A Methodology for Characterizing Sparse Datasets and Its Application to SIMD Performance Prediction

Gangyi Zhu Peng Jiang Gagan Agrawal
Department of Computer Science and Engineering
The Ohio State University
{zhu.926, jiang.952, agrawal.28}@osu.edu

***Abstract*—Irregular computations are commonly seen in many scientific and engineering domains that use unstructured meshes or sparse matrices. The performance of an irregular application is very dependent upon the dataset. This paper poses the following question: *"given an unstructured mesh or a graph,what method(s) can be used to sample it, such that the execution on the resulting sampled dataset can accurately reflect performance characteristics on the full dataset"*. Our first insight is that developing a universal sampling approach for all sparse matrices is unpractical. According to the non-zero distribution of the sparse matrix, we propose two novel sampling strategies: *Stride Average* sampling and *Random Tile* sampling, which are suitable for *uniform* and *skewed* sparse matrices respectively. To help categorize a sparse matrix as uniform or skewed, we introduce clustering coefficient as an important feature which can be propagated into the decision tree model. We also adapt Random Node Neighbor sampling approach for efficient estimation of clustering coefficient.**

We apply our unstructured dataset characterization approach to modeling the performance for SIMD irregular applications, where the sampled dataset obtained is used to predict cache miss rate and SIMD utilization ratio. We also build analytical models to estimate overheads incurred by load imbalance among threads. With knowledge of these factors, we adapt a code skeleton framework SKOPE to capture the workload behaviors and aggregate performance statistics for execution time prediction.

I. Introduction

Irregular computations are commonly seen in many scientific and engineering domains that involve unstructured meshes or sparse matrices. A key characteristic of these applications is *indirect* or *data-dependent* memory access pattern. These applications pose many challenges for the various code generation or optimization steps that are normally taken for scientific applications. For example, typically it cannot be determined at compile-time whether a reference will be a cache hit or not [1], or whether an element needs to obtained from another node's memory or not [2].

As a result, the performance and optimization strategy for an irregular application is highly dependent upon the dataset. For example, in the large body of research on sparse matrices, there is considerable work on choosing the right layout for the matrix, depending upon the sparsity pattern [3]. Similarly, research on graph algorithms has shown that properties of certain graphs, for example, the power-law degree distribution, impact the choice of the implementation approach [4].

The goal of this paper is to answer the question that given an unstructured mesh or a graph, what method is appropriate to obtain a sample, such that the performance behaviors on the sample match the characteristics of the execution with the original input. While there has been considerable work on sampling graphs [5]–[7], the goal has been finding representative vertices, or preserving properties like the diameter. We are not aware of any work that has sampled graph or unstructured datasets with the goal of capturing performance characteristics on a modern parallel architecture.

In this paper, we sample an irregular sparse dataset using the following approach. We find that clustering coefficient is an important measure to classify a sparse dataset as uniform or skewed. We also combine clustering coefficient obtained with certain other common features of a sparse matrix and use a decision tree for classification. Based on the classification, our framework can select Stride Average sampling for uniform datasets and Random Tile sampling for skewed datasets.

We propose a technique to accelerate estimating clustering coefficient based on Random Node Neighbor sampling. We are able to show that our approach is very efficient and effective in assessing clustering coefficient.

A. Specific Application: Performance Modeling in Presence of SIMD Features

Accurate performance prediction plays an important role in the high performance computing field [8]–[13]. For example, a supercomputing center planning to upgrade their machines will like to know what aspects are the most critical to the performance of the target workload and how does the new machine improve the performance. Similarly, an application developer can be interested in estimating the performance for different implementations and pick the optimal solution. In the case of irregular applications, where the performance is dataset-dependent, choosing the best version among a set of implementations or optimization options can also require performance prediction.

978-1-7281-3613-4/19/$31.00 ©2019 IEEE
DOI 10.1109/PACT.2019.00042

Performance prediction of irregular applications over sparse datasets is complicated by at least two factors – cache performance and SIMD utilization. Although a variety of approaches have been proposed to study and model cache behaviors, few of them pay attention to irregular memory accesses. Among the approaches that consider irregular computations [14]–[16], certain limitations exist, as we will elaborate later. A relatively new (but now important) consideration in performance modeling of irregular applications is the potential of use of SIMD parallelism for irregular applications. Although SIMD parallelism has been available in popular architectures for over a decade, it had traditionally been considered suitable only for regular applications. However, recent SIMD architecture developments including wider SIMD lanes (up to 512 bits), combination of SIMD parallelism and high MIMD parallelism, and more flexible programming APIs, make it possible to apply SIMD architectural advances to irregular applications. For example, Chen *et al.* [17] propose a methodology of *tiling and grouping* recently which can achieve high SIMD and MIMD parallelisms for irregular applications without contentions incurred.

In this paper, we apply our sparse dataset characterization approach to modeling and predicting the performance (execution time) for SIMD irregular applications over sparse datasets. We obtain a sampled dataset from input using the strategy automatically selected based on our characterization method. This sample can be used to predict cache performance and SIMD utilization. With knowledge of these factors, we adapt a code skeleton framework SKOPE [18] to capture the workload behaviors and aggregate performance statistics. Moreover, we improve the performance model inside SKOPE to accommodate SIMD irregular applications and incorporate analytical models to estimate overheads incurred by MIMD load imbalance.

Experiments are carried out on TACC stampede2 cluster [19] with two irregular reduction applications and one graph application executed over 100 sparse matrices with varying characteristics from 37 groups in SuiteSparse Matrix Collections [20]. The results show that our hybrid sampling approaches deliver accurate prediction for cache performance and SIMD utilization, with average error rates of 8.28% and 7.75%, respectively. Our prediction framework achieves low average error rates of 8.02% for one thread SIMD case and 11.95% for four MIMD configurations (2, 4, 16 and 68 threads). Our approach also achieves high efficiency with overheads of prediction only ranging from 8.24% to 18.14% of actual execution time.

II. Background

This section presents the background of the irregular applications targeted in our work, latest SIMD features from Intel, and Chen *et al.*'s work [17] to apply SIMD advances to irregular applications.

A. Irregular Applications

We now discuss irregular reductions and certain graph computations, which are two representative types of irregular applications.

Irregular reductions arise, for example, from unstructured grid computations, where the nodes are explicitly connected by edges. Examples of such reductions include `Euler`, which is a simulation of Computational Fluid Dynamic (CFD) [21], and `Moldyn`, which simulates the interaction and motion of molecules during the interaction period [22]. An example of irregular reduction loop is shown in Figure 1. During each iteration of the loop, two nodes values `X(IA(e, 1))` and `X(IA(e, 2))` are updated. The access to the node array `X` is indexed by the indirection array `IA`, and thus the access is irregular. When a node is connected by multiple edges, the updates to this node's value involve reduction operations.

```
Real X (numNodes), Y (numEdges); // node/edge arrays
Integer IA (numEdges, 2);        // indirection array
for (e = 0; e < numEdges; e++) {
     val = f(X(IA(e,1)), X(IA(e,2)), Y(e));
     X(IA(e,1)) = X(IA(e,1)) + val;
     X(IA(e,2)) = X(IA(e,2)) - val;
}
```

Fig. 1: An Irregular Reduction Loop

Since graph consists of nodes and edges, graph algorithms often have an irregular data access pattern. All nodes are connected by the indirection array (edges) such that the access to the node array is irregular. For example, `PageRank` is widely used in ranking search engine results [23]. It assigns a numerical weight to the elements of a hyperlinked set of documents, and measures the relative importance of each element within the set.

B. SIMD Instruction Set and Irregular Applications

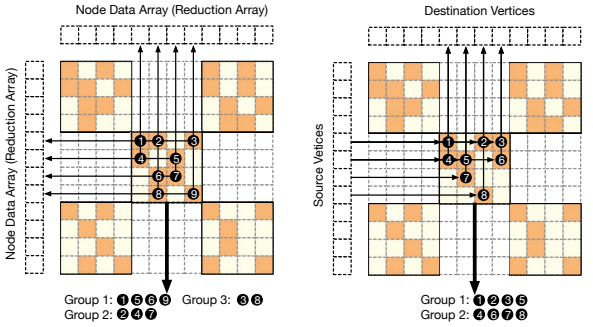
Consider the AVX-512 instruction set proposed by Intel in July 2013 [24]. The instruction set is currently supported in Intel Xeon Phi Knights Landing and Skylake-X Core i7 and i9 models.

AVX-512 has a family of *gather/scatter* primitives for loading/storing data at unaligned and non-continuous memory addresses. It also supports a *mask* data type with a set of *mask* operations, which allow computations on only a specified subset of lanes within a SIMD vector. Particularly, there is a class of *mask_gather/mask_scatter* instructions that allow reading/writing data on specified lanes of a SIMD vector. These features have enabled a broad class of irregular applications to benefit from SIMD [17], [25]–[27].

While gather and scatter instructions enable processing of irregular applications, it turns out achieving access locality is critical for performance. To improve locality and also to remove conflicts across SIMD lanes, Chen *et al.* [17] propose a methodology named *tiling and grouping*. We briefly summarize this approach here. Chen *et al.* observe that such indirect memory accesses in many irregular applications can

be modeled as operations on a sparse matrix as shown in Figure 2. Specifically, if a graph is represented by an adjacency matrix M with dimensions of $N \times N$ (N is the number of nodes), an interaction (`IA(e, 1)`, `IA(e, 2)`) in `IA` corresponds to a non-zero at the position (`IA(e, 1)`, `IA(e, 2)`) in M. To improve data locality and ensure sufficient number of non-zeros (i.e., high SIMD utilization) in each tile, Chen *et al.* tile the sparse matrix iteratively.

In the irregular applications targeted in this work, each iteration involves a pass over all non-zeros in the sparse matrix. For example, for irregular reduction application `Euler` and `Moldyn`, each iteration will pass all edges as shown in Figure 1. For graph algorithm `PageRank`, operation of Sparse Matrix-Vector Multiplication (SpMV) is performed in each iteration, i.e. all non-zeros are computed once in each iteration. The outer loop iterates over the inner loop without adding or removing non-zeros in the sparse matrix. The computation pattern and memory access pattern remain unchanged across iterations. These irregular applications we focus on can be considered as *static irregular*.



(a) Irregular Reductions (b) Graph Algorithms

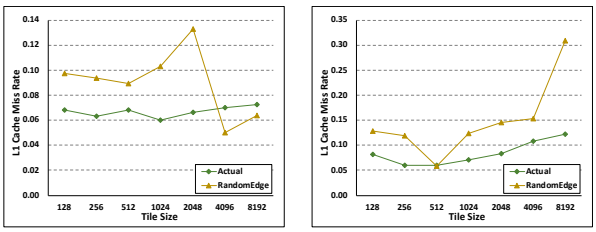
Fig. 2: Access Patterns of Irregular Reductions and Graph Algorithms

III. Sparse Dataset Characterization

As stated earlier, our goal is to obtain a sample of the original sparse dataset, such that the performance characteristics on the sampled dataset match those on the full dataset.

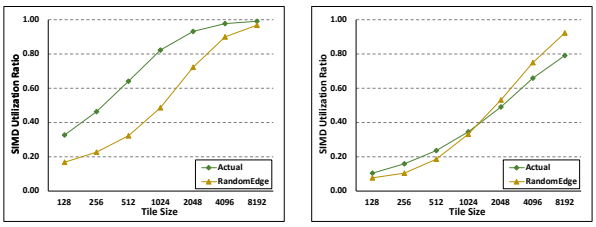
To motivate our work, we experimented with a classic sampling strategy – *Random Edge (RE)* [6] sampling, which uniformly selects an edge at random from the original grid/graph.

We compare the performance behaviors on sampled dataset with those on the original input. Figure 3 and Figure 4 show the results for predicting L1 cache miss rate and SIMD utilization ratio, respectively. The experiments are carried out on a KNL (Knights Landing) node (from TACC Stampede 2 cluster) with one irregular reduction application, `Euler`. We use two datasets from the SuiteSparse Matrix Collection [20] – *gsm_106857* and *kron_g500-logn19*. We can observe that the differences of cache performance and SIMD utilization between the original input and sample data are significant,



(a) gsm_106857 (b) kron_g500-logn19

Fig. 3: L1 Cache Miss Rate – Random Edge Sampling

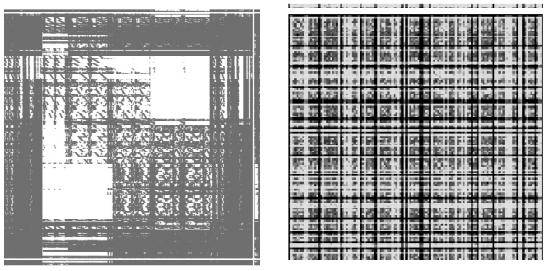


(a) gsm_106857 (b) kron_g500-logn19

Fig. 4: SIMD Utilization Ratio – Random Edge Sampling

which implies that simple random sampling is not capable of providing accurate performance insights. Different from the traditional sampling algorithms, the ideal sampling strategies for SIMD performance prediction should be able to generate a sample which shares similar SIMD performance behaviors rather than graph properties with the original input.

A. Approach Overview



(a) gsm_106857 (b) kron_g500-logn19

Fig. 5: Distribution of non-zeros in Matrices

In developing more nuanced sampling methods, our first observation is that sparse matrices vary very significantly with respect to distribution of non-zeros. A sparse matrix can have an *uniform distribution*, where the edges of the graph it represents are spread relatively evenly among all nodes (Figure 5(b)). On the other hand, nodes in a graph can also cluster together or be *skewed* (Figure 5(a)). As our goal is trying to obtain a sample that is similar to the original input in terms of execution performance, the distribution of sparse matrix is the key to designing our sampling strategies.

Clearly, it is unlikely that the same sampling method can be effective across different distributions of non-zeros. However, if we can have some basic knowledge of the non-zero distribution (either *uniform* or *skewed*), it would be easier

to design a dedicated sampling algorithm for sparse matrices with a specific kind of distribution. This, however, still leads to the following questions: 1) how can we effectively and efficiently characterize a matrix, and 2) what are the appropriate sampling schemes for each of the uniform and skewed type of matrices.

In the rest of this section, we first propose two novel performance-prediction-oriented sampling strategies: Stride Average sampling and Random Tile sampling, and then we elaborate how to categorize a sparse matrix and select the optimal sampling approach.

B. Stride Average Sampling

Given the processing granularity in this work is a tile, an intuitive idea to sample a uniform matrix is to pick a contiguous square region. Within this region, all the data access stride information and community information are preserved. However, an arbitrarily selected region doesn't necessarily deliver similar performance even though this region is similar to the original input in geometry.

In Section II-B, we discussed how an indirection array can be viewed as a sparse matrix. Based on this sparse matrix view, we can observe that the elements closer to the diagonal have shorter data access stride than the elements far away from the diagonal do. This is because that the node indices (`IA(e, 1)`, `IA(e, 2)`) defined by the diagonal elements are closer to each other. As cache performance is highly dependent on data access stride, the key of sampling is to preserve data access stride information. Based on this observation, we propose an approach named *Stride Average (SA)* sampling.

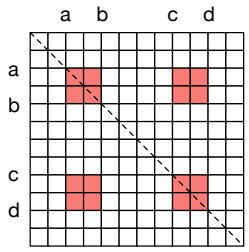


Fig. 6: Stride Average Sampling

The basic idea is to sample the dataset from four representative regions, of which two are on the diagonal while the other two are on the anti-diagonal. The idea originates from *stratified sampling* in statistics, where data is divided into subpopulations of similar elements, called strata, and representative samples from each strata are obtained. Compared to naive random sampling, this approach preserves both the data access stride characteristics and the community information while avoids the uncertainty of random sampling.

This means that the sample can be more likely to reflect actual cache performance and SIMD utilization when the non-zeros in the sparse matrix have relatively uniform distribution.

Figure 6 shows the Stride Average sampling approach. Given a 2D matrix as the input, we sample the data from four red regions in the figure to capture the data access stride and community information.

Algorithm 1: Stride Average Sampling

```
1: N: input adjacency list (neighbor list for each node)
2: n: number of nodes
3: p: percentage of sampled edges
4: V_e: output vector for sampled edges
5:
6: l ← √(p/4) // Percentage of the side length of sampled square
7: a ← (0.25 − l/2) · n, b ← (0.25 + l/2) · n,
   c ← (0.75 − l/2) · n, d ← (0.75 + l/2) · n
8: for u = a, u ≤ b, u++ do
9:    for each node v in N[u] do
10:      if a ≤ v ≤ b OR c ≤ v ≤ d then
11:         Push edge (u, v) into V_e
12:      end if
13:      if v > d then
14:         break
15:      end if
16:   end for
17: end for
18: for u = c, u ≤ d, u++ do
19:   Same as Line 12 to Line 19
20: end for
21: return V_e
```

The implementation of this method is shown as Algorithm 1. Given a sampling percentage p, we can determine the relative side length of four squares to be sampled. Then, a, b, c, and d are starting and ending points for two sides along two axes shown in Figure 6. When a non-zero falls into any squares, it will be added into the sample. This sampling strategy not only captures performance behavior information, but also delivers high efficiency. This method only scans the non-zeros sharing the same row within the sampled squares, and thus the method is very efficient.

C. Random Tile Sampling

For most sparse matrices with uniform distribution of non-zeros, Stride Average sampling is an efficient, accurate and stable approach to gain insights from the original input. On the contrary, for skewed sparse matrices, the sampled regions are likely unable to represent the entire matrix. Given the applications are processing the sparse matrices tile by tile, sampling at the granularity of tile is still helpful (the four sampled regions in Stride Average sampling can be viewed as four large tiles). But we can pick multiple randomly selected tiles rather than four large tiles with fixed positions to accommodate the skewed distribution. Note that "tile" here simply denotes a square region in the original matrix for sampling.

The choice of tile size involves a trade-off. In order to capture more information of non-zero distribution, a large tile is helpful. However, given a relatively small sampling ratio and skewed distribution, if the tile size is very large,

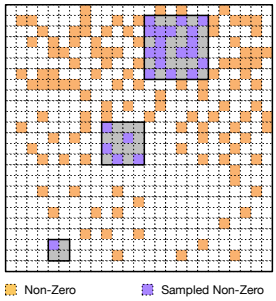


Fig. 7: Random Tile Sampling

then we can sample only a few tiles. In this case, the selection bias would arise. As we are targeting the sparse matrices with skewed distribution, the impact of the selection bias can be significant. On the other hand, if the tile size is small, although we may reduce selection bias, the non-zero distribution information inside each sampled tile is not representative. For example, in the extreme case, we set the tile size as 1, then this approach will be reduced to Random Edge sampling, which is unable to capture local non-zeros distribution, and data access stride information.

To resolve this issue, we propose an adaptive approach named as *Random Tile (RT)* sampling. We present our idea in Figure 7. In this example, the sparse matrix is skewed, and its upper half is denser than its lower half. We set the default tile size for sampling as 4. If the sampled region is "dense", the tile size will be incremented to 6 (1.5 times of the default size, i.e. the largest tile). If the sampled region is "sparse" , the tile size will be reduced to 2 (half of the default size, i.e. the smallest tile). Otherwise, the default tile size will be applied. With this mechanism, the tile size can be dynamically updated based on the local non-zero distribution.

The implementation of Random Tile sampling is presented in Algorithm 2. During each iteration, we first randomly generate the coordinates *(a, b)* as left top element of the tile. Then we compare the average density of the first and last rows with the overall sparse matrix density to determine the sampled region is dense or sparse, and then update the tile size accordingly. With consideration of sampling efficiency, here we use the density of the first and last rows to estimate the region density. All non-zeros within this tile will be sampled. And this process will be repeated until the sampling percentage is reached.

Algorithm 2: Random Tile Sampling

```
N: input adjacency list (neighbor list for each node)
m: number of edges
n: number of nodes
p: percentage of sampled edges
s: density of the sparse matrix
V_e: output vector for sampled edges

while V_e.size() < m · p do
    l ← t // Default tile size
    a, b ← Randomly selected two node from original graph
    if (N[a].size() + N[a + l].size())/(2 * n) < 0.5 · s then
        l ← 0.5 · l // Reduce tile size for a sparse region
    end if
    if (N[a].size() + N[a + l].size())/(2 * n) > 1.5 · s then
        l ← 1.5 · l // Increase tile size for a dense region
    end if
    for u = a, u ≤ a + l, u + + do
        for v in N[u] do
            if b ≤ v ≤ b + l then
                Push edge (u, v) into V_e
            else
                break
            end if
        end for
    end for
end while
return V_e
```

D. Categorizing A Sparse Matrix

Having introduced sampling methods suitable for dealing with uniform and skewed matrices, we now return to the problem of characterizing a given matrix as either uniform or skewed.

1) Introducing Clustering Coefficient for Characterizing Sparse Matrix: There are a variety of graph metrics to characterize a sparse matrix [5]–[7]. The common ones include *degree distribution*, *radius*, *diameter*, *density*, and others. Among all these metrics, *clustering coefficient* [28] is the best-fit for our goal of characterizing sparse matrix in terms of distribution of non-zeros.

Clustering coefficient measures the degree to which nodes in a graph tend to cluster together. The *local clustering coefficient* of a node in a graph quantifies how close its neighbors are to being a complete graph. Given a graph $G = (V, E)$ consisting of a set of vertices V and a set of edges E, and let e_{ij} denote the edge connecting the vertex i and the vertex j. Let N_i be the collection of immediately connected neighbors for vertex i and k_i be the number of its neighbors. The local clustering coefficient for a directed graph is defined as:

$$C_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)} \tag{1}$$

The local clustering coefficient for an undirected graph is defined as:

$$C_i = \frac{2|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)} \tag{2}$$

The overall level of clustering in a graph, i.e., the *global clustering coefficient* is measured as the average of the local clustering coefficients across all the nodes:

$$\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i \tag{3}$$

Based on Equations 1–3, calculating global clustering coefficient over the entire graph is extremely expensive. Given our goal of performance characterization, analyzing all nodes of the dataset is infeasible in practice.

To overcome this drawback, an intuitive way is to sample from the original graph and project the clustering coefficient based on the sample data. However, we can expect that random sampling (either Random Edge sampling ,or Random Row sampling, which randomly select an entire row) from original graph would lead to an underestimation of the clustering coefficient. The reason is that for each sampled node, its neighbors and edges among neighbors in the sampled dataset are fewer than those in the original dataset so that its clustering coefficient will be underevaluated.

To precisely estimate the clustering coefficient with high efficiency, we adapt the *Random Node Neighbor (RNN)* sampling method. Our idea to adapt RNN is that for each uniformly selected node, we calculate its local clustering coefficient with all the out-going edges and neighbors as well as the edges among the neighbors, and then obtain a global clustering coefficient by averaging all local clustering coefficients for the sampled nodes. The process is implemented as Algorithm 3. While RNN has been described in the literature (for example, Leskovec *et al.*'s work [6]), we are not aware of any work specifically viewing it or implementing it as an approach for estimating clustering coefficient.

We evaluated our adapted RNN sampling approach over 100 sparse matrices from SuiteSparse Matrix Collection [20]. The results along with the comparison with Random Row sampling and Random Edge sampling are shown in Figure 8. Here the reported error rate is defined as the absolute difference of clustering coefficient between the sampled dataset and the original dataset, divided by the clustering coefficient of the original dataset. The results verified the above analysis. First, our RNN approach significantly outperforms the other two sampling approaches in three different sampling percentages. It achieves low error rate (5.13%) even at the case of 1% sampling ratio. Second, both Random Row or Random Edge sampling approaches underrated the value of clustering coefficient in most cases (89% and 94%, respectively even at 5% sampling percentage).

Another important advantage of our approach over the other two is that the estimation is accumulative during the sampling process so that it is more efficient than the 2-step procedure of sampling and calculation. And our approach doesn't generate a real sample file during the process. Therefore, our approach is of high efficiency both in execution time and storage space.

2) Beyond Clustering Coefficient: Machine Learning Formulation: Though clustering coefficient can be quite effective in classifying a sparse matrix, we go a step further in adding other features and developing a machine learning (decision tree) formulation. Although machine learning techniques have been commonly seen in the problem of automatically selecting optimal storage format for sparse matrices [29]–[31], its application to characterizing sparse matrices for performance-prediction-oriented sampling is novel.

We use several features presented in Sedaghati *et al.*'s work [30] for our work, which are shown in Table I. These features are used in conjunction with clustering coefficient.

Algorithm 3: Clustering Coefficient Estimation by Random Node Neighbor Sampling

```
1: N: input adjacency list (neighbor list for each node)
2: m: number of edges
3: p: percentage of sampled edges
4:
5: total ← 0 // sum of local clustering coefficient
6: while True do
7:    u ← Randomly selected node from original graph
8:    k ← N[u].size() // number of neighbors
9:    node_ct ← node_ct + 1 // counter for sampled nodes
10:   edge_ct ← edge_ct + k // counter for sampled edges
11:   if k < 2 then
12:      continue
13:   end if
14:   link_ct ← 0 // counter for links among neighbors
15:   for each node v ∈ N[u] do
16:      for each node w ∈ N[u] do
17:         if w ∈ N[v] then
18:            link_ct ← link_ct + 1
19:         end if
20:      end for
21:   end for
22:   total ← total + link_ct/(k(k − 1))
23:   if edge_ct ≥ m · p then
24:      break
25:   end if
26: end while
27: C̄ ← total/node_ct // global clustering coefficient
28: return C̄
```

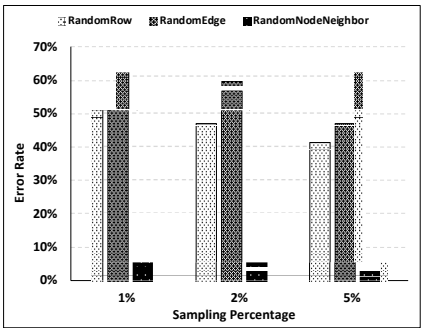


Fig. 8: Estimating Clustering Coefficient by Sampling

Also, it turns out that calculating these basic features is relatively easier than obtaining clustering coefficient. We can simply traverse the matrix once or just apply Random Row sampling to further accelerate the process.

IV. Execution Time Prediction Framework

We now apply the sparse dataset characterization technique from the previous section to developing an execution time prediction framework. The target architecture we use for our experiments is Intel Knights Landing (KNL), which is one of the available systems with advanced SIMD featres. AVX-512 instruction set is supported, which was described earlier. The combination of SIMD and MIMD parallelisms we characterize in the context of KNL is commonly seen in modern CPUs.

TABLE I: Base Features of Matrix

Feature	Description
nnz_tot	Number of non-zeros
nnz_frac	Percentage of non-zeros
nnz_{min, max, mu, sig}	Min, max, average and std. deviation of non-zeros per row
nnzb_tot	Number of non-zero blocks
nnzb_{min, max, mu, sig}	Min, max, average and std. deviation of non-zero blocks per row
snzb_{min, max, mu, sig}	Min, max, average and std. deviation of the size of non-zero blocks per row

A. Challenges

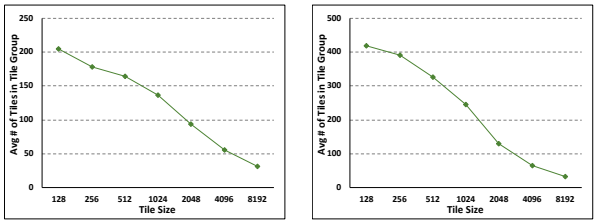
The major challenges for predicting the performance of irregular applications on a system like KNL come from three aspects: irregular data access patterns, the performance of *gather/scatter* operations, and uncertainty with both SIMD and MIMD parallelism achieved. First, due to the irregular data access pattern, it is hard to predict cache performance as the access to the data elements is commonly indexed by an indirection array. The indirection array itself represents the set of edges in unstructured grid or the graph, which is what we have tried characterizing efficiently through our techniques.

Second, since gather/scatter operations are recent developments for SIMD architecture, it is challenging to estimate the performance of these operations in irregular computations. Chen *et al.* [17] carried out an experiment of comparing the performance between the gather/scatter and load/store with varying the *access range size*. We can observe that as the access range increases, the performance gap between gather/scatter and load/store becomes wider. The reason is that as the 16 elements accessed by gather/scatter operation become further from each other, more cache lines are accessed, causing a drop in the performance. This result indicates the non-zero distribution is a dominant factor affecting the gather/scatter performance. Thus, this performance behavior of gather/scatter combined with the irregular access pattern leads to high uncertainty in cache performance.

Third, the level of SIMD (and MIMD) parallelism is also unknown before actual execution. As stated in Section II-B, to avoid write conflicts within SIMD lanes, the non-zeros in the sparse matrix are grouped into conflict-free groups. The number of non-zeros within each tile is between 1 to 16, and it indicates the *SIMD utilization ratio*. Moreover, the tiles are further partitioned into conflict-free tile groups. Figure 9 shows the average number of tiles in tile group for two datasets from the SuiteSparse Matrix Collection [20]. We can observe that as the tile size increases, the average number of tiles in a tile group significantly decreases. Moreover, certain tile groups only have a few tiles. The lack of tiles in these tile groups could magnify the load imbalance among the threads and impact the actual MIMD parallelism.

B. Overview

Having described the challenges, we now give a high-level view of our approach. Figure 10 shows an overview of our framework and the process to obtain projected performance for SIMD irregular applications. It consists of 5 modules



(a) gsm_106857 (b) kron_g500-logn19

Fig. 9: Average # of Tiles in Tile Group

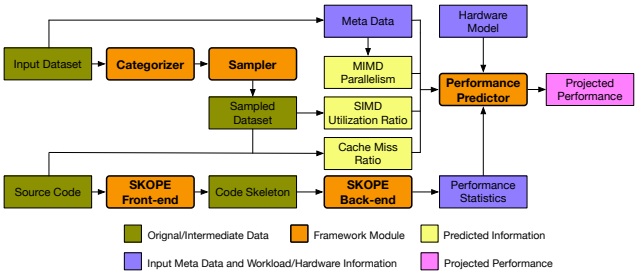


Fig. 10: Framework Overview

– categorizer, sampler, SKOPE front-end, SKOPE back-end, and performance predictor.

The input significantly impacts or determines the data access pattern (cache performance), SIMD utilization ratio, and MIMD parallelism. We use techniques from the last section to obtain a sample that can be executed with high efficiency. The categorizer analyzes the input and assigns an appropriate sampling method to it by our characterization technique. Then the sampler generates a sample with this sampling strategy. By either analyzing the sampled dataset or executing the application over the sample, we could obtain necessary information (i.e. cache performance and SIMD utilization) for performance prediction with a small fraction of the cost of executing the entire application.

In general, SKOPE [18] serves to gather performance statistics of the source code and predict performance based on these statistics and the performance metrics we have obtained. The SKOPE front-end takes source code as input, and converts it to code skeleton. Then SKOPE back-end collects and aggregates performance behavior statistics from the code skeleton. All input information and workload performance statistics as well as a hardware model are propagated into performance predictor to obtain projected performance. The hardware parameters in the hardware model include clock frequency, memory bandwidth, SIMD lane width, and others. The performance predictor contains performance models including our proposed models described in Section IV-C.

C. Modeling Overheads of Load Imbalance

Besides cache performance and SIMD utilization we have discussed above, another factor to consider is the load imbalance among threads. As stated in Section IV-A, tiles are partitioned into conflict-free tile groups, each of which is executed with all threads. If a tile group does not have

sufficient tiles or the tiles cannot be divided exactly by all threads, some threads may stay idle during execution. In general, if we have more tiles in each tile group and fewer threads, it is more likely that each thread keeps busy during execution. Otherwise, the overheads incurred by load imbalance are more significant. Based on this observation, we model this kind of overhead as

$$T_{MIMD} = a \times \frac{num\_threads}{avg\_MIMD} \quad (4)$$

where a is a factor obtained from training. Here we use average MIMD parallelism avg_MIMD to denote the average number of tiles in tile group. Based on the results shown in Figure 9, we can observe that the average number of tiles in tile group decreases as tile size increases. Besides, when the dataset has more non-zeros (edges), the average number of tiles in tile group is larger. Hence, we model the average number of tiles as:

$$avg\_MIMD = b \times \frac{num\_edges}{tile\_size} \quad (5)$$

Combining Equation 5 to Equation 4 (b and a can be combined to one factor), we can estimate the overheads incurred by load imbalance.

D. Predicting Execution Time with SKOPE

Although SKOPE supports SIMD application to some extent, it is not fully applicable in predicting SIMD irregular applications involving tiling and grouping. To take advantage of SKOPE, we customize the performance model in SKOPE to accommodate the properties of irregular SIMD applications. First, we introduce SIMD utilization ratio to performance model. In the original SKOPE performance projection module, SIMD application is assumed to exploit all SIMD parallelism. In other words, it assumes the SIMD lanes are always full during execution. However, in practice, especially for irregular applications, it is common that the SIMD lanes are not full due to write conflicts. Hence, we use actual SIMD lane width instead of theoretical SIMD lane width and derive it as:

$$SIMD\_Width_{actual} = SIMD\_Width_{orig} \times SIMD\_Ratio \quad (6)$$

where $SIMD\_Ratio$ is the SIMD utilization ratio captured over sampled dataset.

Overall, the process of predicting execution time for irregular SIMD applications involves three major steps. The first step is to obtain input information with categorizer and sampler, where the cache miss ratio and SIMD utilization ratio are estimated. Second, we need to obtain the code skeleton for the irregular application by using the translator in SKOPE front-end. This code skeleton is the foundation for the prediction since it captures the workload behaviors. The last step is to predict performance. Besides the insights gained in the first step and performance statistics obtained in the second step, a hardware model for the target architecture should be built. The parameters of a hardware model can be obtained from documentations or benchmark results. By propagating all these information into the customized performance model, our framework is able to predict execution time for SIMD processing of irregular applications.

V. Experimental Results

In this section, we extensively evaluate our approach for characterization and performance prediction. The first set of experiments evaluate the effectiveness of our characterization approach in categorizing sparse matrix inputs. The second set focuses on evaluating accuracy in predicting three metrics: the cache miss rate, SIMD utilization ratio, and the overall execution time on a KNL node. The last set evaluates the efficiency of our approach by comparing the prediction overheads with the actual execution time. We experiment with two irregular reduction applications – `Euler` and `Moldyn`, and one graph algorithm – `PageRank`. The inputs are 100 randomly selected sparse matrices from 37 groups in SuiteSparse Matrix Collections [20].

All experiments are carried out on Intel Xeon Phi 7250 (Knights Landing) available from the TACC Stampede2 cluster [19]. Each KNL node has 68 cores with 1.4 GHz clock frequency and 96 GB DDR4 RAM plus 16 GB high-speed MCDRAM. As stated before, each node also has 32 KB L1 data cache per core, and 1 MB L2 cache per two-core tile.

The sampling rates used across all experiments are set as 5% to achieve a good balance between accuracy and efficiency. The results are measured with absolute *error rate* which is defined as:

$$error\_rate = \frac{|\ Value_{actual} - Value_{predicted}\ |}{Value_{actual}} \quad (7)$$

where $Value_{actual}$ and $Value_{predicted}$ are actual and predicted results, respectively. And any random sampling approaches will be tested 5 times and results are averaged to reduce random error.

The entire set of 100 matrices is split into two groups based on the 80%-20% rule. The training of the decision tree model is conducted over the 80% of the inputs, and the rest 20% are used for testing. This process will be repeated 5 times to conduct 5-fold cross validation.

A. Categorizing Sparse Matrices

For this set of experiments, we first apply the Random Row sampling to generate the main features as shown in Table I, and we also utilize our adapted Random Node Neighbor sampling approach to calculate the clustering coefficient as an extra feature for all matrices. Second, we tag each matrix with the label of the "better sampling" strategy (either Stride Average sampling or Random Tile sampling). The label is selected based on which method better predicts SIMD utilization ratio, calculated over the samples generated by these two approaches. The reason to choose SIMD utilization ratio

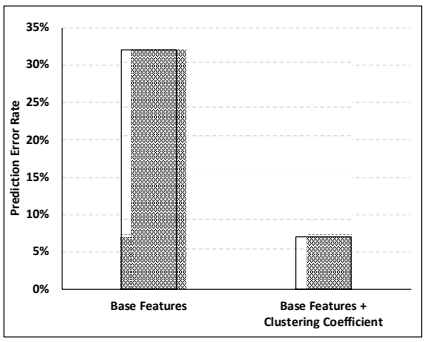


Fig. 11: Categorizing Sparse Matrices

as the criterion is that it has the greatest impact on execution time.

The prediction results by the decision tree model are presented in Figure 11. We can compare the prediction error rates of using two different feature vectors: one only include the base features, and the other has an extra feature – the clustering coefficient. We can observe that the error rate of using only the main features is over 30% while adding clustering coefficient to the feature vector helps to keep the error rate below 7%.

B. Predicting SIMD Utilization Ratio

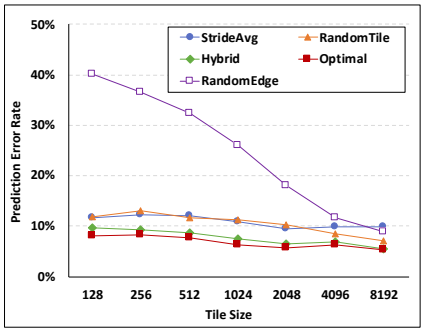


Fig. 12: SIMD Utilization Ratio Prediction

In this section, we evaluate the SIMD utilization ratio prediction results with tile size varied from 128 to 8192. We compare the results in three cases: using Stride Average sampling only (denoted as "`StrideAvg`"), using Random Tile sampling only (denoted as "`RandomTile`"), and the hybrid approach by selecting one of two sampling strategies based on input's predicted category (denoted as "`Hybrid`"). Besides, a theoretical approach that selects better of these two (denoted as "`Better`") and the result of Random Edge sampling (denoted as "`RandomEdge`") are also presented for comparison.

The results are presented in Figure 12. Each reported data point is the average error rate of the 100 sparse matrices. The key observations are as follows. First, the prediction error rates decrease as tile size increases. As tile size increases, each tile would have more non-zeros and the overall SIMD utilization is approaching 100%, and this trend is also visible in Figure 4. In this case, when the tile size is large enough and the SIMD utilization almost reaches the ceiling, the prediction error rate could be reduced.

Second, both Stride Average sampling and Random Tile sampling outperform the Random Edge sampling in almost all cases. Compared with Random Edge sampling, which omits the structure and community information, the two proposed approaches keep low error rates (¡ 15%) in all cases. The average error rates for Stride Average sampling and Random Tile sampling are 10.92% and 10.56% respectively, while Random Edge sampling has a high average error rate of 24.86%. Note that although the difference between Random Edge sampling and our approaches becomes negligible when tile size goes sufficiently large, Random Edge sampling is still not applicable in practice. Increasing tile size could boost the SIMD utilization, but large tile size would also involve larger data access strides, which can negatively impact the overall performance. The optimal tile size is the one that can balance SIMD utilization and cache performance. In other words, the tile size selected in practice would not be very large so that Random Edge sampling fails to deliver accurate predictions.

Third, our hybrid approach improves the prediction accuracy and reduces the average error rate to 7.75%. which is very close to the "better" result (6.83%). By selecting a sampling approach based on the input's predicted category, our hybrid approach is able to avoid the weakness of each sampling approach.

C. Predicting Cache Performance

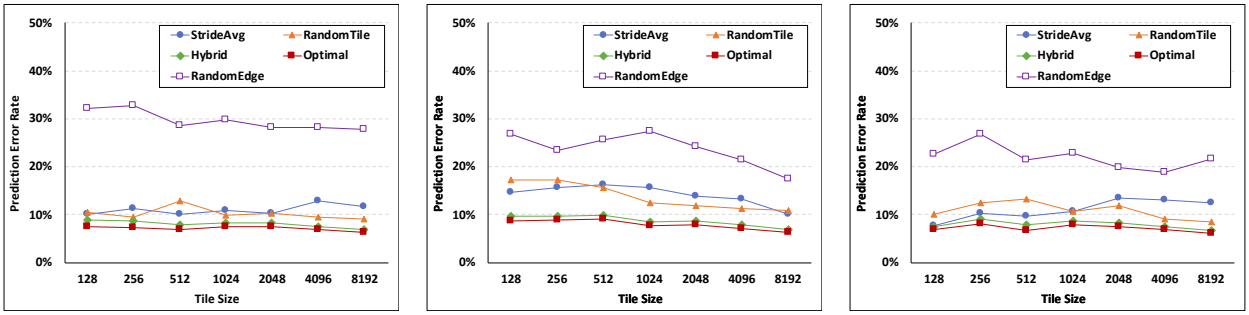
Besides the SIMD utilization ratio, cache performance is another key factor for performance prediction. Compared with predicting SIMD utilization ratio, predicting cache performance is even more challenging. Cache performance is dependent on not only the number of non-zeros in each tile, but also the relative distance (access stride) between two non-zeros.

Figure 13 is the results of predicting L1 cache miss rate for three applications. For Stride Average sampling, the average error rates for three applications across different tile sizes are 11.02%, 14.23% and 11.05% respectively. For Random Tile sampling, the average error rates for three applications are 10.22%, 13.80% and 10.81%. As a comparison, Random Edge sampling has average error rates of 29.62%, 23.78% and 21.99%, which are much higher than the error rates of other two approaches.

Our characterization-based hybrid approach further reduces the error rates to 8.09%, 8.78% and 7.98% in three cases, which are comparable to the "better" results (7.12%, 7.96% and 7.17%). The results prove that our characterization approach is effective in selecting the right sampling approach for a given input.

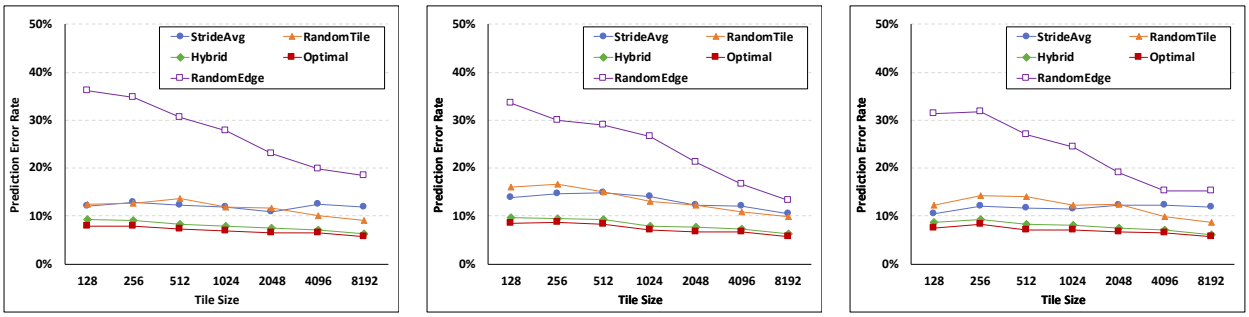
D. Predicting Execution Time

This set of experiments evaluates our framework for predicting execution time for irregular applications under dif-



(a) Euler (b) Moldyn (c) PageRank

Fig. 13: L1 Cache Miss Rate Prediction



(a) Euler (b) Moldyn (c) PageRank

Fig. 14: Execution Time Prediction

TABLE II: Error Rates of MIMD Execution Time Prediction

Application	2	4	16	68	Avg
Euler	9.32%	8.39%	11.87%	13.21%	10.70%
Moldyn	10.41%	11.25%	13.23%	15.21%	12.53%
PageRank	9.14%	10.78%	14.32%	16.28%	12.63%

ferent tile sizes and different number of threads. First, we evaluate the prediction of execution time in the sequential scenario (1 thread) with tile size varied from 128 to 8192. And then we extend the experiments to MIMD cases with MIMD parallelism varied from 2 to 68.

The results of the sequential experiments are shown in Figure 14. The average prediction error rates of execution time by Stride Average sampling are 12.07%, 13.20% and 11.75% for three applications respectively. And average error rates for Random Tile sampling are 11.64%, 13.40% and 11.97%. Random Edge sampling approach delivers average error rates of 27.24%, 24.32% and 23.43%. We can observe that the inaccurate prediction of SIMD utilization and cache performance leads to inaccurate execution time prediction. The average error rates of our hybrid approach are 7.92%, 8.27% and 7.87%, while the "better" results are 6.97%, 7.39% and 7.00%.

The prediction error rates of MIMD cases are summarized in Table II. We can observe that even in the MIMD scenarios, our framework still predicts the overall performance accurately with average error rates of 10.70%, 12.53% and 12.63%. Another observation is that the prediction error rate increases as the MIMD parallelism increases. This is because when the number of threads is large, the load imbalance among threads is more significant and some of the threads cannot be fully utilized. Although our approach models this kind of overheads to a certain extent and achieves relatively high accuracy, it is challenging to model it very precisely due to inevitable uncertainty.

E. Efficiency Analysis

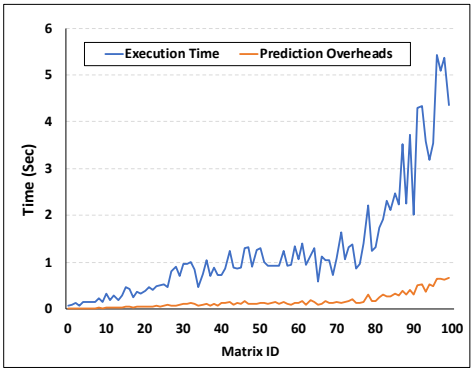


Fig. 15: Efficiency Analysis

In this section, we evaluate the efficiency of our approach by comparing the prediction overheads with the application execution time. The overheads incurred during the prediction process involve four components: sampling, categorization, execution over sample, and final performance prediction. Results are presented in Figure 15. For this experiment, target application `Euler` is executed with tile size of 512 in single-thread SIMD fashion. The reported value is the average time for a single iteration. The 100 datasets are sorted by number of non-zeros and each dataset is assigned a matrix id based on its ranking.

We can observe that the proportion of prediction overheads to execution time ranges from 8.24% to 18.14%. Typically, this ratio is lower for a larger dataset. This is because the overheads of sampling (including calculating clustering coefficient and features) and execution are dependent on the matrix size while the overheads of categorization and final performance prediction are more stable (which indicates a smaller share in the overall overheads for a larger dataset). These results demonstrates that our approach is much more efficient as compared to actual execution of the application. Moreover, since the sampling and categorization are one-time operations for each matrix, our approach would be more efficient in predicting performance of multiple execution setups (different applications using the same dataset, or different tile sizes or number of threads).

VI. Related Work

Although irregular computation patterns are commonly seen in many scientific domains, modeling their performance on modern processors has not drawn too much attention. Song *et al.* [32] develop a profiling methodology to estimate a computing node's computational capability for graph processing and guide graph partitioning in heterogeneous environment. Friese *et al.* [14] proposed an approach for generating performance models for irregular applications based on *hierarchical critical path analysis*. It uses Intel's load latency monitoring technique to estimate the data access costs, but the problem of quantifying the impact of changes in data access patterns is not considered. Andrade *et al.* [15] extended the PME (Probabilistic Miss Equations, originally proposed by Fraguela *et al.* [33]) model for irregular codes. Though this extended approach can model cache behavior for codes with irregular accesses, the drawback of the original PME model still exists – most critically, it assumes that all elements in the array have the same probability of being accessed by the means of indirection. This assumption does not hold in practice, especially for graph applications. Langguth *et al.* [16] present a quantitative understanding of the achievable performance of cell-centered finite volume method on 3D unstructured meshes on GPUs and CPUs. Their focus is on predicting the performance upper bound instead of the exact running time. Hence, memory and cache bandwidth are the key factors they consider in their model. Other factors including data access pattern and cache behavior are not modeled. None of the work above considers SIMD scenario. Zhu *et al.* [34] proposed an Adaptive Stratified Row sampling approach to capture irregular cache behaviors, but it does not consider the impact of tiling and grouping as well as the SIMD scenario. SIMD performance modeling is discussed in *Roofline* model that is designed for floating-point programs and multicore architectures [35]. Although it considers SIMD throughput in the model, the Roofline model is not capable of estimating the impact of irregular access patterns to SIMD performance.

VII. Conclusions

This paper has three major contributions. First, we have proposed two novel performance-prediction-oriented sampling approaches that are able to generate a sample with similar SIMD performance behavior as the original input. Second, we have developed a methodology for characterizing a sparse dataset, from the view-point of performance of an irregular application using this dataset. By categorizing the input into different types based on an estimation of clustering coefficient, our framework chooses the appropriate sampling solution from our proposed Stride Average sampling and Random Tile sampling. Third, we have built on this approach to develop an execution time prediction tool for SIMD (and MIMD) irregular applications. With the predicted cache performance and SIMD utilization ratio based on the sample, and performance behavior statistics collected by SKOPE, as well as customized performance models, our framework is able to precisely predict execution time for SIMD irregular applications. Our experimental evaluation has shown that our approach achieves high accuracy, high efficiency and broad applicability.

Acknowledgement: This work was funded by NSF grants CCF-1526386 and CCF-1629392.

References

[1] C. Ding and K. Kennedy, "Improving cache performance in dynamic applications through data and computation reorganization at run time," in *ACM SIGPLAN Notices*, vol. 34, no. 5. ACM, 1999, pp. 229–241.

[2] R. Das, M. Uysal, J. Saltz, and Y.-S. Hwang, "Communication optimizations for irregular scientific computations on distributed memory architectures," *Journal of Parallel and Distributed Computing*, vol. 22, no. 3, pp. 462–479, Sep. 1994.

[3] R. Vuduc, J. W. Demmel, and K. A. Yelick, "Oski: A library of automatically tuned sparse matrix kernels," *Journal of Physics: Conference Series*, vol. 16, no. 1, p. 521, 2005.

[4] X. Yang, S. Parthasarathy, and P. Sadayappan, "Fast sparse matrix-vector multiplication on gpus: implications for graph mining," *Proceedings of the VLDB Endowment*, vol. 4, no. 4, pp. 231–242, 2011.

[5] E. M. Airoldi and K. M. Carley, "Sampling algorithms for pure network topologies: a study on the stability and the separability of metric embeddings," *ACM SIGKDD Explorations Newsletter*, vol. 7, no. 2, pp. 13–22, 2005.

[6] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 631–636.

[7] P. Hu and W. C. Lau, "A survey and taxonomy of graph sampling," *arXiv preprint arXiv:1308.5865*, 2013.

[8] J. Bhimani, N. Mi, M. Leeser, and Z. Yang, "Fim: performance prediction for parallel computation in iterative data processing applications," in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*. IEEE, 2017, pp. 359–366.

[9] R. Mitra, B. S. Joshi, A. Ravindran, A. Mukherjee, and R. Adams, "Performance modeling of shared memory multiple issue multicore machines," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*. IEEE, 2012, pp. 464–473.

[10] B. Putigny, B. Goglin, and D. Barthou, "A benchmark-based performance model for memory-bound hpc applications," in *High Performance Computing & Simulation (HPCS), 2014 International Conference on*. IEEE, 2014, pp. 943–950.

[11] P. A. Dinda, "Online Prediction of the Running Time of Tasks: Summary," in *Proceedings of ACM SIGMETRICS*, 2001, pp. 336–337.

[12] D. Ofelt and J. L. Hennessy, "Efficient Performance Prediction for Modern Microprocessors," in *Proceedings of ACM SIGMETRICS 2000*, Jun. 2000, pp. 229–239.

[13] G. Zhu, Y. Wang, and G. Agrawal, "Scicsm: novel contrast set mining over scientific datasets using bitmap indices," in *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*. ACM, 2015, p. 38.

[14] R. D. Friese, N. R. Tallent, A. Vishnu, D. J. Kerbyson, and A. Hoisie, "Generating performance models for irregular applications," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 317–326.

[15] D. Andrade, B. B. Fraguela, and R. Doallo, "Precise automatable analytical modeling of the cache behavior of codes with indirections," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 4, no. 3, p. 16, 2007.

[16] J. Langguth, N. Wu, J. Chai, and X. Cai, "Parallel performance modeling of irregular applications in cell-centered finite volume methods over unstructured tetrahedral meshes," *Journal of Parallel and Distributed Computing*, vol. 76, pp. 120–131, 2015.

[17] L. Chen, P. Jiang, and G. Agrawal, "Exploiting recent simd architectural advances for irregular applications," in *Proceedings of the 2016 International Symposium on Code Generation and Optimization*. ACM, 2016, pp. 47–58.

[18] J. Meng, X. Wu, V. Morozov, V. Vishwanath, K. Kumaran, and V. Taylor, "Skope: A framework for modeling and exploring workload behavior," in *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM, 2014, p. 6.

[19] "Stampede2 User Guide," https://portal.tacc.utexas.edu/user-guides/stampede2, Online.

[20] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[21] R. Das, D. Mavriplis, J. Saltz, S. Gupta, and R. Ponnysamy, *The design and implementation of a parallel unstructured Euler solver using software primitives*. Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, 1992, vol. 189625.

[22] M. P. Allen *et al.*, "Introduction to molecular dynamics simulation," *Computational soft matter: from synthetic polymers to proteins*, vol. 23, pp. 1–28, 2004.

[23] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine," *Computer networks*, vol. 56, no. 18, pp. 3825–3833, 2012.

[24] J. Reinders, "Intel avx-512 instructions," https://software.intel.com/en-us/blogs/2013/avx-512-instructions, Jun. 2017.

[25] B. Ren, T. Poutanen, T. Mytkowicz, W. Schulte, G. Agrawal, and J. R. Larus, "Simd parallelization of applications that traverse irregular data structures," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, ser. CGO '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/CGO.2013.6494989

[26] B. Ren, Y. Jo, S. Krishnamoorthy, K. Agrawal, and M. Kulkarni, "Efficient execution of recursive programs on commodity vector hardware," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '15. New York, NY, USA: ACM, 2015, pp. 509–520. [Online]. Available: http://doi.acm.org/10.1145/2737924.2738004

[27] P. Jiang and G. Agrawal, "Efficient simd and mimd parallelization of hash-based aggregation by conflict mitigation," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '17. New York, NY, USA: ACM, 2017, pp. 24:1–24:11. [Online]. Available: http://doi.acm.org/10.1145/3079079.3079080

[28] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[29] J. Li, G. Tan, M. Chen, and N. Sun, "Smat: an input adaptive auto-tuner for sparse matrix-vector multiplication," in *ACM SIGPLAN Notices*, vol. 48, no. 6. ACM, 2013, pp. 117–126.

[30] N. Sedaghati, T. Mu, L.-N. Pouchet, S. Parthasarathy, and P. Sadayappan, "Automatic selection of sparse matrix representation on gpus," in *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015, pp. 99–108.

[31] Y. Zhao, C. Liao, J. Li, and X. Shen, "Bridging the gap between deep learning and sparse matrix format selection," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2018, pp. 94–108.

[32] S. Song, M. Li, X. Zheng, M. LeBeane, J. H. Ryoo, R. Panda, A. Gerstlauer, and L. K. John, "Proxy-guided load balancing of graph processing workloads on heterogeneous clusters," in *2016 45th International Conference on Parallel Processing (ICPP)*. IEEE, 2016, pp. 77–86.

[33] B. B. Fraguela, R. Doallo, and E. L. Zapata, "Probabilistic miss equations: Evaluating memory hierarchy performance," *IEEE Transactions on Computers*, vol. 52, no. 3, pp. 321–336, 2003.

[34] G. Zhu and G. Agrawal, "A performance prediction framework for irregular applications," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*. IEEE, 2018, pp. 304–313.

[35] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.