core shared-memory machines. We introduce software hints, generated from programmer insertion, that explicitly indicate the processing phase of a graph processing program, i.e., scatter or gather. Assisted by the software hints, we develop phase-specific prediction models that use attention-based neural networks, trained by memory traces with rich context information. We use three widely-used graph algorithms and a variety of datasets for evaluation. With regard to F1-score, SHARP outperforms the widely-used model Delta-LSTM by 16.45%–18.93% for the scatter phase and 9.50%–22.25% for the gather phase, and outperforms the state-of-the-art model TransFetch by 3.66%–7.48% for the scatter phase and 2.69%–7.59% for the gather phase.**

***Index Terms*—memory access prediction, graph analytics, software hint, attention mechanism**

I. Introduction

Graphs are widely used structures describing network data in multiple scientific and engineering domains, such as social media, bioinformatics, and transportation. *Graph Analytics* are analytic tools used to determine the strength and direction of relationships between objects in a graph. With the rise of big data, graph analytics offer high potential in exploring the relationship between objects because of the virtue of graphs in explicitly representing relations through a set of entities (vertices) and their inter-connections (edges) [1].

Graph processing is typically bounded by memory latency due to the "memory wall" problem [2], [3]. Though prior works have proposed using distributed frameworks to process very large graphs on clusters [4], [5], the high communication overheads hinder the performance of clusters [6]. With the advance in DDR capacity allowing large graphs to fit in the main memory of a single server, many frameworks have been developed for high-performance graph analytics on *shared-memory machines* [7]–[10]. However, arbitrary data accesses and updates in the shared-memory machine cause the cores to stall waiting for data fetch from DRAM. Algorithm-level optimizations have been proposed in prior works to improve memory efficiency, such as block propagation [11] and partition-centric paradigm [10], [12].

Data prefetching is a technique that hides memory access latency by predicting future data accesses and brings the data closer (in a hierarchical memory system) to the processor before it is requested [13]–[15]. Effective prefetching depends on accurate *memory access prediction*. Traditional prefetchers use pre-defined rules, based on heuristic patterns [16], [17], spatial locality [18]–[21] or temporal locality of references [22], [23], to predict future accesses. These methods suffer from low adaptability and low generalizability [24], [25], making them not powerful enough to speed up the increasingly complex memory access patterns from graph analytics algorithms.

Machine Learning (ML) algorithms have entered the domain of memory access prediction, offering insight into data prefetching. Rahman et al. [26] use logistic regression and decision tree for pattern classification. Long Short-Term Memory (LSTM) model is widely used due to its high performance in sequence modeling [25], [27]–[29]. The combination of LSTM with existing prefetchers [30], meta-learning [31], and attention mechanism [32] are also explored in prior work. Recently, Zhang et al. [33] proposed TransFetch that uses fine-grained memory address input and an attention-based model for multi-label memory access prediction, achieving state-of-the-art performance.

Existing ML-based memory access prediction approaches are restricted to building general models, ignoring the change of memory access patterns caused by the shift of graph processing phases. For example, graph analytics algorithms programmed under *Scatter-Gather* paradigm [10], [12], [34] have two distinct phases for each iteration: *scatter* for propagating the current value of a vertex to its neighbors along edges, and *gather* for accumulating values from neighbors to compute the next value of a vertex. The memory access patterns are notably different for the two phases, which challenges the training of a general model [35]. Moreover, context information including thread number and read/write operations have not been fully exploited by prior work. By extracting and utilizing the phase and context information, we aim to build a memory access prediction model that can achieve higher prediction performance.

We propose SHARP, a novel software hint-assisted memory access prediction approach for graph analytics under Scatter-Gather paradigm for multi-core shared-memory machines. First, we introduce software hints, generated from programmer insertion, that explicitly and precisely indicate the processing phase for a graph processing program. According to the software hint regarding phases, two phase-specific models are developed for scatter and gather phases respectively. For inference, a simple switch controls the model selection triggered by software hints. Second, we generate context-rich memory traces. Context information including program counter, page distance, thread ID, and read/write operation are extracted along with memory addresses. Third, we train attention-based delta predictors as the phase-specific models to fit the mapping between context-rich memory address sequence and future address deltas (difference from future addresses to the current address). SHARP can be easily extended to other paradigms [36] and graph machine learning [37] with multiple processing phases.

Our contribution can be summarized as follows:

- We introduce software hints that are indicators of phases in a graph processing program using Scatter-Gather paradigm. According to the software hints, we analyze the memory accesses for the two phases and demonstrate the notable pattern differences.
- We propose SHARP, a novel approach to predict memory accesses assisted by the software hints. SHARP uses two phase-specific attention-based models, trained by context-rich memory traces of the corresponding phase, and selected by the software hints for inference.
- We evaluate SHARP using three popular graph algorithms on a variety of datasets. with regard to F1-score, SHARP outperforms the widely-used model Delta-LSTM by 16.45%–18.93% for the scatter phase and 9.50%–22.25% for the gather phase, outperforms state-of-the-art TransFetch by 3.66%–7.48% for the scatter phase and 2.69%–7.59% for the gather phase.

II. Background

A. Scatter-Gather Graph Processing Paradigm

Scatter-Gather is a widely-used parallel programming paradigm for graph processing [10], [38], [39]. Scatter-Gather decouples the message sending from the message collection and state update [40]. As is shown in Figure 1, for each iteration, a graph processing algorithm under vertex-centric Scatter-Gather paradigm operates in two phases:

1) **Scatter**: a vertex propagate updates (the current state of the vertex) to its neighbors along edge.
2) **Gather**: a vertex accumulate updates (states of neighbors) to compute the next state of the vertex.

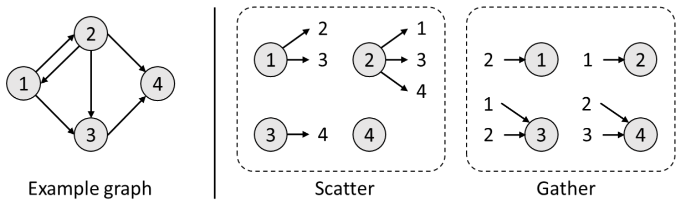


Fig. 1: Graph processing in Scatter-Gather paradigm.

A *Partition-Centric Scatter-Gather* paradigm is proposed in Graph Processing Over Partitions (GPOP) [10] by generalizing the vertices in the vertex-centric Scatter-Gather paradigm to parts of a graph. Assuming the entire graph can fit into the main memory, the graph is partitioned to fit better in caches. GPOP ensures high locality of memory reference and improves memory efficiency for graph processing from algorithm perspective, which offers opportunity for memory access prediction from architecture perspective.

Existing methods develop one general model for the entire application running period, ignoring the pattern differences of the phases in graph processing. We extract the phase information from software and exploit the information for developing higher-performing memory prediction models.

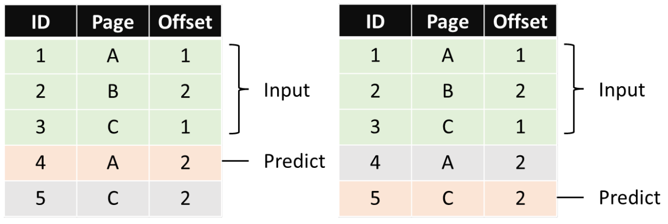
B. ML for Memory Access Prediction

Memory access prediction is a task that exploiting the correlation between history memory accesses, detecting the memory access patterns, and predicting future memory access addresses, which is essential for effective data prefetching.

While traditional methods for memory access prediction use table-based predictors exploiting spatial or temporal locality of memory references [19], [20], [22], [23], [41], Machine Learning (ML) algorithms have shown advantage in a data-driven approach [25].

A general problem formulation for the ML-based modeling of memory access prediction is: Let $X_t = \{x_1, x_2, ..., x_N\}$ be the sequence of N history memory addresses at time t; let $Y_t = \{y_1, y_2, ..., y_k\}$ be a set of k outputs that will be accessed in the future; a ML model can be trained to approximate $P(Y_t|X_t)$, the probability that the future addresses Y_t will be accessed given the history events X_t.

While most existing ML models use past memory access sequence as the model input, there are two settings for the output future accesses, as is shown in Figure 2. First, Hashemi et al. [42] and Srivastava et al. [24], [43] predicts the globally next memory access, as is shown in Figure 2a. While this setting allows entire address space prediction, the interleave of patterns hinders the prediction performance. Second, Zhang et al. [33], [44] proposes predicting next memory access within



ID	Page	Offset
1	A	1
2	B	2
3	C	1
4	A	2
5	C	2

Input
Predict

(a) Predict the next access globally

ID	Page	Offset
1	A	1
2	B	2
3	C	1
4	A	2
5	C	2

Input
Predict

(b) Predict within a page range

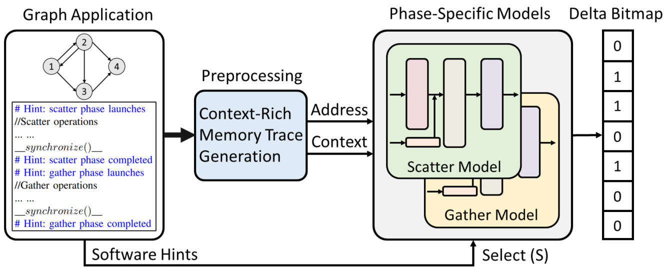
Fig. 2: Two settings for memory access prediction.

a spatial range to the current address, as is shown in Figure 2b with the spatial range as one page, which have shown state-of-the-art performance by predicting more trackable patterns. In this work we use the second setting by predicting the memory access within a spatial range.

III. Approach

A. *Overall Design*

Figure 3 shows the overview of SHARP. First, software hints indicating the phases in graph processing are generated from a graph application (see Section III-B). According to the software hints, a phase-specific model for the scatter or gather phase is selected. The models are trained by context-rich memory traces, using memory address sequences and the corresponding context information as input (see Section III-C). The models are based on attention mechanism and predict future memory address deltas in the format of bitmap as output (see Section III-D). By adding the current address and the predicted future deltas, SHARP can predict multiple future memory accesses for one inference.



```
# Hint: scatter phase launches
//Scatter operations
... ...
__synchronize()__
# Hint: scatter phase completed
# Hint: gather phase launches
//Gather operations
... ...
__synchronize()__
# Hint: gather phase completed
```

Fig. 3: Overall design of SHARP.

B. *Software Hints*

We introduce software hints that serve as indicators of graph processing phases in a program. The software can be generated from either programmer insertion or profiling [45].

We illustrate the software hints from programmer insertion using the partition-centric Scatter-Gather paradigm proposed by GPOP [10], as is described in Algorithm 1. For a given graph $G(V, E)$ in the format of Compressed Sparse Row (CSR), let $N_i(v)$ denotes the in-neighbors of vertex v and $N_o(v)$ denotes the out-neighbors of vertex v. The graph is

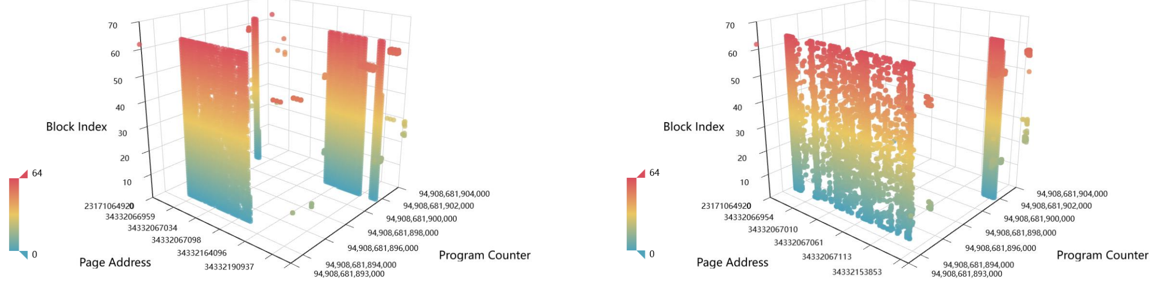
Algorithm 1 An interation of Partition-Centric Scatter-Gather with Programmer Insertion of Software Hints

```
1:  $P$: set or partitions
2:  # Hint: scatter phase launches
3:  for $p \in P$ do in parallel                    ▷ Scatter
4:      for all $v \in p$ do
5:          for all $q \in N_o[v]$ do
6:              $bin[p][q] \leftarrow scatterFunc(v)$
7:          end for
8:      end for
9:      for all $v \in p$ do
10:         $val[v] \leftarrow initFunc(v)$
11:     end for
12: $\_\_synchronize()\_\_$
13: # Hint: scatter phase completed
14: # Hint: gather phase launches
15: for $p \in P$ do in parallel                    ▷ Gather
16:     for all $msg \in bin[:][p]$ do
17:         for all $v \in msg.id$ do
18:             $val[v] \leftarrow gatherFunc(val[v], v)$
19:         end for
20:     end for
21:     for all $v \in p$ do
22:         $val[v] \leftarrow filterFunc(v)$
23:     end for
24: $\_\_synchronize()\_\_$
25: # Hint: gather phase completed
```

partitioned to P parts for partition-centric processing. For one iteration, all the partitions will be processed in parallel twice: in the scatter phase and in the gather phase. In the scatter phase, the vertex data will be updated to bins that store the destination vertex IDs, defined in $scatterFunc$. Then, the current vertex values will be reinitialized for gathering using $initFunc$. In the gather phase, a partition will read the messages from the corresponding bins to update vertex values using $gatherFunc$. Then, a $filterFunc$ is defined to process the vertex value. We insert hints in the software code to label the launching and completing of the scatter and gather phases, as is shown in Algorithm 1 Line 2, Line 13-14, and Line 25.

Due to the two separate traversals of the input graph, the memory access patterns of scatter phase and gather phase are dramatically different. We visualize the memory access patterns of PageRank under graph *Wikipedia hyperlinks* [46] in Figure 4 using 3D scatter plots. We extract the first 40K memory accesses following the launching of a phase, triggered by the software hints. The dimensions in the plots are the page address, the block index, and the program counter. The comparison between the scatter phase memory access pattern (Figure 4a) and the gather phase memory access pattern (Figure 4b) demonstrates the pattern differences in program counters, frequently accessed pages, and the distribution of accessed blocks.

The pattern differences between the two phases motivates



(a) Scatter (b) Gather

Fig. 4: Different memory access patterns between scatter and gather phases in graph processing. Three dimensions are page address, program counter, and block index, where block index is the page offset shifted by block offset considering memory accesses are operated in the unit of a block.

the design of phase-specific models. Instead of one general prediction models for the entire process like existing methods, we use two separate models, each trained specifically for one operation phase, i.e., scatter or gather. In prediction, the selection of model is controlled by the software hints.

C. Context-Rich Memory Trace

The context enhancement effect by program counter (PC) and page distance for memory access prediction has been proved in previous work [33]. For graph applications on multi-core shared-memory machines, we extract more context information for memory access prediction, which formulates context-rich memory trace, including features:

- Memory access address. Our approach works on virtual address space, each address is processed using fine-grained address segmentation method [33]. Using the size of page offset, we split one address into a vector of segments for further model input.
- Program counter (PC). PC value is compressed using a folding method hash function and then normalized using the range of the hash function.
- Page distance (PD). For a sequence of memory access of prediction model input, the page address distance (absolute difference) from history accesses to the current address indicates the spatial locality. The inversion of page distance is injected to the model for facilitating model training.
- Thread ID (TH). Our approach is designed for multi-threaded shared-memory parallel graph applications. The shared-memory accesses are requested from interleaved threads. Therefore, we apply thread ID for an access to enhance access prediction. The thread ID need to be normalized using the total number of threads as preprocessing.
- Read/Write (RW). The direction of memory access, including read and write, can provide more precise information compared with program counter. This feature is binary and can be directly fed into a prediction model.

A summary of the features for the context-rich memory trace is shown in Table I.

TABLE I: Context-Rich Memory Trace

Feature	Description	Type
Address	Memory address processed by segmentation	Vector[Int]
PC	Program counter processed by hashing	Float
PD	Inversion of page distance	Float
TH	The thread ID that request the memory access	Int
RW	Memory access direction: read or write	Binary

D. Attention-Based Delta Predictor

We build attention-based memory access delta predictors as the phase-specific models, as is shown in Figure 5. The scatter model and the gather model have the same structure, but trained independently using context-rich memory traces for the corresponding phases only. We describe the input, output, structure, and loss function of a predictor in detail in this section.

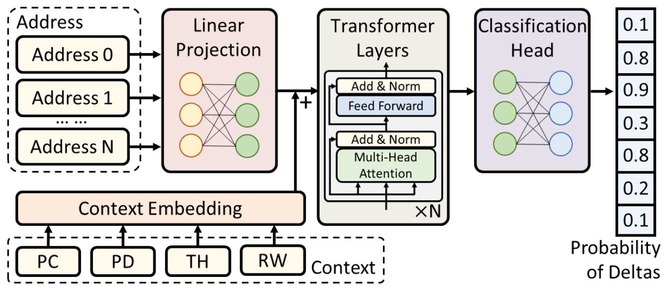


Fig. 5: The structure of an attention-based delta predictor, serving as the phase-specific model in SHARP.

1) Input Features: The model input includes two categories: memory addresses and the context information, both from a window of history N accesses. The segmented memory addresses are flattened and fed into a linear layer for further processing. The context information, including PC, PD, TH, and RW, as processed in Table I, are concatenated and embedded first, then injected into the model as side-information through addition operation.

2) Transformer Layers: We apply Transformer [47] to learn the latent patterns from the input features. Transformer is an attention-based neural network that has shown high performance in natural language processing [48] and computer vision [49]. One transformer layer is made up by a multi-head attention (MSA), a point-wise feed-forward network (FFN), their perspective residual connections and normalizations, as is shown in Figure 5 and Equation 1–4.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

$$\text{head}_\text{i} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{2}$$

$$\text{MSA}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_\text{H}\right)W^O \tag{3}$$

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right)W_2 + b_2 \tag{4}$$

where Q represents the queries; K the keys; V the values; d the dimension of layer input; W_i^Q, W_i^K, W_i^V, and W^O the projection matrices for multi-head attention; H the number of heads; W_1, W_2, b_1, and b_2 are the projection matrices and biases for the feed-forward network.

3) Multi-Label Classification: After the latent features are extracted from the Transformer layers, a multi-layer perceptron is designed as a classification head. The model is trained using deltas in the format of bitmap as labels. The labels are first collected from future memory address deltas (address difference from the future access and the current access) within a spatial range, then mapped to a bitmap with 1 indicating the appearance of the corresponding delta. The independent and multiple labels in bitmap formulate the problem as multi-label classification. We use binary cross-entropy loss to train the model:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log\left(p\left(y_i\right)\right) + \left(1 - y_i\right)\log\left(1 - p\left(y_i\right)\right) \tag{5}$$

where y_i is the label and $p(y_i)$ is the predicted probability for sample i being 1, which indicates the probability of a delta, as is shown in the model output of Figure 5.

IV. Experiments

A. Experimental Setup

1) Datasets: We evaluate SHARP and the baselines using 6 real world graph datasets [50] from a variety of domains, as is summarized in Table II. The graphs are represented in compressed sparse row (CSR) format for algorithm processing.

TABLE II: Graph Datasets

Dataset	Description	# Vertices	# Edges
amazon [51]	Purchase networks	0.26 M	1.23 M
google [52]	Google web graph	0.88 M	5.11 M
roadCA [52]	California Road network	1.96 M	2.76 M
soclj [52]	LiveJournal social network	4.84 M	68.99 M
wiki [46]	Wikipedia hyperlinks	1.79 M	28.51 M
youtube [53]	Youtube social network	1.13 M	2.99 M

2) Algorithms: We implemented three widely used graph processing algorithms based on GPOP framework [10]:

- **Breadth-First Search (BFS)** – A graph search and traverse algorithm that starts from a source node and traverses the graph in a level-based approach.
- **Connected Components (CC)** – Labels connected components (path exists between two nodes) in a graph, implemented using Label Propagation method [54].
- **PageRank (PR)** – A node ranking algorithm that determines the "popularity" of nodes in a graph, proposed for the sorting of web search results [55].

The implemented algorithms are based on Scatter-Gather paradigm and hints indicating the processing phases are inserted in the software.

3) Trace Generation: We develop a Pintool using Intel Pin [56] to obtain memory access traces. Our Pintool detects the memory access operations and records the corresponding context information. The Pintool can also detect the inserted software hints and record the current operation phase of a memory access.

B. Metrics

We use F1-score to evaluate the memory access prediction performance. F1-score is a weighted average of precision and recall [58], defined as Equation 6–8:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{6}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} \ + \text{False Negative}} \tag{7}$$

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

C. Baselines

We compare SHARP with state-of-the-art models for memory access prediction in prior works. Agnostic to the phases in graph processing, the baseline models are trained using the same memory traces and labels (delta bitmap):

- **Delta-LSTM** [25] – Consecutive address deltas are calculated as the LSTM input, recurrently process the input sequence.
- **TransFetch** [33] – Addresses are segmented as the input for an attention-based model. Context with program counter and page distance are incorporated.

D. Training and Evaluation

Based on the software hints on phases, we split the memory trace for an application into four sets with equal lengths: scatter training, scatter testing, gather training, and gather testing. For baselines, we use the mix of the scatter training set and gather training set for model training. For SHARP, we train two models using the two training sets, respectively. All models are evaluated using the two testing sets.

TABLE III: F1-Score of SHARP and Baselines

Algorithm	Phase	Model	Graph						Average
			amazon	**google**	**roadCA**	**soclj**	**wiki**	**youtube**	
BFS	Scatter	Delta-LSTM [57]	0.7454	0.6953	0.6814	0.7507	0.7177	0.9194	0.7545
		TransFetch [33]	0.8314	0.7769	0.8191	0.8634	0.8156	0.9652	0.8465
		SHARP	**0.8851**	**0.8042**	**0.8724**	**0.8856**	**0.8544**	**0.9678**	**0.8786**
	Gather	Delta-LSTM	0.6808	0.6546	0.6884	0.7254	0.7450	0.7678	0.7976
		TransFetch	0.7401	0.7533	0.8249	0.8585	0.8439	0.8936	0.8206
		SHARP	**0.8400**	**0.8138**	**0.8739**	**0.9135**	**0.8860**	**0.9074**	**0.8734**
CC	Scatter	Delta-LSTM	0.8097	0.8359	0.8164	0.8168	0.8274	0.7880	0.8161
		TransFetch	0.9628	0.9558	0.9322	0.9159	0.9325	0.9175	0.9363
		SHARP	**0.9792**	**0.9746**	**0.9720**	**0.9532**	**0.9760**	**0.9684**	**0.9706**
	Gather	Delta-LSTM	0.7721	0.7748	0.7445	0.7717	0.8072	0.7474	0.7964
		TransFetch	0.9389	0.9581	0.9484	0.9408	0.9448	0.9575	0.9481
		SHARP	**0.9726**	**0.9780**	**0.9707**	**0.9739**	**0.9789**	**0.9673**	**0.9736**
PR	Scatter	Delta-LSTM	0.8610	0.6001	0.8765	0.6930	0.6194	0.8751	0.7578
		TransFetch	0.9632	0.8594	0.9532	0.7389	0.7660	0.8781	0.8359
		SHARP	**0.9792**	**0.8662**	**0.9751**	**0.8168**	**0.8162**	**0.9313**	**0.8984**
	Gather	Delta-LSTM	0.8224	0.7974	0.7415	0.7751	0.7289	0.7501	0.7698
		TransFetch	0.8326	0.8217	0.7739	0.8381	0.8283	0.8305	0.8192
		SHARP	**0.8926**	**0.9157**	**0.9098**	**0.8949**	**0.8543**	**0.8842**	**0.8814**

TABLE IV: Effectiveness of Phase

Phase	Model	Precision	Recall	F1-Score
Scatter	Phase-agnostic	0.8968	0.8641	0.8802
	Phase-informed	0.8991	0.8790	0.8889
	Phase-specific	**0.9093**	**0.8975**	**0.9034**
Gather	Phase-agnostic	0.8772	0.8618	0.8694
	Phase-informed	0.8813	0.8780	0.8796
	Phase-specific	**0.8957**	**0.8986**	**0.8971**

TABLE V: Effectiveness of Context

Phase	Model	Precision	Recall	F1-Score
Scatter	**SHARP**	**0.9200**	**0.9131**	**0.9165**
	w/o PC	0.9133	0.9012	0.9072
	w/o TH	0.9108	0.9122	0.9115
	w/o RW	0.9187	0.9129	0.9158
	w/o PD	0.9073	0.9096	0.9084
Gather	**SHARP**	**0.9142**	**0.9049**	**0.9095**
	w/o PC	0.8991	0.9022	0.9006
	w/o TH	0.9114	0.9016	0.9065
	w/o RW	0.9017	0.902	0.9018
	w/o PD	0.9009	0.9044	0.9026

E. Results

Table III shows the F1-score of SHARP and the baselines. The average F1-score is calculated by the micro-averaged precision and recall [59]. From the table we can observe that SHARP achieves the best performance across all the algorithms and graphs. For scatter phase, SHARP outperforms Delta-LSTM by 16.45%–18.93% and outperforms TransFetch by 3.66%–7.48%. For gather phase, SHARP outperforms Delta-LSTM by 9.50%–22.25% and outperforms TransFetch by 2.69%–7.59%.

F. Ablation Study

To evaluate the effectiveness of components in SHARP, we conduct ablation studies on phase and context. Average precision, recall, and F1-score across all algorithms are reported.

1) Effectiveness of phase: Based on a base model, the attention-based predictor without the incorporation of context information, we explore the effectiveness of phases. Three models are implemented:

- **Phase-agnostic model** – The base model trained with a mix of scatter training set and gather training set.
- **Phase-informed model** – The base model with the phase information injected into the model as context.
- **Phase-specific model** – Two base models trained for the two phases using the corresponding training sets.

Results are shown in Table IV. Phase-specific models achieve the highest performance, offering 2.63% higher F1-score for scatter and 3.19% higher F1-score for gather, compared with the phase-agnostic model.

2) Effectiveness of context: Based on SHARP, we remove the components of context and show the results at Table V. Results show that each context component contributes to the model performance. PC and PD show higher influence than TH and RW. For scatter phase TH is more influential than RW while for gather phase RW shows higher influence.

V. Conclusion

We presented SHARP, a novel way to predict memory accesses for graph analytics under Scatter-Gather paradigm for multi-core shared-memory machines. The keys to our approach are using software hints to indicate the processing phases, using attention-based models to conduct phase-specific memory access prediction, and using context-rich memory traces as model input for training and inference. Compared with state-of-the-art models, SHARP achieves up to 22.25% higher F1-score than Delta-LSTM and up to 7.59% higher F1-score than TransFetch. In future work, we plan to explore the optimization of SHARP in hardware implementation and its integration into computer architecture.

Acknowledgment

This work has been supported by the U.S. National Science Foundation under grant numbers CCF-1912680 and PPoSS-2119816.

References

[1] A. Drosou, I. Kalamaras, S. Papadopoulos, and D. Tzovaras, "An enhanced graph analytics platform (gap) providing insight in big network data," *Journal of Innovation in Digital Ecosystems*, vol. 3, no. 2, pp. 83–97, 2016.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[3] C. Carvalho, "The gap between processor and memory speeds," in *Proc. of IEEE International Conference on Control and Automation*, 2002.

[4] 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.

[5] M. Han and K. Daudjee, "Giraph unchained: barrierless asynchronous parallel execution in pregel-like graph processing systems," *Proceedings of the VLDB Endowment*, vol. 8, no. 9, pp. 950–961, 2015.

[6] F. McSherry, M. Isard, and D. G. Murray, "Scalability! but at what cost?" in *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, ser. HOTOS'15. USENIX Association, 2015, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2831090.2831104

[7] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *ACM Sigplan Notices*, vol. 48. ACM, 2013, pp. 135–146.

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

[9] D. Nguyen, A. Lenharth, and K. Pingali, "A lightweight infrastructure for graph analytics," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 456–471.

[10] K. Lakhotia, R. Kannan, S. Pati, and V. Prasanna, "Gpop: A scalable cache-and memory-efficient framework for graph processing over parts," *ACM Transactions on Parallel Computing (TOPC)*, vol. 7, no. 1, pp. 1–24, 2020.

[11] S. Beamer, K. Asanović, and D. Patterson, "Reducing pagerank communication via propagation blocking," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 820–831.

[12] K. Lakhotia, R. Kannan, and V. Prasanna, "Accelerating pagerank using partition-centric processing," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, 2018.

[13] M. Dubois, M. Annavaram, and P. Stenström, *Parallel computer organization and design*. cambridge university press, 2012.

[14] S. P. Vander Wiel and D. J. Lilja, "When caches aren't enough: Data prefetching techniques," *Computer*, vol. 30, no. 7, pp. 23–30, 1997.

[15] S. Byna, Y. Chen, and X.-H. Sun, "A taxonomy of data prefetching mechanisms," in *2008 International Symposium on Parallel Architectures, Algorithms, and Networks (i-span 2008)*. IEEE, 2008, pp. 19–24.

[16] A. J. Smith, "Sequential program prefetching in memory hierarchies," *Computer*, vol. 11, no. 12, pp. 7–21, 1978.

[17] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *ACM SIGARCH Computer Architecture News*, vol. 18, no. 2SI, pp. 364–373, 1990.

[18] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, pp. 252–263, 2006.

[19] P. Michaud, "Best-offset hardware prefetching," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 469–480.

[20] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2015, pp. 141–152.

[21] J. Kim, S. H. Pugsley, P. V. Gratz, A. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[22] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 247–259.

[23] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Domino temporal data prefetcher," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 131–142.

[24] A. Srivastava, A. Lazaris, B. Brooks, R. Kannan, and V. K. Prasanna, "Predicting memory accesses: the road to compact ml-driven prefetcher," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 461–470.

[25] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," *CoRR*, vol. abs/1803.02329, 2018. [Online]. Available: http://arxiv.org/abs/1803.02329

[26] S. Rahman, M. Burtscher, Z. Zong, and A. Qasem, "Maximizing hardware prefetch effectiveness with machine learning," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, Aug. 2015, pp. 383–389.

[27] L. Peled, U. Weiser, and Y. Etsion, "A neural network memory prefetcher using semantic locality," *arXiv preprint arXiv:1804.00478*, 2018.

[28] Y. Zeng and X. Guo, "Long short term memory based hardware prefetcher: a case study," in *Proceedings of the International Symposium on Memory Systems*, 2017, pp. 305–311.

[29] P. Braun and H. Litz, "Understanding memory access patterns for prefetching," in *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA*, 2019.

[30] P. Zhang, A. Srivastava, B. Brooks, R. Kannan, and V. K. Prasanna, "Raop: Recurrent neural network augmented offset prefetcher," in *The International Symposium on Memory Systems*, 2020, pp. 352–362.

[31] P. Zhang, A. Srivastava, T.-Y. Wang, C. A. De Rose, R. Kannan, and V. K. Prasanna, "C-memmap: clustering-driven compact, adaptable, and generalizable meta-lstm models for memory access prediction," *International Journal of Data Science and Analytics*, pp. 1–14, 2021.

[32] Z. Shi, A. Jain, K. Swersky, M. Hashemi, P. Ranganathan, and C. Lin, "A hierarchical neural model of data prefetching," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 861–873.

[33] P. Zhang, A. Srivastava, A. V. Nori, R. Kannan, and V. K. Prasanna, "Fine-grained address segmentation for attention-based variable-degree prefetching," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, 2022, pp. 103–112.

[34] 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.

[35] K. A. Kaufman and R. S. Michalski, "Learning from inconsistent and noisy data: the aq18 approach," in *International Symposium on Methodologies for Intelligent Systems*. Springer, 1999, pp. 411–419.

[36] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First international workshop on graph data management experiences and systems*, 2013, pp. 1–6.

[37] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 2, pp. 109–127, 2021.

[38] 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*, 2010, pp. 135–146.

[39] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "{PowerGraph}: Distributed {Graph-Parallel} computation on natural graphs," in *10th USENIX symposium on operating systems design and implementation (OSDI 12)*, 2012, pp. 17–30.

[40] 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.

[41] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," in *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No. 98CB36235)*. IEEE, 1998, pp. 357–368.

[42] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," *arXiv preprint arXiv:1803.02329*, 2018.

[43] A. Srivastava, T.-Y. Wang, P. Zhang, C. A. F. De Rose, R. Kannan, and V. K. Prasanna, “Memmap: Compact and generalizable meta-lstm models for memory access prediction,” in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2020, pp. 57–68.

[44] P. Zhang, A. Srivastava, R. Kannan, A. V. Nori, and V. K. Prasanna, “Transformap: Transformer for memory access prediction,” in *The International Symposium on Computer Architecture (ISCA), ML for Computer Architecture and Systems Workshop, 2021*, 2021.

[45] F. Wen, M. Qin, P. Gratz, and N. Reddy, “Software hint-driven data management for hybrid memory in mobile systems,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 1, pp. 1–18, 2022.

[46] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, “Local higher-order graph clustering,” in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 555–564.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[48] L. Dong, S. Xu, and B. Xu, “Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5884–5888.

[49] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.

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

[51] J. Leskovec, L. A. Adamic, and B. A. Huberman, “The dynamics of viral marketing,” *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, pp. 5–es, 2007.

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

[53] J. Yang and J. Leskovec, “Defining and evaluating network communities based on ground-truth,” *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[54] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, “The connected-component labeling problem: A review of state-of-the-art algorithms,” *Pattern Recognition*, vol. 70, pp. 25–43, 2017.

[55] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.” Stanford InfoLab, Tech. Rep., 1999.

[56] 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,” *SIGPLAN Not.*, vol. 40, no. 6, pp. 190–200, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1064978.1065034

[57] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[58] D. M. Powers, “Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation,” *arXiv preprint arXiv:2010.16061*, 2020.

[59] N. Ghamrawi and A. McCallum, “Collective multi-label classification,” in *Proceedings of the 14th ACM international conference on Information and knowledge management*, 2005, pp. 195–200.