MemFriend: Understanding Memory Performance with Spatial-Temporal Affinity

Yasodha Suriyakumar*
Portland State University
USA
yasodhan@pdx.edu

Nathan R. Tallent
Pacific Northwest National Laboratory
USA
tallent@pnnl.gov

Andrés Marquez
Pacific Northwest National Laboratory
USA
andres.marquez@pnnl.gov

Karen L. Karavanic†
Portland State University
USA
karavanic@pdx.edu

Ozgur O. Kilic
Brookhaven National Laboratory
USA
okilic@bnl.gov

Abstract

In HPC applications, memory access behavior is one of the main factors affecting performance. Improving an application's memory access behavior requires studying spatial-temporal data locality. Existing data locality analyses focus on *single* locations. We introduce locality metrics between *pairs* of memory locations that quantify three dimensions of *spatial-temporal affinity*: temporal access proximity, forward access correlation, and nearby access correlation. We describe methods for distinguishing between potential vs. realized affinity and for reasoning about affinity (or friendship) at multiple resolutions (4D, 3D, 2D, 1D). Finally, we construct spatial-temporal affinity signatures that classify memory behavior and are used to reason about changes in software (data relayout, code refactoring) or hardware (caching, prefetching). We describe methods for signature visualization, interpretation, and quantitative comparison of signatures. We evaluate our methodology using applications with variants that contrast data structures, data layouts and algorithms. We show that spatial-temporal affinity analysis provides novel insights and enables predictive reasoning about application performance.

Keywords

Data locality, spatial-temporal affinity, memory access patterns, affinity signature, application performance

1 Introduction

In current HPC applications, the memory system is frequently a significant source of performance bottlenecks [29, 44] that affects nearly all machine platforms such as CPUs, GPUs or a heterogeneous combination [8]. Addressing memory system bottlenecks requires tools for diagnosing memory bottlenecks, characterizing application-platform suitability and evaluating memory systems and designs. An open problem in data locality analysis is how to concisely represent locality in a way that both predicts performance and distinguishes between classes of behavior such as access patterns, cache utilization and hardware prefetching.

Prior work on data locality includes metrics such as access frequency, reuse distance and footprint [28, 45, 46]. These metrics are defined with respect to a *single memory location*. Multiple efforts [4, 11, 12, 19, 27] have reported on the limitations of these metrics, such as their focus on temporal locality and their inability to capture access patterns and guide layout optimizations. Others [4, 11, 12, 27] explore capturing spatial-temporal locality to address the limitations of reuse distance by adding analyses at multiple granularities.

We argue that answering the more general question "What location j is likely to be accessed within the vicinity of an access to location i?" provides a precise and comprehensive view of the application's spatial-temporal locality. Though previous efforts attempt to answer this question, they take different approaches by limiting the problem to specific temporal windows to pack correlated objects into a cache line. *Reference affinity* [26, 47–49] pairs/splits arrays and structures using similarities in their reuse distance values. Miucin and Fedorova [30] classify objects and object fields into communities based on correlation in access patterns. A limitation in these methods is that by using a single metric, it may be difficult to distinguish when it is more important to prioritize optimizations that improve caching vs. those that improve prefetching.

We introduce *spatial-temporal affinity metrics* that quantify spatial-temporal locality between *pairs* of memory locations. The three metrics — *interval*, *anticipation*, and *density* — highlight different dimensions of pair-wise locality, respectively: temporal access proximity, *forward* reference correlation (cf. prefetching), and *nearby* reference correlation (cf. caching). We describe intuitive rules that guide optimizations based on these spatial-temporal affinity metrics. Affinity metrics enable the identification of *friendship* or *affinity clusters* of related memory locations or data objects that guide decisions on object allocation, data layouts, code refactoring, caching, or prefetching.

Affinity analysis has similarities to both statistical correlation and *market basket* analysis used in recommender systems. As a kind of correlation, naive affinity analysis based on pair-wise comparisons requires quadratic space and time. Therefore, narrowing the space of interesting possibilities is important. We develop efficient *location*-based, multi-resolution zooming to find hot memory regions by access *density* and access *frequency*. As a result, our

*Work performed while at Pacific Northwest National Laboratory.

†This material is based in part upon work supported by (while serving at) the National Science Foundation.

affinity analysis considers a small fraction of the possible combinations. Other forms of pattern analysis [2, 5, 7, 25] can be used to find correlated memory locations, but they use fundamentally different techniques with large time and space requirements.

We evaluate spatial-temporal affinity analysis using several representative applications. We show that affinity metrics are more informative and predictive of memory behavior when contrasted with several existing locality metrics; and demonstrate that the signatures are predictive of memory performance. Our contributions are as follows:

- We introduce novel measures (§3) of *spatial-temporal affinity* between pairs of memory locations that characterize the degree of temporal access proximity, *forward* reference correlation, and *nearby* reference correlation.
- We describe methods (§4) for distinguishing between potential vs. realized affinity and for reasoning about affinity at multiple resolutions (4D, 3D, 2D, 1D). We also present interpretive guidelines and comparison methods.
- We develop the MemFriend tool (memory *friendship*; § 5 and 6), a new module for MemGaze [18, 20]. MemFriend constructs spatial-temporal affinity signatures that classify memory behavior and are more predictive of expected performance than single-location reuse analysis.
- We evaluate spatial-temporal affinity (§7) by comparing against state-of-the-art metrics for several benchmarks with varying (only) 1) data structures, 2) data layouts, or 3) algorithms.

2 Motivating Multiple Affinity Relations

In this paper, we develop multiple metrics to answer the question "What location j is likely to be accessed within the vicinity of an access to location i?" As shown in Figure 1, each of the three metrics highlights a different dimension of pair-wise locality: *interval* or temporal reference (access) proximity; *anticipation* or *forward* reference correlation (cf. prefetching); and *density* or *nearby* reference correlation (cf. caching). Spatial-temporal affinity can therefore be viewed as a 3-tuple on 3 axes. Spatial *interval* is the distance of a reference interval. Spatial *anticipation* and *density* can be thought of as two different measures of the conditional probability of block j occurring given block i, computed as a ratio of j to i. Thus, we usually adopt the syntax $(j|i)$. We call i the *reference* location and j the *affinity* location.

We provide the following guidelines for using spatial-temporal affinity to reason about different situations and performance opportunities.

1. We expect high performance when a block and its *spatially contiguous* blocks have (a) good temporal proximity, i.e., low values of $interval(i, j)$ and (b) good correlation, i.e., high $anticipation(j|i)$ and/or $density(j|i)$.

2. There are opportunities for data relayout when a block i and its correlated j have unfavorable spatial separation with good temporal proximity, i.e., (a) low $interval(i, j)$ for large positive j or small/large negative j and (b) good reference correlation, i.e., high values of $anticipation(j|i)$ and/or $density(j|i)$. Specifically, these conditions identify when a block pair's logical spatial locality is not aligned with the actual spatial locality.

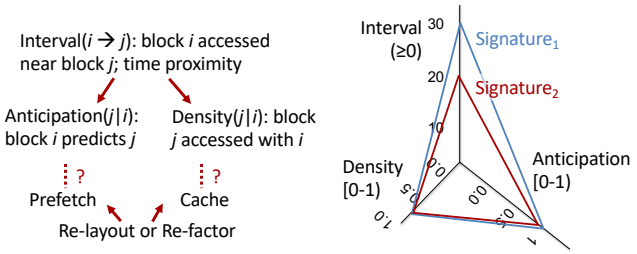


Figure 1: (Left) Affinity metrics and their interpretation. Both *anticipation* and *density* adopt the syntax of conditional probability. (Right) Two examples of 1D affinity signatures, specifically *affinity vectors* are shown as radar plots.

3. There is potential for code refactoring, such as bringing two time-separated statements together, to exploit latent locality (long-distance reuse) when there are (a) *hot* blocks with (b) poor temporal proximity, i.e., high $interval(i, j)$ and (c) poor reference correlation, i.e., low $anticipation(j|i)$ and $density(j|i)$.

4. There is potential to insert impactful software prefetching (to preload j, after access to i) when there is (a) poor temporal proximity, i.e., high $interval(i, j)$ to permit a sufficient interval for the prefetching distance window and (b) high correlation between accesses to i and j, i.e., high $anticipation(j|i)$.

5. There is potential for increasing reuse when a block i has (a) poor temporal proximity to itself, i.e., high $interval(i, i)$ and (b) poor nearby reference correlation, i.e., low value of $density(i|i)$.

3 Spatial-Temporal Affinity

This section defines three location-centric *spatial affinity* metrics—*interval*, *anticipation*, and *density*—that can be applied at different resolutions.

3.1 Preliminaries

We use the following well known memory access metrics.

Accesses, A(b). Access total for block b.

Access intensity, AI(b). Normalized access fraction (in $[0, 1]$) of block b. Given accesses to blocks in a region r, let $\mathrm{A}_{\max}(b)$ be the maximum. Then, $\mathrm{AI}(b) = \frac{\mathrm{A}(b)}{\mathrm{A}_{\max}(b)}$.

Reuse distance, RD(b). Number of unique blocks accessed between consecutive accesses to block b, is equal to the average RD for block b.

Reuse interval, RI(b). Number of expected accesses encountered before the reuse of a block b, is equal to the average RI (in non-unique accesses) of block b.

We introduce and use the following two definitions:

Affinity block pair, (i, j). Affinity is between *reference block* i, and *affinity blocks* j.

Lifetime, L(b). Number of (non-unique) accesses between the first and last access to block b.

3.2 Spatial interval

Spatial interval is a generalization of block reuse interval that measures *temporal access proximity*, where a memory access sequence is an abstract form of time. Spatial interval is important because

spatial locality is most beneficial when it occurs within small time windows. Whether it is accessing all elements of a cache line, two distinct cache blocks, or locations with a DRAM module that can be indexed together, the time window must be short enough to be beneficial.

Spatial interval is an integer. Smaller is generally better as it means that block j is accessed more quickly after i.

Definition. For memory block pair (i, j), the *spatial interval* $\mathsf{SI}_1(i, j)$ is number of accesses (≥ 0) between an access to block j and the *first previous* access to block i. The interval is *directional* in that accesses progress forward from block i to j according to program execution. (When determining interval size, we exclude the access to i and j.)

Usually, it is more convenient to average affinity interval.

Definition. For a memory access sequence and a pair of memory blocks, let $\mathcal{SI}(i, j)$ represent the set of access intervals that partition the sequence. That is, each element $\alpha = (a, b)$ in $\mathcal{SI}(i, j)$ is an interval $\mathsf{SI}_1(i, j)$ with access indices a and b, where a and b access blocks i and j, respectively. Then, average *spatial interval* $\mathcal{SI}(i, j)$ is

$$\mathsf{SI}(i, j) = \frac{\sum_{\alpha \in \mathcal{SI}(i,j)} \mathsf{SI}_1(\alpha_i, \alpha_j)}{|\mathcal{SI}(i, j)|} \tag{1}$$

3.3 Spatial anticipation

Spatial anticipation $\mathsf{SA}(j|i)$ captures *directional* spatial locality or the probability that j is accessed after i. It is a real number within $[0, 1)$, where higher is better.

Definition. For block pair (i, j), *spatial anticipation* is the ratio of a) the *number* of intervals in the spatial interval set $\mathcal{SI}(i, j)$ and b) $\mathsf{A}(i)$.

$$\mathsf{SA}(j|i) = \frac{|\mathcal{SI}(i, j)|}{\mathsf{A}(i)} \tag{2}$$

As $\mathsf{SA}(j|i)$ defines anticipation of j to i, intuitively it is attributed to prefetching (both measure and preload).

3.4 Spatial density

Spatial density emphasizes the *hotness* of j over the *lifetime* of i. It is the probability of accessing block j between accesses to block i. It is a real number within $[0, 1)$, where higher is better.

Definition. For block pair (i, j), *spatial density* $\mathsf{SD}(j|i)$ is the ratio of a) accesses to j within b) the self-spatial interval of i:

$$\mathsf{SD}(j|i) = \operatorname*{avg}_{\alpha \in \mathcal{SI}(i,i)} \frac{\mathsf{A}_\alpha(j)}{\mathsf{SI}_\alpha(i, i)} \tag{3}$$

$$\approx \frac{\mathsf{A}(j)}{\sum_\alpha \mathsf{SI}_\alpha(i, i)} = \frac{\mathsf{A}(j)}{\sum_\alpha \mathsf{L}_\alpha(i)} \tag{4}$$

$$\approx \frac{\mathsf{A}(j)}{\mathsf{L}(i)} \tag{5}$$

where $\mathsf{A}_\alpha(j)$ represents number of j within interval α.

Equation (5) approximates Eq. (3) in two ways. First, the sum of fractions in Eq. (3) is converted into separate sums in the numerator and denominator. Second, it approximates the denominator in Eq. (4) with lifetime of i, which is equal to $\mathsf{A}(i) + \sum_\alpha \mathsf{SI}_\alpha(i, i)$.

$\mathsf{SD}(j|i)$ defines *nearby reference correlation* of j to i, and is used to measure caching with the assumption of a fully associative cache. Note that, footprint [45, 46] as a metric adequately analyzes caching and associativity. But, footprint by definition uses time windows, whereas $\mathsf{SD}(j|i)$ is defined w.r.to accesses to location i and includes spatial locality.

3.5 Weighted affinity

In practice, it is most important to understand spatial affinity for memory blocks that are frequently accessed.

Definition. For memory block pair (i, j), we define a weighting factor (between $(0, 1]$) based on access intensity $\mathsf{AI}(i)$ (normalized access frequency) within memory region r.

We represent weighted SA and SD by scaling the respective metric in an obvious manner: $\mathsf{AI}(i) \cdot \mathsf{SA}(j|i)$ or $\mathsf{AI}(i) \cdot \mathsf{SD}(j|i)$.

Note that the weighting factor applies to the *reference* location (i) only.

The resulting weighted affinities have desirable properties. For example, if spatial anticipation SA is large, but i is cold, *weighted* SA will be small, which is correct. Alternatively, if i is hot but SA is small, *weighted* SA will be small, which again is correct.

If spatial density SD is large, but i is cold, *weighted* SD will be small, which is correct. Alternatively, if i is hot but SD is small, *weighted* SD will be small, which again is correct.

4 Spatial Affinity Scores

As previously observed (§2), relating spatial interval with each of spatial anticipation and density yields powerful interpretive insights. For example, good spatial anticipation is more impactful when spatial interval is also good. To quantify these insights, we develop affinity *scores* SD^* and SA^* that coordinate the strengths of each of these affinity relations. For example, $\mathsf{SD}^*(j|i)$ captures pair-wise $\mathsf{SD}(j|i)$ and 'goodness' of $\mathsf{SI}(i, j)$ with a single metric.

4.1 Scores

To define scores, we characterize the goodness of spatial intervals with a *coefficient* $\gamma(i, j)$, based on the observation that smaller SI is generally better. The coefficient maps smaller intervals to higher values. We then define affinity scores with respect to an affinity block pair (i, j) and its $\gamma(i, j)$.

Definition. For each affinity pair (i, j) *spatial anticipation score* $\mathsf{SA}^*(j|i)$ is

$$\mathsf{SA}^*(j|i) = \gamma(i, j)\mathsf{SA}(j|i) \tag{6}$$

Definition. Similarly, for each location i and its affinity block j, $\mathsf{SD}^*(j|i)$ is

$$\mathsf{SD}^*(j|i) = \gamma(i, j)\mathsf{SD}(j|i) \tag{7}$$

The *goodness coefficient* is based on the *goodness rank* $g(i, j)$ for the spatial interval $\mathsf{SI}(i, j)$. The rank ranges from 1 to n_r (lower is better); we often choose $n_\mathsf{r} = 5$ to cover interval values in each sample of memory access sequences in the trace. Since a smaller SI is better, we define the rank as:

$$g(i, j) = \min\left(n_\mathsf{r}, \left\lfloor \frac{\mathsf{SI}(i, j)}{n_\mathsf{si}} + 1 \right\rfloor\right) \tag{8}$$

We suggest two ways to set n_si that are independent of workload. One is relative to the load queue size (loads in flight). Another is loads to fill some fraction of cache lines. Our experiments use 1/4 the load queue size.

The goodness coefficient $\gamma(i, j)$ maps better ranks to higher values in $(0, 1]$:

$$\gamma(i, j) = \frac{n_r - g(i, j) + 1}{n_r} \tag{9}$$

4.2 Multi-resolution scores

To facilitate top-down analysis using scores, we extend scores to multiple resolutions, enabling both rapid high-level comparisons and detailed understanding. We aggregate the affinities for each individual *reference* block as a weighted sum, and then aggregate that for a region of blocks.

For each reference block i in region r, the aggregate *spatial anticipation and density scores* are, respectively, $\mathrm{SA}^*(i)$ and $\mathrm{SD}^*(i)$:

$$\mathrm{SA}^*(i) = \sum_j \mathrm{AI}(i) \cdot \mathrm{SA}^*(j|i) \;\forall\; i \neq j \tag{10}$$

$$\mathrm{SD}^*(i) = \sum_j \mathrm{AI}(i) \cdot \mathrm{SD}^*(j|i) \tag{11}$$

where $\mathrm{AI}(i)$ is the affinity weight (§3.5).

The corresponding scores for region r are:

$$\mathrm{SA}^*_r = \sum_i \sum_j \mathrm{AI}(i) \cdot \mathrm{SA}^*(j|i) \tag{12}$$

$$\mathrm{SD}^*_r = \sum_i \sum_j \mathrm{AI}(i) \cdot \mathrm{SD}^*(j|i) \tag{13}$$

4.3 Potential vs. realized score

We distinguish between *potential* and *realized* affinity to focus on possible candidates for data layout optimizations. *Realized* represents affinity in the current layout, and *potential* indicates affinity that could be realized with changes to access order or data layout. For a *reference* location i, *realized* score includes a subset of affinity locations (j) such as *hot-lines* and 'relevant contiguous locations' only (SA^*: +1, +2 offsets; SD^*: −1, +1 offsets, and 'self'), whereas *potential* score includes all affinity locations.

5 Affinity Signatures

We develop affinity signatures that concisely represent spatial-temporal affinity in accesses, enable the use of our diagnostic rules to reason about likely performance, and compare between variants and/or different data structures of applications.

5.1 Affinity heatmap

For a given affinity metric, an affinity heatmap is a matrix that represents three affinity dimensions: *reference location* i, *affinity location* j, and the affinity value for the pair (i, j). A signature can include heatmaps for multiple metrics and is thus 4D.

Consider the *affinity matrix* in Fig. 2(a). Matrix columns denote *reference locations* (i), rows denote *affinity locations* (j), and each (i, j) cell represents a metric value. Matrix rows for j are organized into two bands. The lower band shows contiguous locations that are spatially related to block i, and demonstrate locality within contiguous locations. An upper band shows other blocks that highlight opportunities for reorganization, for example, hot blocks in execution, pages or regions.

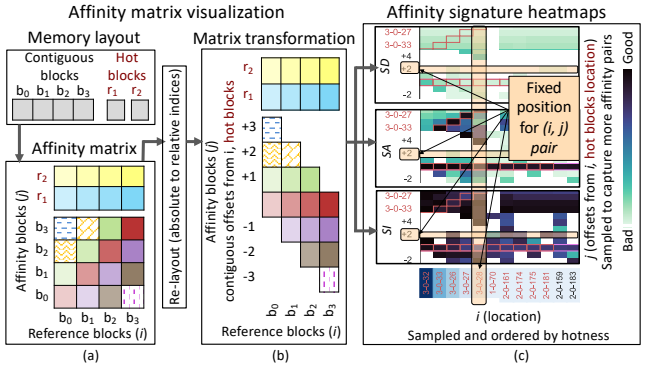


Figure 2: Affinity heatmap visualization and signature. Given (a) memory layout and affinity matrix, (b) visualization transforms j indices. Contiguous blocks b_0 - b_3 are converted to relative offsets (+/- from i, b_1 is +1 for b_0, and b_1 is -1 for b_2) shown at the bottom, and hot blocks without a spatial relation to reference blocks are shown at the top. MemFriend 's heatmaps display sampled offset locations within a range (+/- 256) and hot lines in execution. Affinity signature (c) combines heatmaps for all three metrics SD, SA and SI. Each (i, j) pair has fixed position (aligned) in all three heatmaps.

Fig. 2(b) shows the transformation into a heatmap. Note that in this example, the cell (colors) represents the transformation rather than the normal values. The heatmap's *affinity locations* in the lower band are transformed to *relative addresses*, i.e., offset (+/-) locations. Blocks b_0 - b_3 that are spatially contiguous are represented by relative offset locations (b_1 is +1 for b_0, and b_1 is -1 for b_2). Hot blocks r_1 and r_2 are shown as absolute locations.

Figure 2(c) shows the *signature*, a combination of three heatmaps. MemFriend's visualization shows SD, SA and SI heatmaps separately rather than combined scores because it allows a) quick estimation of spatial-temporal locality based on SI and b) preserves the qualifying distinction implied by SI for both SD^* and SA^*. The signature includes the affinity value dimension, resulting in a 4D figure.

Each of the three heatmaps (SD, SA and SI) in Fig. 2c shows *reference locations* that are sampled based on *hotness* and ordered by it, and *affinity locations* that are sampled to capture most affinity pairs. Each (i, j) pair has a fixed location across the three heatmaps. *Affinity locations* show *hot-lines* in upper band with absolute locations. *Hot-lines* are a subset of the hottest (cache-line sized) blocks in the application and highlight temporal locality and also opportunities for reorganization.

5.2 Comparing affinity signatures

It is important to compare signatures between variants of an application. To compare signatures, we introduce an alternative 3D/4D representation along with methods for condensing signatures into lower dimensional representations.

5.2.1 Affinity signal. An *affinity signal* shows the same three dimensions as the affinity matrix but changes the grouping. Figure 5 shows a signal plot with four subplots. The top three subplots show the most important affinity locations (j), i.e., relative offsets that

are essential to study reuse and the impact of prefetching. These subplots show scores SD* and SA* that integrate SI for each (i, j) pair (§4). The fourth subplot shows the access frequency of i to focus on access distribution between blocks. These plots show affinity values for all observed *reference locations* and are ordered by address locations.

5.2.2 Affinity histogram. To give a high level perspective, an *affinity histogram* shows the distribution of affinity values. To compare impacts, we focus on affinity *scores* and prefer a continuous probability density function (PDF). Figure 6 shows a histogram as a PDF for each spatial affinity score. The result condenses three dimensions in the signature, namely the range of affinity metrics and both affinity dimensions (the reference and affinity axes).

For example, if the SD* value for pair (2, 3) is 0.3, it is represented in the histogram bin for SD* 0.3. To pinpoint critical affinity relations, histograms focus on the *hot-cluster* section in the heatmap that covers significant *reference locations* (*hot* subset of i), and all its *affinity locations*. Both SD* and SA* are real numbers, with high values indicating better affinity; hence higher number of block pairs in the high-valued score bins indicates better locality.

5.2.3 Affinity vector. An *affinity vector* represents each affinity score as a scalar, where each scalar compresses a single histogram. The scalar includes the importance of i, as defined in *weighted affinity* (§3.5).

6 Affinity Analysis and Zooming

As a pair-wise analysis, naive affinity analysis requires quadratic space and time $O(A^2)$, where A represents unique memory blocks in a trace. This type of analysis, which views addresses at a single resolution, is only practical when address blocks are large, i.e., objects (*regions*) or pages. For efficient analysis at smaller levels of memory blocks, i.e., word or cache line, we use multi-resolution analysis that focuses on smaller but significant (*hot*) segments within *regions of interest* that are identified using location-based zooming.

6.1 Multi-resolution analysis & complexity

Fig. 3(a) shows single resolution and Fig. 3(b) shows multi-resolution analysis. Within the entire memory region (A blocks) single resolution analysis is applicable at the region level for *regions of interest* shown in red boxes. Within the *regions of interest*, multi-resolution analysis chooses *hot* sub-region (R blocks) marked in a blue box in Fig. 3(a), and applies (cache-line sized) block level resolution.

Multi-resolution analysis significantly reduces analysis complexity in practice. The number of blocks A in the entire memory region is far greater than the blocks R in sub-regions. Blocks in excluded regions are consolidated into segments that cover the region and are considered as *significant locations* in the analysis. For a sub-region with R blocks, time and space complexity for multi-resolution analysis is $O(NR)$, where N includes R and other *significant locations*.

6.2 Zooming

To find *regions of interest* in the memory access trace, we use zooming as shown in Fig. 3(c). Zooming uses a recursive tree structure. First, the entire memory region is divided into heap and stack. The heap segment is chosen as the root node for further examination.

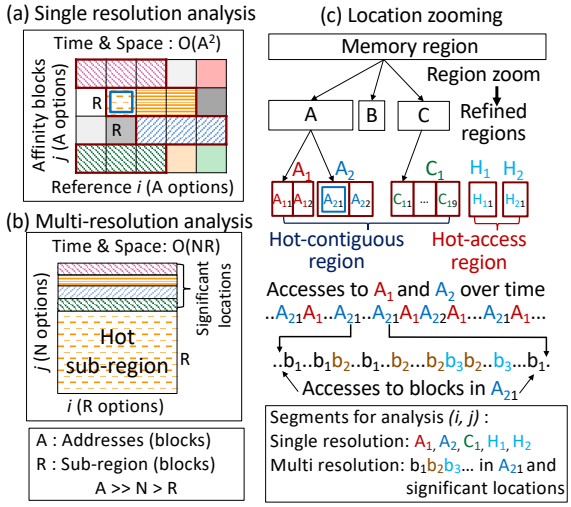


Figure 3: Affinity analysis methods and location zooming. (a) Single resolution focuses on *regions of interest* in red boxes for region level analysis. (b) Multi-resolution analysis chooses a *hot* sub-region within *regions of interest* (marked in a blue box in (a)) for block level analysis. (c) Location zooming shows recursive filtering for finding *regions of interest* A_1, A_2, C_1, H_1 and H_2. Sample access sequence shows expanded access to blocks b_1, b_2 and b_3 in sub-region A_{21}. Single-resolution analyzes affinity between A_1, A_2, C_1, H_1 and H_2. Multi-resolution analyzes affinity between blocks in sub-region A_{21} and *significant* locations.

Zooming proceeds top-down and counts access frequencies at each level. Zooming identifies two types of *hot* regions: *hot-contiguous* and *hot-access*.

To find *hot-contiguous* regions, each examined segment is divided into fixed-size (s_c) chunks. Contiguous chunks with an aggregated access frequency of at least t% of the parent's access are identified as sub-regions, e.g., A, B, and C. t is a controllable parameter; we typically use 10 based on empirical evidence. Sub-regions are analyzed at further levels using recursively reduced sizes for s_c. Zooming ends when s_c reaches a specific size (configurable). The final *hot-contiguous* regions in Fig. 3(c) are A_1, A_2, and C_1.

To find *hot-access* regions, regions with high access counts are identified by filtering *hot* instructions and their associated regions from the trace. Qualifying sub-regions H_1 and H_2 with size s_c are identified using the same access frequency policy, where segments with t% of the parent's access qualify as a child.

The final *regions of interest* are A_1, A_2, C_1, H_1, and H_2. From these regions, multi-resolution analysis is applied to blocks in sub-region A_{21}, and *significant locations* can include A_1, A_2, C_1, H_1, and H_2.

7 Evaluation

We evaluate spatial affinity metrics using a set of representative HPC application benchmarks that vary data structures, storage formats, data layouts and algorithms. We describe the novel insights gained from affinity analysis. As a baseline, we compare against

reuse distance (RD) and show the effectiveness of affinity metrics in distinguishing application performance.

Benchmarks. We study variants (OpenMP implementations) in the benchmarks listed below.

- Reordering of sparse tensors using HiParTI-HICOO [24]
- Sparse matrix storage formats using SpMM kernel in HiParTI-HICOO [23]
- Graph clustering using miniVite [10]
- Large-language models (transformers) via Alpaca.cpp [9]
- Particle transport algorithm using XSBench [41]

Trace collection. We conduct experiments on an Intel Core i9-12900KF (Alder Lake hybrid) with 8 Performance-cores and 8 Efficient-cores. Binaries are instrumented using open-sourced MemGaze [20] framework to capture memory access trace based on Processor Tracing hardware. A mapping table for translating instrumented binary to the original is used to attribute regions to data objects.

Trace consists of samples of access sequences (~250) and includes instruction address, memory address and sample IDs. The length of sequences in a sample as well as trace sizes depend on sampling rate. The sampling rate varies between benchmarks: miniVite uses 5M, HiParTI-HICOO uses 4M, and XSBench uses 5M. Trace collection imposes overheads from 2.5× to 6×. Trace files for HiParTI-HICOO matrix variants with 220 K to 4.7 M access sequences have sizes from 11 MB to 233 MB.

Analysis Overhead. Affinity analysis time depends on the number of sub-regions considered for multi-resolution analysis and trace size. In the case of HiParTI-HICOO matrix variants with trace sizes from 11 MB to 233 MB, and 10 sub-regions, analysis runtime ranges from 2 seconds to 20 seconds. Signature visualization of heatmaps, histograms, and signals take ~20 seconds.

7.1 HiParTI-HiCOO

To study sparse matrix storage formats and tensor data layouts, we choose the HiParTI suite [22]. In HiParTI, SpMM kernels incorporate matrix storage formats such as *compressed sparse row* (CSR), *coordinate* (COO), and HICOO [23]. It also includes multiple tensor reordering variants [24]. HICOO is a compressed `block` storage format for sparse tensors and matrices; it derives from and improves upon the COO format.

7.1.1 Tensor reordering variants. For computational efficiency, sparse tensor data is generally reordered (indices relabeled) to improve data access locality. We analyze locality patterns of tensor reordering variants: Default (no reorder), Random (random order), BFS (breadth first search-like heuristic approach) and Lexi (lexicographical order) [24]. All variants are integrated in an MTTKRP kernel implementation with HICOO storage format. Benchmark is run with *nell-2* [35], a third-order tensor with 77M nonzeroes and a density of 2.4×10^{-5}.

Aff. Heatmap. Figure 4 shows affinity heatmaps for the hottest memory region. The region includes two objects *factor matrices & output*. Heatmaps for the variants show different affinity patterns across reference locations (horizontal axis) for each variant. We make four important observations from the heatmaps.

Table 1: HiParTI-HICOO tensor reordering variants: Data locality and affinity vector.

Region	Variant	Run time	A	RD	Realized		Potential	
					SA^*_r	SD^*_r	SA^*_r	SD^*_r
factor matrices & output	Default	9.5 s	1.83M	2.53	0.3	0.15	0.4	0.2
	Random	9.0 s	1.81M	2.67	0.4	0.2	0.5	0.2
	BFS	4.4 s	1.84M	5.07	2.9	0.5	3.5	0.5
	Lexi	3.4 s	1.81M	3.24	3.3	0.7	5.3	0.8

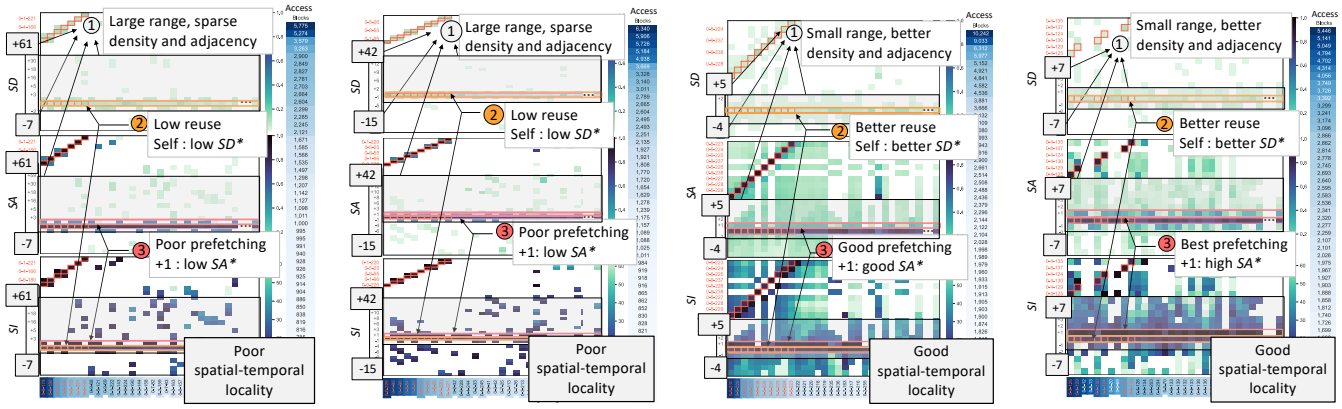
First, consider the SD metric. Figures 4a and 4b show Default and Random variants. In these signatures notice that (a) the shaded box in the SD heatmap shows sparse and irregular affinity, (b) the shaded box in the SI heatmap for the same locations shows low (good) values, and (c) ① shows a large range of contiguous affinity locations (offsets on the vertical axis, +61 to -7 for Default and +42 to -15 for Random) that extend beyond the shaded box. These observations point that distant affinity locations are accessed at closer intervals and there is minimal correlation between accesses to a block and its neighbors. Hence the two variants have no spatial-temporal locality. Figures 4c and 4d show BFS and Lexi variants. In these signatures, observe that (a) the shaded box in the SD heatmap shows high affinity to closely located offset locations, (b) the shaded box in the SI heatmap for the same locations shows increasing SI values with the increase in offset distance, and (c) affinity neighborhood in ① shows a smaller range (+5 to -4 for BFS, and +7 to -7 for Lexi). This pattern with high affinity to adjacent locations and good intervals in BFS and Lexi, exhibits good spatial-temporal locality and complies with guideline §2.1 for *good performance*.

Second, consider SD^* for *self*. ② combines values from SD and SI heatmaps, and shows that SD^* is low across all variants. Though BFS and Lexi have slightly better SD^*, the improvement is small. This suggests that although temporal reuse increases (along with caching opportunities) in BFS and Lexi, the impact will be minor.

Third, consider the SA metric. Figures 4a and 4b show Default and Random variants. In these signatures notice that (a) the shaded box in the SA heatmap shows sparse SA values, (b) SI heatmap for the same locations shows low (good) values, and (c) ① shows that both SA and SI extend to a wider range of affinity locations. SA values are scattered between affinity locations with minimal access proximity between neighbors, suggesting that the access pattern is irregular and spatially sparse in Default and Random. In contrast, in Figs. 4c and 4d for BFS and Lexi notice that (a) the shaded box in the SA heatmap shows better SA and high access proximity to adjacent locations, and (b) as discussed in SD, both SI values and affinity neighborhood range comply with *good performance*. For BFS and Lexi, the observations point to irregular access but with good spatial-temporal locality.

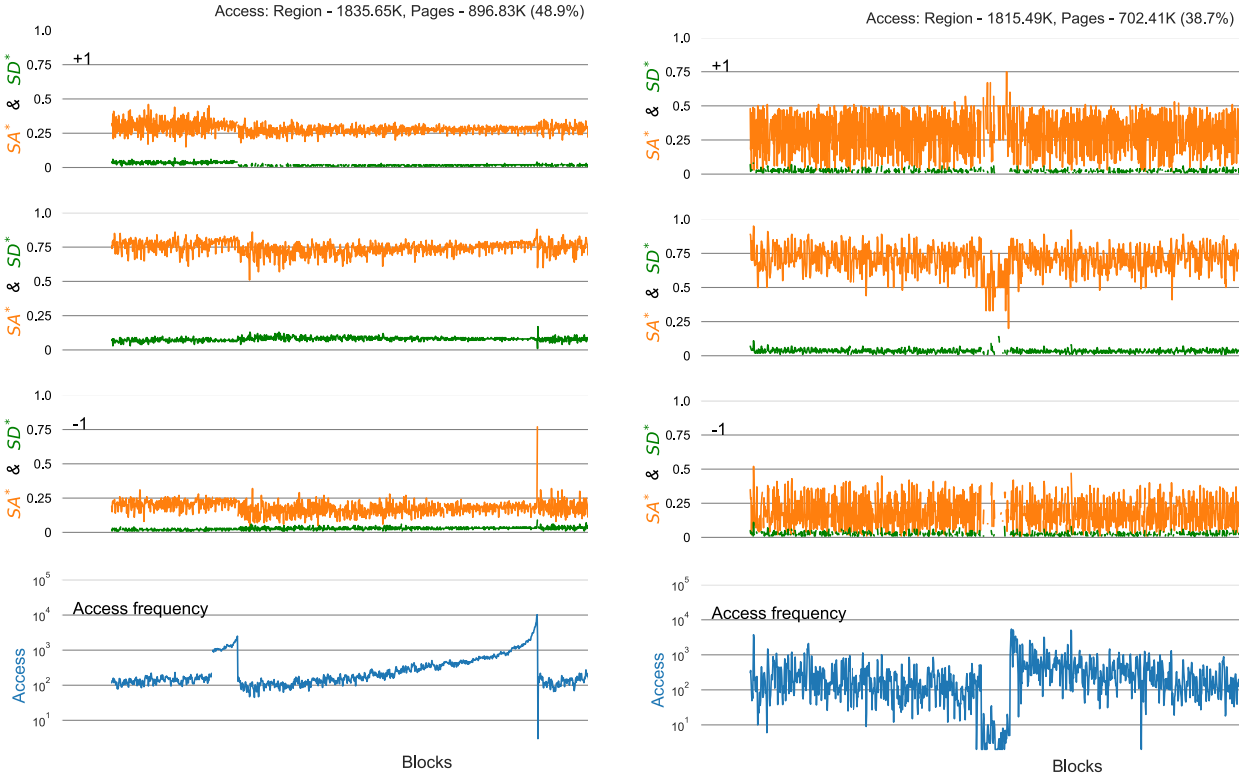
Fourth, consider SA^* for +1 offset location. ③ combines values from SA and SI heatmaps, shows SA^* is high for both BFS and Lexi, whereas Default and Random have no measurable value. This highlights that BFS and Lexi have high anticipatory spatial-temporal locality to +1 offset location and will highly benefit from hardware prefetching.

We conclude that (a) affinity metrics distinguish the performance of the variants and (b) explain that the tensor reorderings primarily improve SA in contrast to SD. The latter means that the reorderings



(a) Default **(b) Random** **(c) BFS** **(d) Lexi**

Figure 4: HiParTI-HICOO tensor reordering variants: Affinity heatmaps for *factor matrices & output* region.

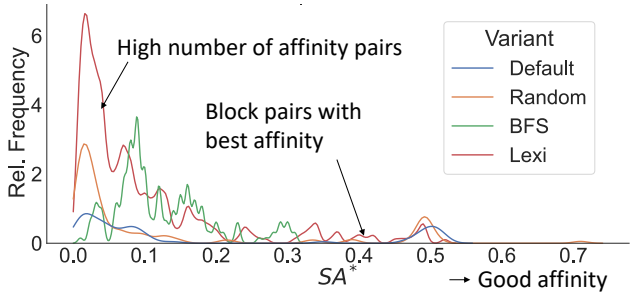


(a) BFS **(b) Lexi**

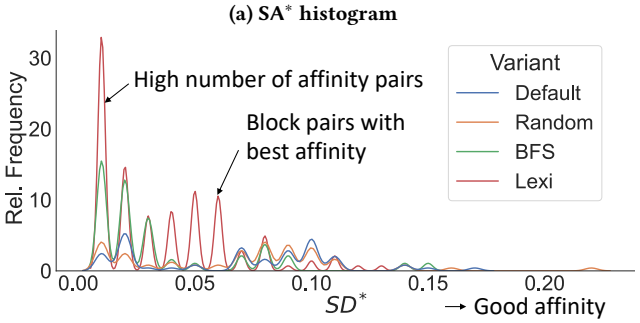
Figure 5: HiParTI-HICOO tensor reordering variants: Affinity signal plots for a section of blocks in *factor matrices & output* region.

primarily depend upon hardware prefetching for impact. Our experiments with disabled prefetchers confirm that runtime degrades from 8% to 28% for Lexi and BFS, over a range of large tensors (*nell-2* and *nell-1* [35], *freebase-music* [17]).

Aff. Signal. We study affinity over the *memory address space* for BFS and Lexi with affinity signal plots (Figs. 5a and 5b) that show SA* and SD* for offset locations (−1 to +1). We focus on SA* values in the plots, as SD* is low for all blocks in both variants. SA* for +1 offset locations in BFS is stagnant at 0.25, whereas Lexi has a significant number of blocks with 0.5. This shows that Lexi has a higher number of blocks with more affinity to adjacent locations than BFS, and these blocks leverage hardware prefetcher to



(a) SA* histogram



(b) SD* histogram

Figure 6: HiParTI-HICOO tensor reordering variants: Affinity score histograms, distribution of block pairs in *hot-cluster* for *factor matrices & output* region.

decrease load latency. Access frequency plots at the bottom show that Lexi has less access frequency variation between blocks than BFS. It indicates that memory locations in Lexi are accessed more frequently with more use of the same location, consistent with more reuse and perhaps more memory-level parallelism. Again, we see that SA* clearly explains the better performance in Lexi.

Aff. Histogram. Figure 6a and Fig. 6b show the distribution of affinity pairs for SA* and SD* for all variants. In these plots, a distribution is better when it has more area that is skewed "up and to the right". Figure 6a shows that Lexi has more affinity pairs (upper section) and affinity pairs at all score levels, including high values (right section). Though Default and Random have few affinity pairs at high score values, their *weighted affinity* (§3.5) is low. The distribution in Fig. 6b shows that pairs in all variants are concentrated towards the lower SD* values (left section). Though Lexi has a better SD* distribution, the plot does not identify a clear winner. ***Thus we see that affinity heatmaps, signals, and histograms all explain the better performance of BFS and Lexi by focusing attention on the impact on SA*.***

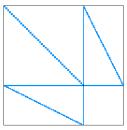
Aff. Vector. First, we consider the baseline locality metric RD in Table 1 shows the best (lowest) value for Default and Random variants, and the worst value for BFS. We observe that RD can be highly misleading as an indicator of application performance.

Second, consider *realized* scores. Recall that *realized* score quantifies affinity to adjacent locations in the current layout. SA^*_r in Table 1 shows high values for Lexi and BFS, indicating the benefits of reordering. Lower *realized* SA^*_r value for Default and Random represent the sparse layout and access pattern. Though *realized* SD^*_r is higher for Lexi, the minimal range of values does not distinguish between variants.

Table 2: HiParTI Matrix variants: Data locality and SD*.

Matrix	Variant	Run time	A	RD	SD* (Avg)
B	CSR	5.4 s	157K	0.95	0.25
	COO-Reduce	8.0 s	627K	0.99	0.41
	HICOO-S	4.7 s	156K	0.95	0.26

A's structure



Third, consider *potential* scores. Recall that *potential* scores represent a possible value for affinity under the assumption that layout can be changed dynamically without cost, but it introduces overheads. *Potential* vectors in Table 1 also show high values for Lexi. In Lexi, the increase in *potential* score for SA^*_r is attributed to the -1 offset location. For the detailed explanation, recall that the -1 offset location has high temporal proximity (lower SI, high SA in Fig. 4d and high SA* in Fig. 5b). It is possible that the -1 offset location is present in the cache from prior access, and actual *realized* affinity is higher. *Potential* SD^*_r values are similar to *realized* as SD values are concentrated towards *hot-lines*, and as in *realized* SD^*_r the values are low and do not distinguish between variants.

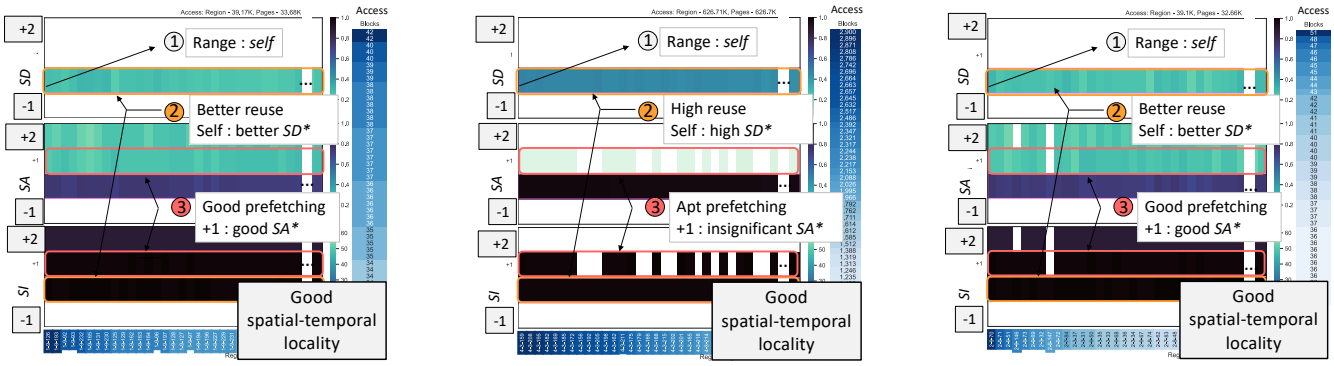
7.1.2 Matrix variants. We analyze memory locality effects of sparse matrix storage formats CSR, COO and HICOO in SpMM kernel. SpMM kernel computes $C = A \times B$ where A is a sparse matrix, B and C are dense matrices. In our configuration, A (blockqp1 [6]) has a density 1.77 x 10^{-4} and the block structure is shown in Table 2; number of columns in B is set to 4096. We focus on efficient parallel implementations and select CSR, COO-Reduce, HICOO-S variants for analysis. CSR parallelizes rows in A. COO-Reduce parallelizes the number of non-zeroes and uses a buffer for C. HICOO-S parallelizes compressed `blocks`.

Aff. Heatmap. Fig. 7 shows heatmaps for the hottest memory region, matrix B in all variants. It shows uniform locality patterns where the affinity pattern remains the same across all reference locations (horizontal axis). We make four observations from the heatmaps.

First, consider the SD metric. In all three variants, (a) yellow box in the SD heatmap shows that the variants have high SD values for *self* location only, (b) yellow box in the SI heatmap shows low (best) SI values for the same affinity location *self*, and (c) ① shows affinity range contains *self* location only. These observations indicate that no other affinity locations are accessed within the lifetime of each reference block, and blocks are reused within a short interval.

Second, consider SD* for *self*. ② combines SD and SI heatmaps and shows high values in all variants, indicating that reuse is high, with the best values in COO-Reduce.

Third, consider the SA metric. For CSR in Fig. 7a and HICOO-S in Fig. 7c, SA is high and similar across contiguous locations, and SI remains low (good) for all reference blocks. For COO-Reduce in Fig. 7b, SA is not uniform and the values are insignificant, as it incorporates the highest access counts of reference blocks, but note that it is accompanied by good SI values. Though SA for COO-Reduce differs from the other two, SI remains similar. These observations



(a) CSR **(b) COO-Reduce** **(c) HICOO-S**

Figure 7: HiParTI matrix variants: Affinity heatmaps for matrix B.

point to regular, forward traversal access patterns in all three variants.

Fourth, consider SA* for +1 offset location. ③ combines SA and SI heatmaps and shows high values in CSR and HICOO-S variants, indicating that they effectively leverage the prefetcher. Though SA* for +1 offset location is low in COO-Reduce, it is accessed in a temporal forward direction.

To summarize, for all variants (a) range of affinity blocks is *self*, (b) blocks have high reuse, and (c) SI heatmaps show effective prefetching. These observations point to regular, strided access with high spatial-temporal locality and suggest that all variants follow guideline §2.1 behavior for *good performance*, but there are differences in the rate of reuse. We conclude that same-line locality is most important for the matrix variants. Although this type of locality does not strictly correspond to either caching or prefetch schemes, it is important and captured by our affinity metrics.

Aff. Vector. Baseline RD values for all variants in Table 2 are equal (best value of ~1) and indicate similar behavior, whereas distinct SD* values show the trade-off between storage format and locality. Note that COO-Reduce with the highest SD* value (high reuse) is not the best-performing variant, because it requires additional buffering as shown in very high access counts (A). Though CSR and HICOO-S have smaller SD* values, they are efficient because of effective data reuse based on instruction level parallelism (A in Table 2 are 4× to account for SSE).

It is worth noting that higher reuse, measured by SD* values translates to superior performance within the bounds of access frequency. So, ***spatial affinity metrics combined with other characteristics such as access frequency provide complete information about performance.*** At the same time, ***spatial affinity metrics provide precise spatial-temporal locality measure than reuse distance (even with best value ~1).***

7.2 miniVite

miniVite [10] is a graph benchmark for community detection that uses Louvain optimization. Variants in our analysis use different hash table implementations for the hottest *map* object. v1 uses a C++ `unordered_map`, an open hash table with an array of keys, each containing a linked list for items, and hence irregular accesses. v2 and v3 use TSL `hopscotch` [15, 38], a closed hash table that stores

Table 3: miniVite: Data locality and affinity vector for *map* object.

Region	Variant	Run time	A	RD	Realized		Potential	
					SA^*_r	SD^*_r	SA^*_r	SD^*_r
map	v1	9.1 s	301.8K	2.8	4.3	1.3	4.8	1.4
(hash	v2	6.7 s	487.4K	3.3	7.5	2.9	8.1	2.9
table)	v3	4.9 s	284.7K	2.9	6.0	1.9	7.1	2.1

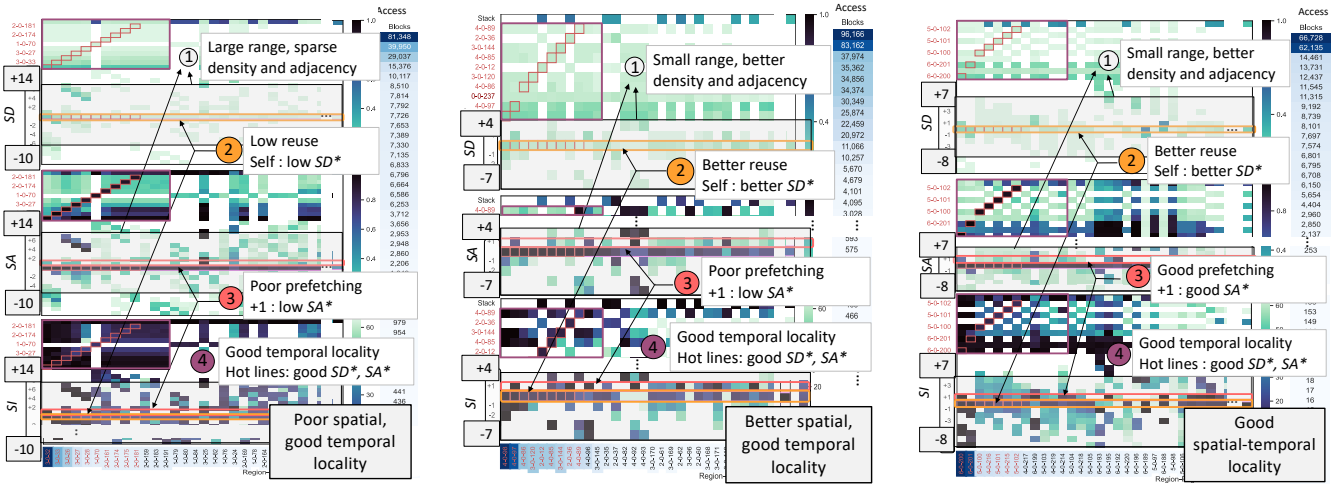
items in a contiguous array, and replaces irregular accesses with strided ones. v2 uses the default table size and does dynamic resizing. v3 avoids resizing by specifying right-size for each instance.

Aff. Heatmap. Figure 8 shows heatmaps for the *map* object, with varying affinity patterns among reference blocks (horizontal axis) in each variant. We begin with five observations.

First, consider the SD metric. In Fig. 8a for v1 observe that (a) the shaded box in the SD heatmap shows sparse and scattered affinity to contiguous locations, (b) the shaded box in the SI heatmap shows low (good) SI values for these locations, and (c) ① shows that range of affinity locations extend to a wider neighborhood (+14 to -10 offset locations in vertical axis). Scattered SD values within the range, along with good SI indicate that distant locations are accessed in close temporal proximity. This pattern points to negligible spatial and temporal affinity to adjacent locations, reflective of irregular accesses in the linked list. For v2 and v3 in Fig. 8b and Fig. 8c, note that (a) the shaded box in the SD heatmap shows SD values that are congregated towards same-line and adjacent locations, (b) the shaded box in the SI heatmap shows a trend of increasing SI with offset distances, and (c) ① for v2 and v3 show affinity to low range of adjacent locations (v2: +4 to -7, v3: +7 to -8). Concentrated SD along with a good trend in SI values indicates better affinity to adjacent locations, evident of the strided traversal of *map*.

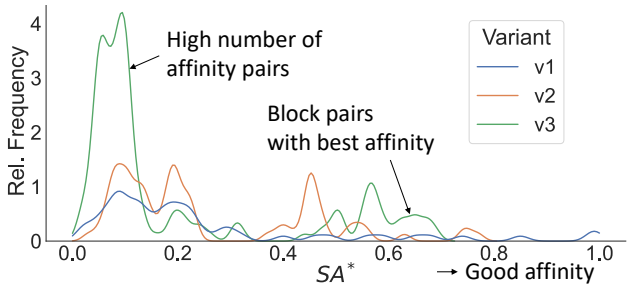
Second, consider SD* for *self*. ② combines SD and SI heatmaps and shows low SD* value in all variants. v1's SD* values are very low. Both v3 and v2 have better (slightly higher) SD* values as the traversal limits the reuse of blocks. We infer that cache-friendly and same-line reuse improves in v2 and v3, but it is still relatively low in all variants.

Third, consider the SA metric. In Fig. 8a for v1 notice that (a) the shaded box in the SA heatmap shows non-uniform SA values that are spread over contiguous locations, (b) as noted in SD discussions,

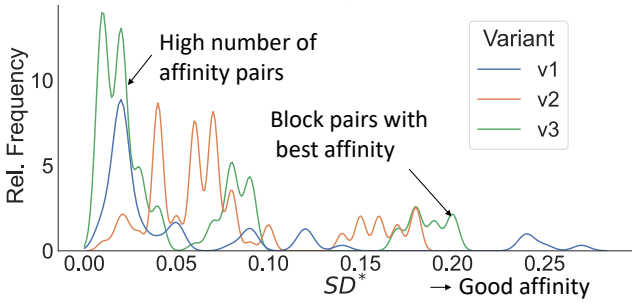


(a) v1 **(b) v2** **(c) v3**

Figure 8: miniVite variants: Affinity heatmaps for *map* object.



(a) SA* histogram



(b) SD* histogram

Figure 9: miniVite variants: Affinity score histogram, distribution of block pairs in *hot-cluster* of *map* object's affinity heatmap.

SI values also favor distant locations, and (c) ① shows v1's anticipation extends to a wide range of contiguous locations. SA correlation to distant locations shows that there is no anticipatory access to spatially adjacent locations. In Fig. 8b and Fig. 8c for v2 and v3, (a) the shaded box in the SA heatmap shows high SA values for closer neighbors, especially in v3, (b) as noted in SD discussions, SI values show preference to adjacent locations, and (c) ① for v2 and v3 show anticipation to adjacent locations. High SA values along with good SI within a much smaller range of adjacent locations, indicate anticipatory access with better spatial locality in v3.

Fourth, consider SA* to +1 offset location. For v1 in Fig. 8a ③ (combines SA and SI heatmaps) shows negligible values, indicating that v1 has no prefetching advantage. From ③ in Fig. 8b and Fig. 8c, we note that v3 has more reference locations with impactful SA* to +1 offset location, pointing to beneficial prefetching than v2.

Finally, temporal locality (④ upper band in all heatmaps) to *hot-lines* remains significant in all variants, and it is similar between all variants.

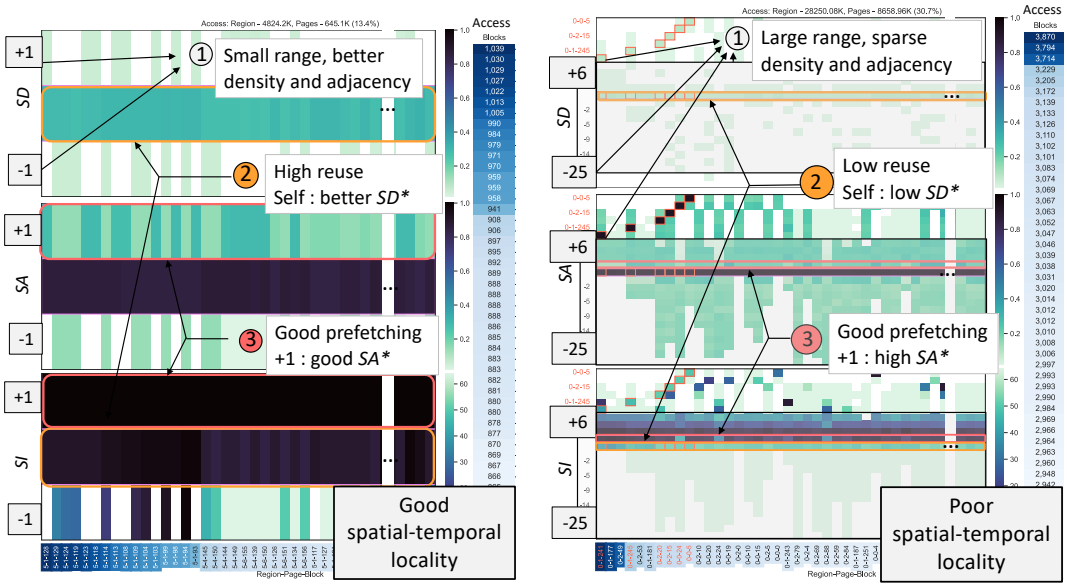
Interestingly, no variant has affinity relations that follow guideline §2.1 for *good performance*, but the SA and SI heatmaps show improving spatial locality from v1 to v3.

In summary, changing *map*'s data structure from open to closed hash table (a) improves SA than SD to adjacent locations and (b) though caching (SD) for adjacent locations remains low, it is high for *hot-lines* in all variants. We conclude that the primary explanation of v2' and v3's performance is a data structure that exploits hardware prefetching.

Aff. Histogram. Figure 9 shows histogram plots; recall that distribution is better when it has more area that is skewed "up and to the right". Figure 9a for SA* shows that v3 has high distribution of affinity pairs (upper section), with a greater number of pairs at the higher score values (right section). v3 has far more impactful affinity as the weighted value of the area under the curve increases towards high SA* values. Figure 9b for SD* shows that both v3 and v2 have a better distribution of affinity pairs than v1. Though v2 has almost comparable distribution as v3, a significant portion of the accesses in v2 are unnecessary (resizing) and hinder performance. Again, SD* shows an improving trend from v1 to v3, but SA* shows better variation and explains the better performance of v3.

Aff. Vector. First, similar values for baseline RD in Table 3 for v1 and v3 fail to differentiate the performance of these variants.

Second, consider *realized* scores for quantifying the current layout. SA^*_r shows a higher value for v3 when compared to v1, reflecting the beneficial prefetching. *Realized* SD^*_r shows high value for



(a) `src0_row` (b) `src1_col`

Figure 10: Alpaca.cpp: Affinity heatmaps for hot regions *src0_row* and *src1_col*.

Table 4: Alpaca.cpp: Data locality and affinity vector.

Region	A	RD	Realized		Potential	
			SA_r^*	SD_r^*	SA_r^*	SD_r^*
src0_row	4.8M	1.25	0.4	0.7	0.6	0.7
src1_col	28.2M	2.99	1.4	0.2	2.8	0.26

v3 than v1, indicating better reuse. In both cases, though v2 has the highest scores, the impact of the affinity score is reduced as the accesses include added copying along with traversal through the array.

Third, consider *potential* scores for possible layout changes and their impact on affinity. *Potential* SA_r^* also shows a higher value for v3. Possible options for *potential* score are spread over a small neighborhood of adjacent locations within the range, indicating that re-layout can lead to better locality. For all variants, SD_r^* doesn't show a difference between *realized* and *potential* scores as SD values are concentrated among the *hot-lines*.

7.3 Alpaca.cpp

Alpaca.cpp [9] is an LLM inferencing application implemented in C++, parallelized using Pthreads. It combines efforts such as LLaMA [39] and Alpaca [37]. Inferencing in our evaluation uses the LLaMA 7B parameter model quantized with 4-bit integers (*ggml-alpaca-7b-q4.bin*) with seed value 1685480810, and default values for other parameters.

Among the 12K lines of code (7 MB binary), location analysis highlights two regions *x* and *y* in hotspot function *ggml_vec_dot_q4_0*. *x* region corresponds to `src0_row` with spatially sparse accesses identified by *hot-access* analysis; *y* region corresponds to `src1_col` and it is a *hot-contiguous* region.

Aff. Heatmap. Heatmaps in Fig. 10 show different locality patterns with differing affinity patterns across reference locations (horizontal axis) for each region. We use signatures to differentiate the access patterns in the two regions.

src0_row. We start with `src0_row` object and its signature in Fig. 10a. First, consider the SD metric. SD heatmap shows high values for *self*, and ① shows that affinity is limited to +1,−1 offset locations and *self*. This indicates that only adjacent locations are accessed after access to a reference block.

Second, consider SD* for *self*. ② combines SD and SI heatmaps, and shows high values of SD* (~0.3) for all blocks, and points that these blocks are highly reused.

Third, consider the SA metric. SA heatmaps show high values for +1 offset than −1 offset. SI heatmap also shows low (good) SI values for +1 offset and high (bad) values for −1 offset. This shows that the +1 offset location has a high correlation and the access pattern is regular, strided access with mostly forward traversal.

Fourth, consider SA for +1 offset location. ③ shows a combination of low SI values and higher SA values, implying that +1 offset locations are accessed in close temporal proximity.

The observed pattern suggests that access behavior for `src0_row` has good spatial-temporal locality as in guideline §2.1.

src1_col. Now, we discuss `src1_col` object's signature in Fig. 10b. First, consider the SD metric. In Fig. 10b observe that (a) the shaded box in the SD heatmap shows scattered and sparse SD values, (b) the shaded box in the SI heatmap points to increasing SI for positive offset locations, and high values for *self* and negative offsets, and (c) ① shows affinity range that is higher (+6 to -25 offset locations). Scattered affinity to locations across the range, with irregular SI values, show that the access pattern in `src1_col` traverses a wide range of adjacent locations within each reference block's lifetime.

Table 5: XSBench-event variants: Data locality and affinity vector for *material* and *nuclide* region.

Region	Variant	Run time	A	RD	Realized		Potential	
					SA^*_r	SD^*_r	SA^*_r	SD^*_r
material and *nuclide*	k0	52 s	1.5M	2.2	1.24	0.95	1.24	0.95
	k1	14 s	1.5M	2.3	1.25	0.97	1.27	0.97

Second, consider SD^* for *self*. ② shows high SI and low SD (~0.1) for *self* locations, pointing that reuse remains low and is an example of guideline §2.5.

Third, consider the SA metric. Notice that (a) the shaded box in the SA heatmap shows uniform SA values for all locations, and (b) as noted in SD discussions, SI values show a preference for positive offset locations. We observe that the access pattern in `src1_col` has beneficial anticipation towards positive offset locations.

Fourth, consider SA^* for +1 offset location. ③ with high SA^* value shows that high anticipatory spatial-temporal locality to +1 offset location, and access pattern that leverages the hardware prefetcher.

We validated that decoder implementation in Alpaca.cpp uses a key-value cache for efficient inferencing. We are exploring other ways to improve locality for `src1_col`.

Aff. Vector. First, baseline RD values in Table 4 shows low value and hence better locality for `src0_row` and informs about its good temporal locality. Second, *realized* scores show high SA^*_r value for `src1_col` and low values for `src0_row` highlighting the benefits of spatial locality and prefetching in `src1_col`. Also, *realized* SD^*_r shows high values for `src0_row`, and low values for `src1_col`, pointing to low reuse in `src1_col`. Third, *potential* score SA^*_r shows that efforts to reorder accesses for improved prefetching should focus on `src1_col`. *Potential* SD^*_r doesn't show a difference to *realized* score suggesting that efforts to improve reuse might need higher levels of refactoring.

Thus, we observe that ***spatial affinity metrics provide more insights about data layout, access patterns, and their implications on memory performance, than reuse distance analysis.***

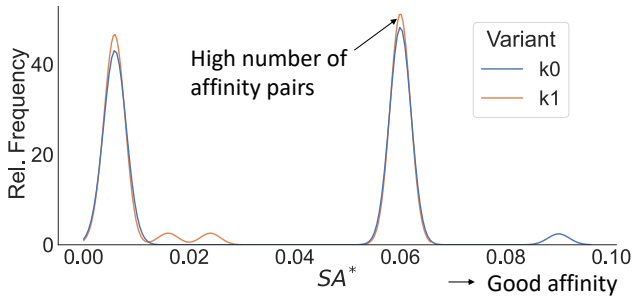
7.4 XSBench

XSBench [40, 41] is a proxy application that models Monte Carlo neutron transport algorithm, specifically the calculation of macroscopic neutron cross sections. We evaluate two variants of the *event-based* transport model, baseline k0 and optimized k1. Baseline k0 parallelizes cross section lookups for all particles with varying materials and follows random access pattern. Optimized k1 sorts the particles by material and energy and facilitates efficient memory access.

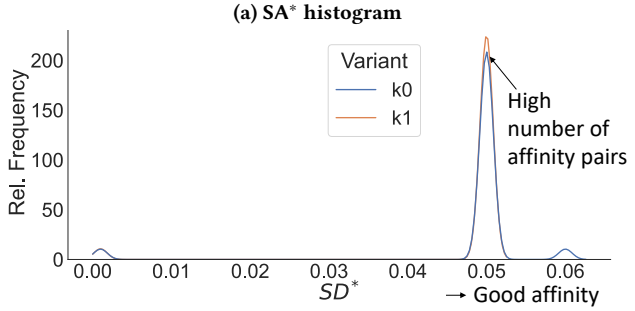
Locality analysis identifies three regions: *hot-contiguous* region *material* and *nuclide*; and two *hot-access* regions with sparse accesses, *energy grid*, and *grid cross section*.

First, we describe signatures for *hot-contiguous* region *material* and *nuclide*.

Aff. Histogram. We exclude heatmaps for *hot-contiguous* regions as both variants use the same data structure and affinity patterns are similar. Histogram of affinity pairs for *material* and *nuclide* in Fig. 11a and Fig. 11b show better number of affinity pairs (skewed "up and to the right") and better affinity in k1.



(a) SA^* histogram



(b) SD^* histogram

Figure 11: XSBench-event variants: Affinity score histogram, distribution of block pairs in *hot-cluster* of *material* and *nuclide* region.

Table 6: XSBench-event variants: Reuse distance and access frequency for *energy grid* and *grid cross section* regions.

Region	Variant	A	RD
energy grid	k0	32.7K	1.0
	k1	253.2K	7.3
grid cross section	k0	9.3M	0.0
	k1	9.4M	0.2

Aff. Vector. First, baseline RD shows better (lower) values for k0, and misleads about performance. Second, *realized* scores in Table 5 show better SA^*_r and SD^*_r values for k1, reflecting the optimal accesses in k1. Third, consider *potential* scores. The similarity between *realized* and *potential* scores suggests that *material* and *nuclide* region has the best possible layout, and optimizations should focus on other regions.

Now, we focus on sparsely accessed regions (*energy grid* and *grid cross section*). We use signals to compare against baseline RD, as the scores are not applicable due to highly sparse accesses.

Aff. Signal. Baseline RD in Table 6 for the sparsely accessed regions (*energy grid* and *grid cross section*) shows better (lower) values for k0, and misleads about the performance.

For the same regions, affinity signals are shown in Fig. 12. k0's signal plots are shown in Fig. 12a. The bottom subplot for access frequency shows sparsely accessed blocks. The middle subplot for *self* shows varying SA^* and SD^*. The top subplot for +1 offset shows no SA^* to +1 offset locations. These observations indicate worse temporal as well as spatial locality among all blocks.

k1's signal plots are shown in Fig. 12b. The bottom subplot for access frequency shows more blocks with higher access frequency.

(a) k0 **(b) k1**

Figure 12: XSBench-event variants: Affinity signal plots for a section of blocks in *energy grid & grid cross section* region.

The middle subplot for *self* shows varying SA* and SD*, but they are effective as the blocks have high access frequency. The top subplot for +1 offset shows considerable SA* to +1 locations. These observations indicate that k1 has better spatial-temporal locality. k1's signatures with better spatial-temporal affinity patterns reflect the optimized memory access patterns in k1.

We conclude that spatial affinity signatures capture differences in access patterns for regions with vastly different footprints, and the performance of k1 depends on both caching and prefetching.

8 Related Work

We introduce new spatial-temporal affinity metrics, and scalable affinity analysis that can operate at multiple resolutions.

Affinity and correlation. In the closest work [30], Miucin et al. measure co-occurence of pair of data objects within a window to detect communities between objects to guide data layout. It utilizes co-occurence within a defined window as a measure of affinity to pack objects into a cache line and reduce cache miss rate. In contrast, we use generalized windows with the inclusion of temporal locality and lifetime, and use strength of pair of metrics to identify locality signatures. Also, our multi-resolution analysis is applicable for objects as well as memory locations within large data objects.

Another related concept, *reference affinity* [49] measures affinity between two data objects based on similarities in their single-location reuse. *Reference affinity* has been extended to add more insights: weakly affined references in [48]; hierarchical data locality in [47]; conditional probability based co-occurence between pair of affine data in [26]. Additionally, Ning et al. [31] propose improved *field affinity* to measure affinity between *structure fields* to reorganize structure layouts within cache lines to increase cache reuse. Both *reference affinity* and *field affinity* follow actor-centric formulation for analyzing accesses to user guided access sites. In contrast, our location-centric analysis covers spatial-temporal locality and reuse of memory locations to describe affinity between locations.

Sobel et al. [36] present an improved algorithm (with linear complexity) for co-occurence [26] counting for multiple pattern sets. In our effort, we use zooming to restrict the pattern sets for affinity analysis, as the reduced pattern set focuses on *significant* locations, and is sufficient to construct affinity signatures. If many more pattern sets are needed for capturing affinity signature, one could use [36] as an alternative.

Spatial-temporal probability. Anghel et al. [1] introduce multi-dimensional spatial-temporal locality for single address locations, measure the probability of memory accesses with specific spatial distance and reference windows, and visualize the probability distribution as a heatmap. Our work focuses on the correlation between multiple locations and generalizes reference windows as lifetime in *spatial density*, and forward temporal distance in *spatial interval*.

Spatial locality measurements. Multiple efforts have recognized the limitations (single location, temporal dimension) of using *reuse distance* for spatial locality analysis and explored novel ways to measure spatial locality. Gu et al. [11] measure the change of reuse distance at different block granularities to identify program components to improve data layout. Multi-spectral reuse distance [4] uses reuse distance histograms at different data block sizes, and footprint metrics to study access patterns and identify appropriate page sizes. Maeda et al. [27] implement hierarchical reuse distance to preserve locality details at multiple granularities to aid in design exploration. Gupta et al. [12] measure aggregated spatial locality with specific neighborhood and window sizes to evaluate the effects of cache specifications and their impact on performance. These methods are constrained to specific neighborhood and window sizes, whereas we use multi-resolution analysis for expanding the neighborhood, and use measured temporal distance SI to quantify locality.

Along with addressing limitations of *reuse distance* analysis, added benefits in our work include: novel spatial-temporal affinity metrics and visualization to show access patterns and highlight affinity between locations; and combinations of metrics to identify opportunities for improving data layout.

Pattern analysis. Pattern analysis in various forms has been exploited in a variety of fields. We focus on pattern analysis methods used to improve prefetching in storage systems as it is related to our effort. Occurrence of frequent sequences [7, 25] in I/O access patterns is used to identify block correlations within specified windows to direct prefetching of blocks. These efforts use temporal [25] or spatial-temporal [7] information to identify correlations between blocks within specific window to improve I/O response times. These techniques exemplify the impact of correlation between multiple blocks and influence our work in finding highly correlated memory locations.

There has been recent interest in learning access patterns using machine learning. This work ranges from learning optimal replacement policies [3, 16, 34, 43] to learning patterns to assist prefetching [13, 14, 21, 32, 33, 42]. In general, these approaches require substantial training time or data. Also, most prefetching approaches are critically dependent on program control structure, not data locality.

9 Conclusions

We present novel spatial affinity metrics that capture affinity between pairs of memory locations along multiple dimensions and at multiple resolutions. We show how to characterize spatial-temporal affinity with efficient location-based multi-resolution analysis that enables analysis of full applications using large memory footprints. Our evaluations demonstrate that affinity signatures can predictively represent important and differing forms of spatial-temporal locality, including differences that arise from variant data structures, layouts, and access patterns. We conclude that spatial affinity metrics provide more interpretive insight into the impact of spatial-temporal locality compared to prior methods and metrics that rely on single location. Our future work includes applying our work to application data objects with dynamic lifetimes, memory allocation and data layouts, and hardware-software co-design.

Acknowledgments

This research is supported by the U.S. Department of Energy (DOE) through the Office of Advanced Scientific Computing Research's "Advanced Memory to Support Artificial Intelligence for Science" and "Orchestration for Distributed & Data-Intensive Scientific Exploration". PNNL is operated by Battelle for the DOE under Contract DE-AC05-76RL01830.

References

[1] Andreea Anghel, Gero Dittmann, Rik Jongerius, and R Luijten. 2013. Spatio-Temporal Locality Characterization. In *1st Workshop on Near-Data Processing*. 1–5.

[2] Amro Awad and Yan Solihin. 2014. Stm: Cloning the spatial and temporal memory access behavior. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 237–247.

[3] Daniel S. Berger. 2018. Towards Lightweight and Robust Machine Learning for CDN Caching. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks* (Redmond, WA, USA) *(HotNets '18)*. Association for Computing Machinery, New York, NY, USA, 134–140. https://doi.org/10.1145/3286062.3286082

[4] Anthony M. Cabrera, Roger D. Chamberlain, and Jonathan C. Beard. 2019. Multi-spectral Reuse Distance: Divining Spatial Information from Temporal Data. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–8. https://doi.org/10.1109/HPEC.2019.8916398

[5] Trishul M Chilimbi. 2001. Efficient representations and abstractions for quantifying and exploiting data reference locality. *ACM SIGPLAN Notices* 36, 5 (2001), 191–202.

[6] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1–25.

[7] Xiaoning Ding, Song Jiang, Feng Chen, Kei Davis, and Xiaodong Zhang. 2007. DiskSeen: Exploiting Disk Layout and Access History to Enhance I/O Prefetch.. In *USENIX Annual Technical Conference*, Vol. 7. 261–274.

[8] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. 2011. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. 365–376.

[9] Georgi Gerganov. 2023. Alpaca.cpp: Inference of Meta's LLaMA model (and others) in pure C/C++. https://github.com/antimatter15/alpaca.cpp.

[10] Sayan Ghosh, Mahantesh Halappanavar, Antonino Tumeo, Ananth Kalyanaraman, and Assefaw H Gebremedhin. 2018. miniVite: A Graph Analytics Benchmarking Tool for Massively Parallel Systems. In *Proc. of the 9th Intl. Workshop on Performance Modeling, Benchmarking, and Simulation of High-Performance Computer Systems*.

[11] Xiaoming Gu, Ian Christopher, Tongxin Bai, Chengliang Zhang, and Chen Ding. 2009. A component model of spatial locality. In *Proceedings of the 2009 international symposium on Memory management*. 99–108.

[12] Saurabh Gupta, Ping Xiang, Yi Yang, and Huiyang Zhou. 2013. Locality principle revisited: A probability-based quantitative approach. *J. Parallel and Distrib. Comput.* 73, 7 (2013), 1011–1027.

[13] Diana Guttman, Mahmut Taylan Kandemir, Meena Arunachalam, and Rahul Khanna. 2015. Machine learning techniques for improved data prefetching. In *5th International Conference on Energy Aware Computing Systems & Applications*. IEEE, 1–4.

[14] Milad Hashemi, Kevin Swersky, Jamie A Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning memory access patterns. *arXiv preprint arXiv:1803.02329* (2018).

[15] Maurice Herlihy, Nir Shavit, and Moran Tzafrir. 2008. Hopscotch Hashing. In *Distributed Computing*, Gadi Taubenfeld (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 350–364.

[16] Akanksha Jain and Calvin Lin. 2016. Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 78–89. https://doi.org/10.1109/ISCA.2016.17

[17] Inah Jeon, Evangelos E. Papalexakis, U Kang, and Christos Faloutsos. 2015. HaTen2: Billion-scale Tensor Decompositions. In *IEEE International Conference on Data Engineering (ICDE)*.

[18] Ozgur Kilic, Yasodha Suriyakumar, Nathan R. Tallent, and Andrés Marquez. 2024. MemGaze. https://github.com/pnnl/memgaze.

[19] Ozgur O. Kilic, Nathan R. Tallent, and Ryan D. Friese. 2020. Rapid Memory Footprint Access Diagnostics. In *Proc. of the 2020 IEEE Intl. Symp. on Performance Analysis of Systems and Software*. IEEE Computer Society.

[20] Ozgur O. Kilic, Nathan R. Tallent, Yasodha Suriyakumar, Chenhao Xie, Andrés Marquez, and Stephane Eranian. 2022. MemGaze: Rapid and Effective Load-Level Memory Trace Analysis. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. 484–495. https://doi.org/10.1109/CLUSTER51413.2022.00058

[21] Arezki Laga, Jalil Boukhobza, Michel Koskas, and Frank Singhoff. 2016. Lynx: A learning linux prefetching mechanism for ssd performance model. In *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 1–6.

[22] Jiajia Li and PNNL Developer Central. 2020. *HiarTI: A Hierarchical Parallel Tensor Infrastructure.* https://github.com/pnnl/HiParTI Last accessed: 2023-03-20.

[23] Jiajia Li, Jimeng Sun, and Richard Vuduc. 2018. HiCOO: Hierarchical Storage of Sparse Tensors. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 238–252. https://doi.org/10.1109/SC.2018.00022

[24] Jiajia Li, Bora Uçar, Ümit V Çatalyürek, Jimeng Sun, Kevin Barker, and Richard Vuduc. 2019. Efficient and effective sparse tensor reordering. In *Proceedings of the ACM International Conference on Supercomputing* (Phoenix, Arizona) *(ICS '19)*. Association for Computing Machinery, New York, NY, USA, 227–237. https://doi.org/10.1145/3330345.3330366

[25] Zhenmin Li, Zhifeng Chen, and Yuanyuan Zhou. 2005. Mining Block Correlations to Improve Storage Performance. *ACM Trans. Storage* 1, 2 (may 2005), 213–245. https://doi.org/10.1145/1063786.1063790

[26] Yumeng Liu, Daniel Busaba, Chen Ding, and Daniel Gildea. 2018. All timescale window co-occurrence: efficient analysis and a possible use. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*. 289–292.

[27] Rafael K. V. Maeda, Qiong Cai, Jiang Xu, Zhe Wang, and Zhongyuan Tian. 2017. Fast and Accurate Exploration of Multi-level Caches Using Hierarchical Reuse Distance. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 145–156. https://doi.org/10.1109/HPCA.2017.11

[28] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. 1970. Evaluation techniques for storage hierarchies. *IBM Systems Journal* 9, 2 (1970), 78–117. https://doi.org/10.1147/sj.92.0078

[29] Sally A McKee. 2004. Reflections on the memory wall. In *Proceedings of the 1st conference on Computing frontiers*. 162.

[30] Svetozar Miucin and Alexandra Fedorova. 2018. Data-driven spatial locality. In *Proceedings of the International Symposium on Memory Systems*. 243–253.

[31] Zhuorui Ning, Naijie Gu, Junjie Su, and Dongsheng Qi. 2022. STAFF: A Model for Structure Layout Optimization. In *2022 7th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 115–122.

[32] Leeor Peled, Shie Mannor, Uri Weiser, and Yoav Etsion. 2015. Semantic locality and context-based prefetching using reinforcement learning. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 285–297.

[33] Saami Rahman, Martin Burtscher, Ziliang Zong, and Apan Qasem. 2015. 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*. IEEE, 383–389.

[34] Zhan Shi, Xiangru Huang, Akanksha Jain, and Calvin Lin. 2019. Applying Deep Learning to the Cache Replacement Problem. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 413–425. https://doi.org/10.1145/3352460.3358319

[35] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. 2017. *FROSTT: The Formidable Repository of Open Sparse Tensors and Tools*. http://frostt.io/ Last accessed: 2023-03-20.

[36] Joshua Sobel, Noah Bertram, Chen Ding, Fatemeh Nargesian, and Daniel Gildea. 2020. AWLCO: All-Window Length Co-Occurrence. In *Annual Symposium on Combinatorial Pattern Matching*.

[37] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[38] Thibaut G. Tessil. 2019. Tessil hopscotch-map. github.com/Tessil/hopscotch-map.

[39] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[40] John Tramm, Ron Rahaman, Amanda Lund, and others contributors. 2021. *XSBench*. https://github.com/ANL-CESAR/XSBench Last accessed: 2023-03-20.

[41] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. 2014. XSBench - The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis. In *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*. Kyoto. https://www.mcs.anl.gov/papers/P5064-0114.pdf

[42] Ahsen J Uppal, Ron C Chiang, and H Howie Huang. 2012. Flashy prefetching for high-performance flash drives. In *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–12.

[43] Nan Wu and Pengcheng Li. 2020. Phoebe: Reuse-Aware Online Caching with Reinforcement Learning for Emerging Storage Models. arXiv:2011.07160 [cs.PF]

[44] Wm A Wulf and Sally A McKee. 1995. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.

[45] Xiaoya Xiang, Chen Ding, Hao Luo, and Bin Bao. 2013. HOTL: A Higher Order Theory of Locality. *SIGARCH Comput. Archit. News* 41, 1 (March 2013), 343–356. https://doi.org/10.1145/2490301.2451153

[46] Liang Yuan, Chen Ding, Wesley Smith, Peter Denning, and Yunquan Zhang. 2019. A Relational Theory of Locality. *ACM Trans. Archit. Code Optim.* 16, 3, Article 33 (Aug. 2019), 26 pages. https://doi.org/10.1145/3341109

[47] Chengliang Zhang, Chen Ding, Mitsunori Ogihara, Yutao Zhong, and Youfeng Wu. 2006. A hierarchical model of data locality. *ACM SIGPLAN Notices* 41, 1 (2006), 16–29.

[48] Chengliang Zhang, Yutao Zhong, Chen Ding, and Mitsunori Ogihara. 2004. Finding the Reference Affinity Groups in Trace using Sampling Method. http://hdl.handle.net/1802/1028

[49] Yutao Zhong, Maksim Orlovich, Xipeng Shen, and Chen Ding. 2004. Array regrouping and structure splitting using whole-program reference affinity. *ACM SIGPLAN Notices* 39, 6 (2004), 255–266.