Mid-term usability report of the SparCity framework

Deliverable No:	D5.2
Deliverable Title:	Mid-term usability report of the SparCity framework
Deliverable Publish Date:	31 March 2023
Project Title:	SparCity: An Optimization and Co-design Framework for Sparse Computation
Call ID:	H2020-JTI-EuroHPC-2019-1
Project No:	956213
Project Duration:	36 months
Project Start Date:	1 April 2021
Contact:	sparcity-project-group@ku.edu.tr

List of partners:

Participant no.	Participant organisation name	Short name	Country
1 (Coordinator)	Koç University	KU	Turkey
2	Sabancı University	SU	Turkey
3	Simula Research Laboratory AS	Simula	Norway
4	Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa	INESC-ID	Portugal
5	Ludwig-Maximilians-Universität München	LMU	Germany
6	Graphcore AS	Graphcore	Norway

CONTENTS

1 INTRODUCTION

The SparCity project is funded by EuroHPC JU (the European High Performance Computing Joint Undertaking) under the 2019 call of Extreme Scale Computing and Data Driven Technologies for research and innovation actions. SparCity aims to create a supercomputing framework that will provide efficient algorithms and coherent tools specifically designed for maximizing the performance and energy efficiency of sparse computations on emerging High Performance Computing (HPC) systems, while also opening up new usage areas for sparse computations in data analytics and deep learning.

Sparse computations are commonly found at the heart of many important applications, but at the same time, it is challenging to achieve high performance when performing sparse computations. SparCity delivers a coherent collection of innovative algorithms and tools for enabling high efficiency of sparse computations on emerging hardware platforms. More specifically, the objectives of the project are:

- to develop a comprehensive application and data characterization mechanism for sparse computation based on the state-of-the-art analytical and machine-learning-based performance and energy models,
- to develop advanced node-level static and dynamic code optimizations designed for massive and heterogeneous parallel architectures with complex memory hierarchy for sparse computation,
- to devise topology-aware partitioning algorithms and communication optimizations to boost the efficiency of system-level parallelism,
- to create digital SuperTwins of supercomputers to evaluate and simulate what-if hardware scenarios,
- to demonstrate the effectiveness and usability of the SparCity framework by enhancing the computing scale and energy efficiency of challenging real-life applications.
- to deliver a robust, well-supported and documented SparCity framework into the hands of computational scientists, data analysts, and deep learning end-users from industry and academia.

1.1 OBJECTIVES OF THIS DELIVERABLE

The main objective of Deliverable 5.2 is to provide an overview of how the software libraries, tools, and methods of the SparCity framework, which have been developed during the project's first two years, can be used in real life. These will be demonstrated by realistic examples. We also want to shed some light on the upcoming refinements and developments.

1.2 WORK PERFORMED

The content of this deliverable is a collection of elements of three categories: *methods* (Section 2), *tools* (Section 3), and *libraries* (Section 4). Each element will have a brief introduction, followed by one or two real-world examples of its actual usage and effect. If relevant, future extensions/plans of the element will also be included.

The five academic partners of SparCity (Koc, Sabanci, Simula, INESC-ID, and LMU) have contributed substantially and collaboratively to the various elements. The industrial partner, Graphcore, has contributed with technical support as well as frequent and in-depth discussions with the other partners during the first 20 months of the project. It is expected that Graphcore will continue to contribute informally to SparCity during the remaining time of the project.

1.3 DEVIATIONS AND COUNTER MEASURES

There was no noteworthy deviation from the research plan that is related to the development and application of the SparCity framework.

2 SparCity Methods

2.1 ML-based Recommendation of Sparse Matrix Storage Format and Algorithm Selection

Sparse matrix-vector multiplication (SpMV) is a key kernel in many applications from different domains, and it often turns out to be a performance bottleneck for these applications. The performance of an SpMV kernel varies widely among instances of the same size, because it depends on several factors such as the sparsity pattern and the storage format for the matrix, in addition to the architecture and memory hierarchy of the processor. This has given rise to techniques that alleviate this problem by selecting the optimal storage formats, algorithms, and reorderings for a given combination of input matrix and architecture.

2.1.1 ML-based Sparse Matrix Format Selection

Our recent work has investigated the format selection problem.[1] In contrast to existing work in the area, we focused on the portability and explainability of the ML-based recommendations. Our results indicate that ensemble learning techniques such as Random Forest or XGBoost yield excellent accuracy, as well as portability of results. Transfer learning is highly effective here, providing competitive accuracies with retraining times which are considerably less than that of the original times. In contrast to earlier work, we found that approaches based on CNNs are less viable when dealing with large instances.

2.1.2 ML-based SpMV Algorithm Selection

For a given SpMV format, there are multiple different algorithms that have different advantages and disadvantages depending on the instance. A key characteristic is the load-balancing strategy for multicore processors. Here, we distinguish between 1D (row-based) and 2D (row- and column-based) load balancings. While 1D algorithms are standard, sophisticated 2D algorithms only recently became more common.[2]

We performed a large-scale comparison of the 1D and 2D algorithms using all larger instances from the SuiteSparse collection.[3] Since the number of instances in SuiteSparse is small, we enhance the test set with 17 large matrices which are created for the collection in WP 5.5. We split the test set by the number of nonzeroes into four groups: *very small* matrices having less than 10^6 nonzeroes, *small* with 10^6 to 10^7, *medium* with 10^7 to 10^8 nonzeroes, and *large* matrices having more than 100 million nonzeroes. Performance results are averages weighted by matrix size over each group. Figure 1 shows the results for six different CPU architectures. As expected, we observe that especially processors with large core counts benefit from the 2D approach. The experiment also included reorderings, which are discussed in the next section.

Note that the overhead of using the 2D algorithm is very small and can be amortized over multiple runs. Therefore, using ML to predict the better algorithm, while possible, is unlikely to improve the time to solution.

[1] Sunidhi Dhandhania et al. "Explaining the Performance of Supervised and Semi-Supervised Methods for Automated Sparse Matrix Format Selection". *50th International Conference on Parallel Processing Workshop*. 2021, pp. 1–10.

[2] Duane Merrill and Michael Garland. "Merge-Based Sparse Matrix-Vector Multiplication (SpMV) Using the CSR Storage Format". *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2016.

[3] Scott P Kolodziej et al. "The suitesparse matrix collection website interface". *Journal of Open Source Software* 4.35 (2019), p. 1244.

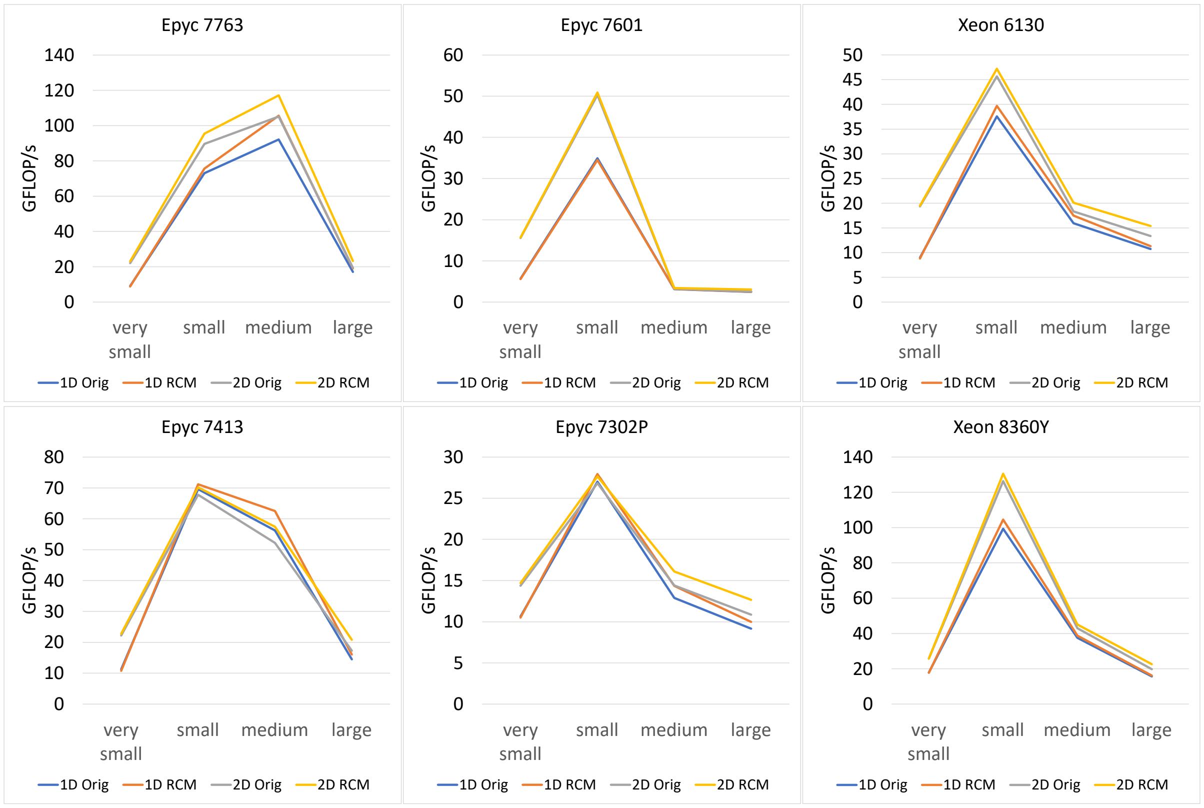


Figure 1 *Performance in GFLOP/s of the 1D and 2D SpMV algorithm by matrix size for original and RCM reordering on each each tested architecture.*

2.1.3 ML-BASED SPARSE REORDERING PERFORMANCE PREDICTION

In a typical CSR matrix with four-byte indices and eight-byte values, SpMV consumes a little more memory bandwidth than 12 bytes per nonzero when all vector elements are cached. If instead one cache line must be fetched for every vector value, the memory bandwidth requirement increases to 76 bytes on a typical CPU, thus causing a slowdown of more than 6× in the extreme case. Memory access latency may make this effect even worse.[4]

This problem can be alleviated by using Reverse Cuthill-McKee (RCM)[5] or other reordering algorithms that improve cache locality. Doing so can also indirectly improve load balance. However, when using 1D SpMV algorithms, reordering for cache reuse can also worsen load imbalance. This happens with matrices that have a widely varying number of nonzeroes per row and a relatively even distribution of longer and shorter rows among the cores. A typical example is Kronecker graphs which are generated for the Graph500 benchmark.[6] In this case, a random ordering of the rows generally ensures a good load balance for 1D algorithms. However, this load balance will typically disappear when applying an RCM reordering, since the bandwidth minimization does not consider load balance and may, e.g., cluster denser rows.

For this reason, we focus on predicting the performance gains of using the RCM algorithm

[4]Johannes Langguth et al. "Parallel performance modeling of irregular applications in cell-centered finite volume methods over unstructured tetrahedral meshes". *Journal of Parallel and Distributed Computing* 76 (2015), pp. 120–131. DOI: 10.1016/j.jpdc.2014.10.005.

[5]Elizabeth Cuthill. "Several Strategies for Reducing the Bandwidth of Matrices". *Sparse Matrices and their Applications*. Springer, 1972, pp. 157–166.

[6]Richard C Murphy et al. "Introducing the Graph 500". *Cray Users Group (CUG)* 19 (2010), pp. 45–74.

combined with the 2D SpMV method. This problem differs from the format selection problem because the features that are commonly used for SPMV format selection are not sensitive to the row order of the matrix. Thus, they are not usable when predicting the effects of matrix reorderings since they are identical for all possible orderings of a given matrix. Therefore, we proposed two new order-dependent features based on simplified simulations of the cache: the *group reuse rate* and the *cache reuse rate*. Both are relatively simple and, unlike CNNs, can be computed efficiently even for large graphs. Furthermore, they do not lose information with increasing matrix size, which is a problem with CNNs since they have to scale large matrices to fit their input size.

The first feature is called *group reuse rate*. It attempts to capture spatial locality through a simple, single-line cache model for a straightforward SpMV algorithm. We assume the single available cache line consists of N consecutive elements which are always loaded simultaneously from memory. We assume that matrix nonzeros are accessed in row-major order. For the first nonzero of the matrix, a load is triggered which moves N consecutive vector elements into the cache, starting with the element corresponding to the position of the nonzero. For each subsequent nonzero, if the corresponding vector element is in the cache, a reuse event is triggered and we move on to the next nonzero. Otherwise, a new load event is triggered, as described above. For simplicity, our model of the group reuse rate does not assume that cache lines begin and end at multiples of the cache line size N. Although this is different from how CPU caches operate in reality, the model works well enough in practice.

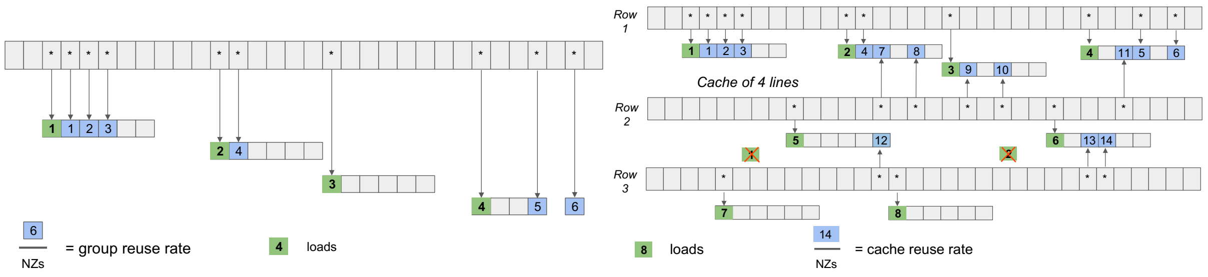


Figure 2 *An example of the group reuse rate (left) and the cache reuse rate (right).*

After all the nonzeros have been accessed, the group reuse rate is obtained by simply dividing the number of reuse events by the number of nonzeros in the matrix. Since each cache line has one load and up to $N-1$ reuse events, the best possible reuse rate is $(N-1)/N$.

We also introduce a second type of feature which we call *cache reuse rate*. It represents a simplified multi-line cache access simulation implemented for a simple SpMV algorithm. As with the *group load and reuse rates*, a cache line consists of N elements, but now there are M cache lines organized in an ordered list. Again, the nonzeros in the matrix are accessed consecutively, and the first element triggers a load event.

For each subsequent nonzero, if the corresponding vector element is currently present in any of the already loaded cache lines, a reuse event is triggered and the cache line that contained the cached element is moved to the top of the cache line access list. If the vector element is not currently in the cache, a load event will be triggered, loading another cache line which is placed on top of the cache line access list. If at least M cache lines have been loaded, the line on the bottom of the access list must be evicted before a new cache line can be loaded. This is part of a single-load event. In this manner, we simulate a simple fully associative least recently used (LRU) policy. Once the simulation is finished, the cache reuse rate is obtained by dividing the number of reuse events by the number of nonzeros in the matrix. An example of both features is shown

Table 1 *Prediction quality of the random forest classifier for medium and large instances. Left: performance metrics. Right: Percentage of maximum performance reached by the different reordering strategies.*

	large	medium
True positives	159	457
True negatives	26	9
False positives	11	95
False negatives	0	2
Accuracy	0.94	0.83
Precision	0.94	0.83
Sensitivity	1	1
Specificity	0.7	0.09
F1 score	0.97	0.9

	Prediction	Always RCM	Never RCM
Large			
Epyc 7763	99.38	96.77	73.23
Epyc 7601	99.27	97.65	77.15
Epyc 7413	99.78	96.07	72.34
Epyc 7302P	99.08	94.19	79.02
Xeon 6130	99.07	94.8	78.78
Xeon 8360Y	99.97	99.28	79.29
Medium			
Epyc 7763	84.42	84.16	74.8
Epyc 7601	97.4	98.19	89.08
Epyc 7413	83.67	82.3	71.98
Epyc 7302P	96.54	97.4	83.5
Xeon 6130	96.87	97.23	79.16
Xeon 8360Y	98.35	98.06	90.93

in Figure 2. By default, we compute this feature with N set to 64 and M to 65536.

Based on the structure-dependent features, we develop a qualitative performance prediction model in order to determine whether a given matrix can be reordered to increase SpMV performance. Thus, we have to solve a classification problem with two classes. For this purpose, we use a standard Random Forest classifier. The Random Forest classifier not only performs well for small datasets, but it can also handle input features of different scales without pre-normalization of the feature values. We train two classifiers, one for large matrices with more than 100 million nonzeroes, and one for medium-sized matrices with 10 to 100 million nonzeroes. There is no need for a classifier for smaller instances since they can run entirely out of cache.

To verify the accuracy of the classifiers, we show the standard classification performance metrics in Table 1 on the left. The classifier for the large matrices shows very good accuracy. However, for the medium-size matrices, the classification performance is lower.

In Table 1, right side, we show the effect of applying the predictions. We weigh every matrix by the number of nonzeroes in order to reflect that mispredictions on larger matrices are more costly. Clearly, RCM is beneficial for most, but not all matrices, and the classifier correctly predicts that in almost all cases. Furthermore, applying RCM when it is not needed is very costly. While we used a slow sequential code, based on the fastest parallel implementation of RCM,[7] reordering takes roughly as much time as 300 SpMV iterations. About 14% of the instances do not benefit from RCM, and in these cases using our system saves this cost.

Thus, we have presented a first classifier that can successfully predict whether applying a reordering is beneficial for SpMV performance, a question that often has a higher impact than the format selection problem which was studied earlier.

2.2 Performance modeling of many-pair point-to-point MPI communication

For many realistic applications of sparse computation, the large computational sizes require using distributed-memory parallel computers, such as clusters of multi-socket, multicore CPUs. When the data connectivity/dependency is irregular and sparse, the typical work/data partitioning

[7] Ariful Azad et al. "The reverse Cuthill-McKee algorithm in distributed-memory". *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2017, pp. 22–31.

result is that both the number of neighbors per partition and the amounts of data exchange needed per pair of neighbors are heterogeneous. In addition, the network topology of a typical cluster is also heterogeneous, consisting of inter-node, inter-socket, and intra-socket levels.

It is thus challenging to accurately model the MPI communication overhead that will incur for the above scenario of distributed-memory parallel computing, due to both the heterogeneities within the network and the potentially heterogeneous contention for the available network bandwidths between many pairs of point-to-point MPI communication commands. The current state-of-the-art *max-rate* model[8][9][10] has, to a certain extent, considered the heterogeneous contention; however, the modeling accuracy suffers in situations where the level of contention changes dynamically. We have therefore developed new performance models for many-pair, point-to-point MPI communication. While all the new models and their details can be found in a newly published journal paper,[11] one of the new models will be presented below.

2.2.1 A general scenario of many-pair point-to-point MPI communication

We target a general situation of P MPI processes each identified with a unique rank. Process with rank i will simultaneously receive M_i^{in} messages from M_i^{in} different neighbors. Each process thus has a list of inward neighbor identities $\{neigh_j^{in}\}_{j=0}^{M_i^{in}-1}$. The neighbors can be of mixed types, i.e., some may reside on the same socket as process i, others may reside on a different socket but inside the same compute node, whereas the rest may reside on other compute nodes. Each process is also assumed to send M_i^{out} outgoing messages to M_i^{out} neighbors (can be different from the M_i^{in} inward neighbors). All the messages can be of varying sizes, and the sizes of the inward and outward messages do not have to match. Thus, we have for each process two sets of message sizes $\{Rsize_j\}_{j=0}^{M_i^{in}-1}$ and $\{Ssize_j\}_{j=0}^{M_i^{out}-1}$. All the point-to-point communications are initiated by `MPI_Isend`/`MPI_Irecv` and concluded by `MPI_Wait`.

2.2.2 A staircase modeling strategy

We will adopt the following modeling strategy:

1. The time needed by process i to complete all its communication tasks is determined by either how soon it can finish receiving all its M_i^{in} incoming messages, or how soon all its M_i^{out} outgoing messages are received at the destinations. The maximum of these two time points will apply. The reason for also considering the delivery of the outgoing messages at the destinations is motivated by the most common situation that uses the rendezvous protocol without intermediate buffering, where experiments show that a sending process cannot be completed until all its outgoing messages are delivered.

2. The "expected" order of which the M_i^{in} incoming messages are received by process i is determined by the sizes of these messages. (The actual order is stochastic, thus not mod-

[8]William Gropp, Luke N. Olson, and Philipp Samfass. "Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test". *Proceedings of the 23rd European MPI Users' Group Meeting*. 2016, pp. 41–50. DOI: 10.1145/2966884.2966919.

[9]Amanda Bienz, William D. Gropp, and Luke N. Olson. "Improving Performance Models for Irregular Point-to-Point Communication". *Proceedings of the 25th European MPI Users' Group Meeting*. 2018, pp. 1–8. DOI: 10.1145/3236367.3236368.

[10]Amanda Bienz, William D Gropp, and Luke N Olson. "Reducing communication in algebraic multigrid with multi-step node awar e communication". *The International Journal of High Performance Computing Applications* 34.5 (2020), pp. 547–561.

[11]Andreas Thune et al. "Detailed Modeling of Heterogeneous and Contention-Constrained Point-to-Point MPI Communication". *IEEE Transactions on Parallel and Distributed Systems* 34.5 (2023), pp. 1580–1593. DOI: 10.1109/TPDS.2023.3253881.

elable.) The smallest incoming message completes first, followed by the second smallest message, and so on. The completion time points for the different incoming messages can be estimated using a *staircase* strategy, which will be detailed by formula (3). If the M_i^{in} incoming messages are of the same size, they are considered to complete simultaneously, due to fairness.

3. When multiple receiving processes, each with one or more incoming messages, compete for the same network connection, another staircase principle applies. The process with the least amount of incoming communication will complete first, letting the remaining processes continue with their remaining communication. This procedure repeats itself until all the processes are completed. At all times, the bandwidth is shared evenly between the still active processes, while the actually available bandwidth can depend on the number of concurrently competing processes.

2.2.3 Model for one-level, multi-neighbor, non-uniform message size

The MPI processes are first organized into groups, where the processes within a group share the same communication bandwidth, i.e., the processes that reside on the same socket for the level of intra-socket communication, or the processes on one socket that share the same inter-socket bandwidth, or the processes that reside in one compute node that share the same inter-node bandwidth. We let N denote the number of processes in a group. The bandwidth under contention is characterized by a set of pre-tabulated aggregate bandwidth values of $BW(1)$, $BW(2)$,..., $BW(N)$, as a function of the competing processes, plus a latency constant τ.

The processes of each group are then sorted with respect to the per-process inward message volume $V_i = \sum_{j=0}^{M_i^{\text{in}}-1} \text{Rsize}_{i,j}$. Thereafter, the MPI time estimate per process T_i is given by the following model:

$$
\begin{aligned}
t_0^{\text{recv}} &= \frac{N \cdot V_0}{BW(N)}, \\
t_i^{\text{recv}} &= t_{i-1}^{\text{recv}} + \frac{(N-i)\cdot(V_i - V_{i-1})}{BW(N-i)}, \quad i = 1, 2, \ldots, N-1, && (1)\\
T_i &= M_i^{\text{in}} \cdot \tau + \max\left(t_i^{\text{recv}}, t_{i,0}^{\text{send}}, t_{i,1}^{\text{send}}, \ldots, t_{i,M_i^{\text{out}}-1}^{\text{send}}\right). && (2)
\end{aligned}
$$

In (2), $t_{i,j}^{\text{send}}$ denotes the delivery time needed by each outgoing message, and it equals the time needed by destination process (with rank $\text{neigh}_{i,j}^{\text{out}}$) to receive this message. Here, we "theoretically" pinpoint the individual time points at which the different messages are received on each destination process. Take for instance process i. It is the destination for M_i^{in} incoming messages, and the total receiving time (without the latency overhead) has been calculated as t_i^{recv} using (1). The following recursive formula, which is again based on a staircase principle, will be used to pinpoint the individual time points at which the M_i^{in} incoming messages are received (the order is sorted according to $\text{Rsize}_{i,0} \leqslant \text{Rsize}_{i,1} \leqslant \ldots \leqslant \text{Rsize}_{i,M_i^{\text{in}}-1}$):

$$
\begin{aligned}
t_{i,0}^{\text{recv}} &= \frac{M_i^{\text{in}} \cdot \text{Rsize}_{i,0}}{V_i} \cdot t_i^{\text{recv}}, \\
t_{i,j}^{\text{recv}} &= t_{i,j-1}^{\text{recv}} + \frac{(M_i^{\text{in}} - j)\cdot(\text{Rsize}_{i,j} - \text{Rsize}_{i,j-1})}{V_i} \cdot t_i^{\text{recv}}, \quad j = 1, 2, \ldots, M_i^{\text{in}} - 1. && (3)
\end{aligned}
$$

2.2.4 An example

As a concrete example, we have partitioned an unstructured computational mesh into 64 pieces using the Zoltan partitioner. Such unstructured meshes normally lead to sparse computations, thus providing a realistic test of the accuracy of the new MPI performance model described in Section 2.2.3. The number of neighbors varies from process to process, while the size of each MPI message varies considerably. The 64 MPI processes are evenly placed onto two sockets of AMD multicore CPUs, and we have ensured that each MPI process only communicates with other processes lying on the remote socket. This is for testing the particular "one-level" model of Section 2.2.3. We remark that we also have more general models that can treat mixtures of messages that communicate on different levels (intra-socket, inter-socket and inter-node), the details can be found in a newly published journal paper.[12]

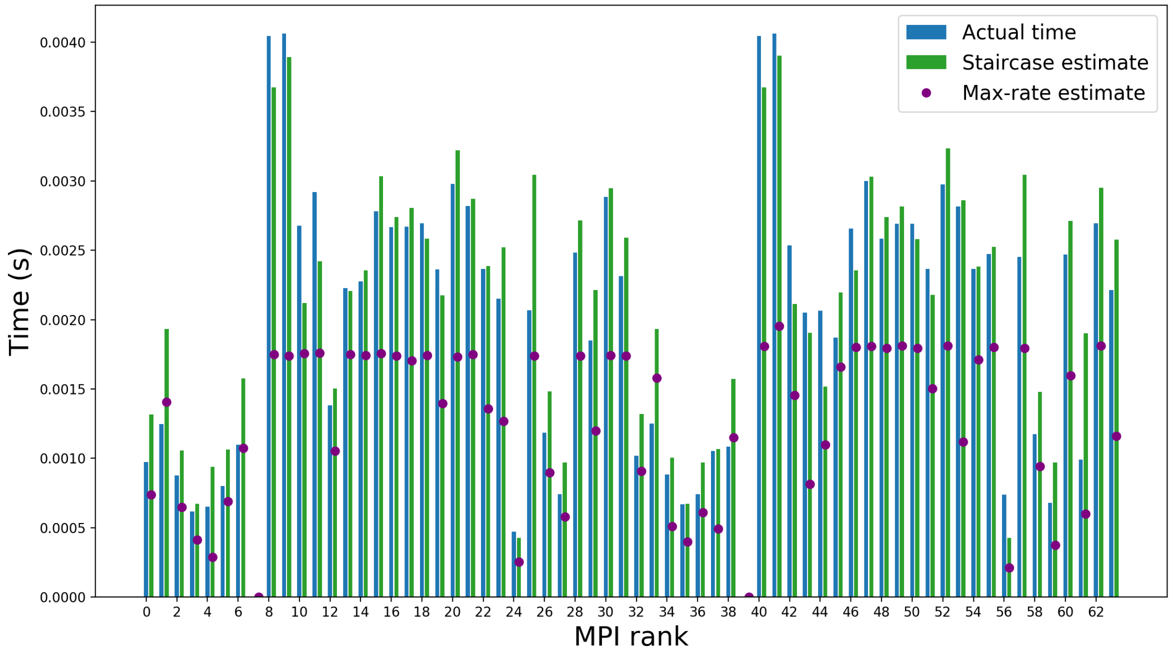


Figure 3 *The predicted and actual time usages per MPI process related to many-pair point-to-point MPI communications.*

Figure 3 shows the per-process time predictions (in green color) produced by our new model, compared with the actual time measurements (in blue color) and the predictions made by the max-rate model[13] (in purple color). It can be observed that our new model is considerably more accurate than the max-rate counterpart.

2.2.5 Future research directions

The new performance models of many-pair point-to-point MPI communication will be used, in combination with the Yloc library (see Section 4.1), to pinpoint the communication overhead related to distributed-memory parallelization of sparse computations. Moreover, hardware- and topology-aware predictions of the communication overhead can be incorporated into the

[12]Thune et al., "Detailed Modeling of Heterogeneous and Contention-Constrained Point-to-Point MPI Communication".

[13]Bienz, Gropp, and Olson, "Reducing communication in algebraic multigrid with multi-step node awar e communication".

next-generation partitioner software (a topic for WP3 of SparCity) to improve the partitioning quality.

3 SPARCITY TOOLS

3.1 MANSARD ROOFLINE MODEL

When modeling the performance upper-bounds, State-of-the-Art (SoA) roofline models[14] may oversimplify the back-end of the micro-architectures by focusing on a subset of functional units and the maximum attainable bandwidth of different memory levels. As such, those models do not consider other hardware components that may limit the performance in any Out-of-Order (OoO) processor, especially the ones related to the retirement of instructions, such as the number of Retirement Slots (RS), Reorder Buffer (ROB) and Physical Register File (PRF). Instead, these models evaluate the performance upper bounds by only considering the isolated performance limits of the different hardware resources, thus giving the illusion of infinite retirement and OoO windows. Moreover, when concurrently executing non-memory and memory instructions, the limited capacities of ROB and PRF can constraint the number of in-flight memory requests, preventing applications from achieving maximum memory bandwidth and increasing the impact of memory latency, especially when accessing the "slower" memory levels. This can hinder the ability of roofline models to provide an accurate characterization of applications that suffer from latency issues. This is the case of several sparse kernels, that due to their irregular memory patterns are highly likely to experience Last Level Cache (LLC) and DRAM latency-related bottlenecks.

To address the main drawbacks of the SoA roofline models, the Mansard Roofline Model (MaRM)[15] considers all the instructions retired by an application, representing performance as Instructions Retired per Cycle (IPC). Since MaRM uses the instruction domain instead of the operations domain, its Arithmetic Intensity (AI) differs from standard roofline models. As illustrated in Figure 4, the AI in MaRM is defined as the number of non-memory instructions (I_{NM}) over the number of memory instructions (I_M). Thus, the ridge point of memory level 'y' (R^y), *i.e.*, the point where memory transfers and computations are completely overlapped in time, is represented as the performance of the non-memory instructions (IPC_{NM}) over the maximum sustainable bandwidth of memory level 'y' ($IPC^y_{M,Max}$), where $y \in \{L1, L2, \ldots, LLC, DRAM\}$.

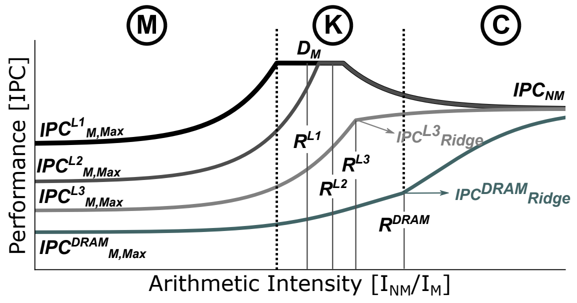


Figure 4 *Illustration of MaRM.*

[14]Samuel Williams, Andrew Waterman, and David Patterson. "Roofline: An Insightful Visual Performance Model for Multicore Architectures". *Commun. ACM* 52.4 (2009), pp. 65–76. ISSN: 0001-0782. DOI: 10.1145/1498765.1498785; Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. "Cache-aware Roofline model: Upgrading the loft". *IEEE Computer Architecture Letters* 13.1 (2013), pp. 21–24.

[15]Diogo Marques, Aleksandar Ilic, and Leonel Sousa. "Mansard Roofline Model: Reinforcing the Accuracy of the Roofs". *ACM Trans. Model. Perform. Eval. Comput. Syst.* 6.2 (2021). ISSN: 2376-3639. DOI: 10.1145/3475866. URL: https://doi.org/10.1145/3475866.

As it can be observed in Figure 4, MaRM includes in the memory-bound region the entire memory hierarchy, with each level represented by its maximum sustainable bandwidth. The compute-bound region of the model is limited by the performance of the non-memory instructions. In contrast to SoA roofline models, the roofs in MaRM have a format similar to a "hill", and the memory roofs are no longer diagonal lines. Since MaRM performance cannot surpass the maximum retirement rate of the micro-architecture, flat regions may occur in the model, indicating areas where the application is limited by the number of retirement slots. This is observed for the L1 roof in Figure 4. While in Cache-Aware Roofline Model (CARM) the ridge point corresponds to the minimum AI that allows attaining maximum performance for any memory levels, in MaRM this does not occur for L3 and Dynamic Random Access Memory (DRAM). Due to the high latency of these memory levels, their effective bandwidth depends on the number of concurrent memory requests. Thus, the ridge points of the L3 cache and DRAM do not correspond to the point where maximum performance is achieved when accessing those memory levels. In fact, the performance continues to increase beyond the ridge point due to the growing contribution of the compute instructions, asymptotically approaching the maximum performance of the compute units. On the other hand, for L1 and L2 caches, the micro-architecture is able to attain the maximum sustainable bandwidth for the entire range of AI, until reaching the ridge point. Hence, the ridge point in MaRM inherits the properties of CARM for L1 and L2 caches.

As represented in Figure 4, MaRM contains three main regions: memory region, where application performance is limited by the memory bandwidth of each memory level; compute region, delimited by the maximum retirement rate of the system; and mixed region (K), where the bottlenecks can be either related to the memory accesses or the maximum achievable performance. The bottleneck identification is performed by plotting a vertical line at the application AI. The intersections right above and below the application dot correspond to the main sources of inefficiencies. Depending on the region where the application is located, a set of optimizations can be derived to improve the execution time. In the memory region, the optimization should focus on improving the memory accesses, while in the compute region vectorization methods can be employed to improve application execution. In the mixed region, techniques from both memory and compute regions might be used.

In the mixed region of the roofs correspondent to the "slower" memory levels, applications are expected to be limited by both memory bandwidth and latency. Compared to the SoA roofline models, this property is exclusive to MaRM, and arises from the ROB impact to the bandwidth of the memory subsystem and retirement of instructions. This effect becomes more relevant as the AI approaches the ridge point. Moreover, since MaRM represents performance as IPC, the model is oblivious to the vector width. For this reason, certain optimizations lead to lower execution time with reduced IPC.

3.1.1 Sparse Matrix-Vector (SpMV) Characterization

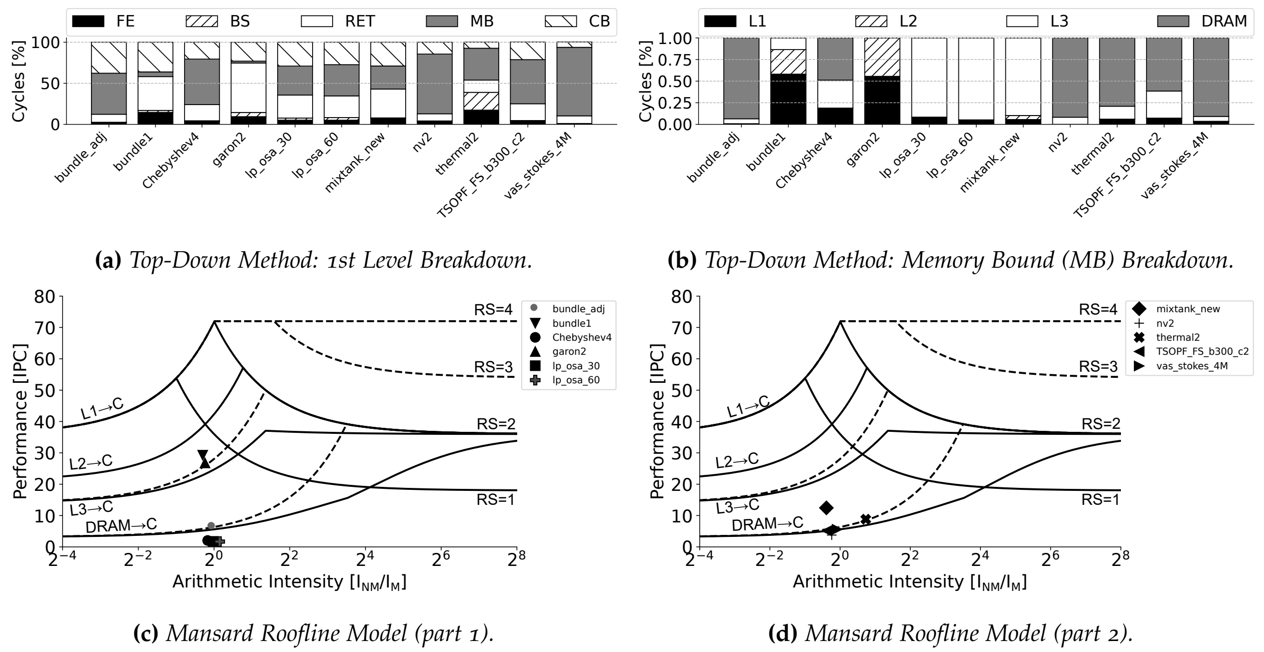
To evaluate the capability of MaRM to provide accurate characterization of sparse kernels, we relied on Intel MKL[16] implementation of the most commonly used sparse operations. For brevity, we focus herein on the sparse matrix-vector (SpMV) multiplication analysis, while the detailed characterization of the sparse matrix-matrix (SpMM) kernel is provided in Deliverable 1.2. For this analysis, a set of SuiteSparse[17] matrices were considered, which cover a wide range of number of non-zeros, rows and columns. The experimental evaluation was conducted on a Intel Xeon

[16]Endong Wang et al. "Intel math kernel library". *High-Performance Computing on the Intel® Xeon Phi™*. Springer, 2014, pp. 167–188.

[17]Kolodziej et al., "The suitesparse matrix collection website interface".

6140 Gold (SKL-X), with 18 cores.

The Top-Down results for MKL SpMV, obtained from Intel VTune,[18] are presented in Figures 5a and 5b. As can be observed from the first level of Top-Down (Figure 5a), all the matrices have significant contributions from retiring, core bound, and memory bound. Since their memory-bound component is almost negligible, the `bundle_1` and `garon2` matrices are expected to be limited by hardware components closer to the core, *e.g.*, private caches. For the remaining matrices that have a significant impact from the memory-bound component, it is also essential to evaluate the memory-bound breakdown provided by Top-Down (Figure 5b), indicating which memory level is the main bottleneck. From the memory breakdown, it is possible to conclude that several matrices are mainly limited by DRAM (`bundle_adj`, `nv2`, `thermal2`, and `vas_stokes_4M`), while `lp_osa_30`, `lp_osa_60` and `mixtank_new` are limited by L3 cache. `Chebyshev4` and `TSOPF_FS_b300_c2` have bottlenecks in several memory levels, mainly from L1, L3 and DRAM.



(a) *Top-Down Method: 1st Level Breakdown.*

(b) *Top-Down Method: Memory Bound (MB) Breakdown.*

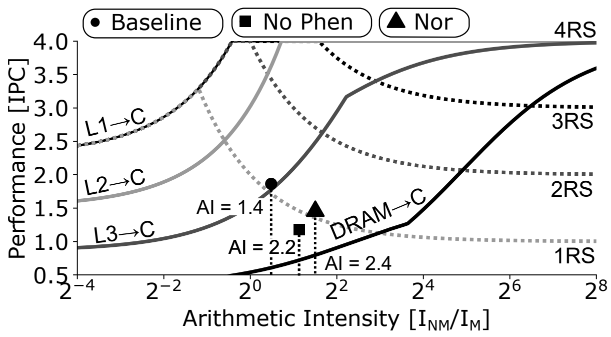
(c) *Mansard Roofline Model (part 1).*

(d) *Mansard Roofline Model (part 2).*

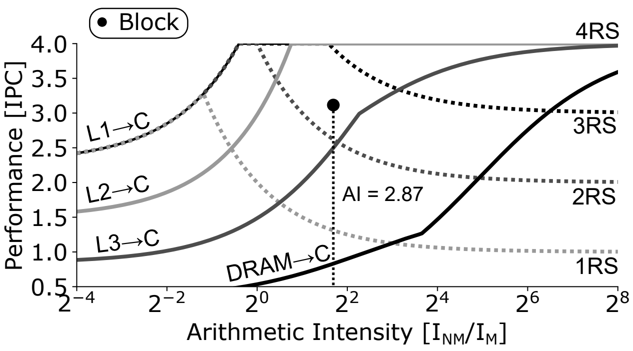
Figure 5 *MKL SpMV in Mansard and Cache-Aware Roofline Models.*

The characterization of these matrices in MaRM is presented in Figures 5c and 5d. It is possible to observe that in some scenarios the characterization provided in MaRM matches the one obtained from the Top-Down method. This is the case of `bundle_adj` and `thermal2`, which are placed above DRAM in MaRM, indicating bottlenecks related to DRAM but also with contributions from components closer to the core that allow surpassing DRAM roof. The `bundle1` and `garon2` matrices are placed in MaRM between L2 and L3 caches, close to the retiring roof of $RS = 1$, which hints that these matrices can be limited by retiring. Moreover, `vas_stokes_4M` and `TSOPF` are placed on top of the DRAM roof in MaRM, which is expected given the DRAM bound nature indicated by the Top-Down method.

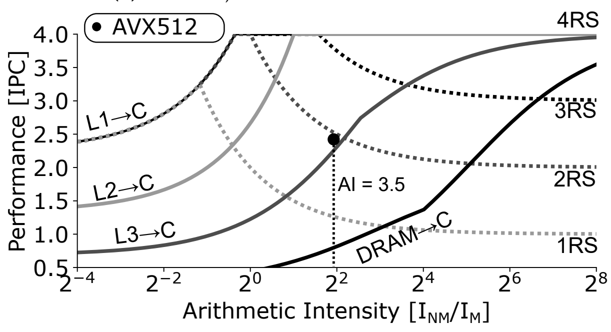
[18] Ahmad Yasin. "A top-down method for performance analysis and counters architecture". *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. 2014, pp. 35–44; Intel Corporation. *VTune Profiler*. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html. [Online; visited June-2022].



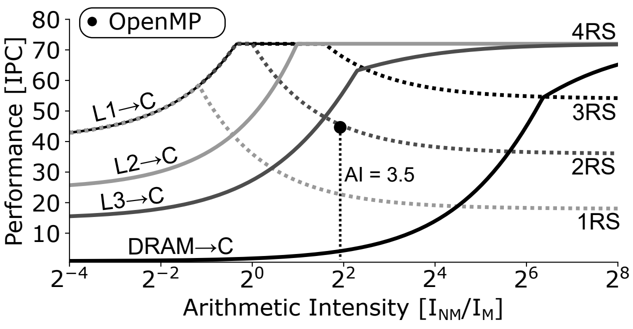
(a) *Baseline, No Phen and Nor versions.*



(b) *Block version.*



(c) *AVX512 version.*



(d) *OpenMP version.*

Figure 6 *Characterization of the Epistasis Detection algorithm in MaRM.*

3.1.2 Case study: second-order epistasis detection

To showcase the ability of MaRM to provide accurate insights when optimizing applications, an epistasis detection algorithm is optimized by following the hints provided by MaRM. It is worth noting that epistasis detection represents one of the core use-case applications in the SparCity project. This algorithm is widely used in bioinformatics to uncover the Single-Nucleotide Polymorphism (SNP) combination that is most likely to cause a disease or trait in a given dataset. The baseline algorithm contains a set of bitwise operations and population count (*popcount*) instructions,[19] which are not commonly covered in the SoA roofline models. The input dataset of this algorithm is a matrix organized with the SNPs in rows and patients in columns. Each SNP is represented by three binary arrays, that express the genotypes. A phenotype is also associated with each patient, indicating if the patient has the disease (case) or does not have the disease (control). In this work, the dataset contains **10040 SNPs and 104448 patients**, *i.e.*, more than 50 million pairwise combinations of SNPs need to be evaluated.

The MaRM characterization of the different optimization steps applied to the epistasis detection algorithm is presented in Figure 6. The baseline algorithm (see Figure 6a) is mainly limited by the L3 cache and placed in the mixed region of the MaRM, *i.e.*, close to the compute roof of 1RS. Since it is memory bound and mainly limited by L3 cache, the user must focus on memory-related optimizations, such as, improving the memory access pattern or reducing the number of memory accesses, which is the goal of *No Phen* and *Nor* optimizations presented in Figure 6a. In the first optimization (No Phen), the phenotype is discarded by separating the dataset into cases and controls. The second optimization (Nor) only uses genotypes 0 and 1 for each patient, while genotype 2 is obtained by applying `nor` operation over the two remaining genotypes. Both these

[19]Ricardo Nobre et al. "Exploring the Binary Precision Capabilities of Tensor Cores for Epistasis Detection". *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2020, pp. 338–347.

optimizations allow for reducing the number of memory accesses.

As can be observed in Figure 6a, these optimization techniques resulted in an increase in the arithmetic intensity of the application, through the reduction of the memory instructions performed. However, it is also possible to verify that the IPC of these two versions is lower than the baseline, although the execution time is reduced 1.42x for the *No Phen* version and 1.72x for the *Nor* version, in comparison to the Baseline application. This effect occurs due to the reduction of the total instructions performed by the applications. Compared to the baseline algorithm, the retired instructions for *No Phen* version reduced 2.27x, while for *Nor* version this reduction was around 2x. Since the decrease in instructions is higher than the time/cycles improvement, the applications attain a lower IPC and become more memory bound. Moreover, MaRM is also able to hint the DRAM latency issues for these two application versions; the application is placed in the region around the DRAM ridge point, where the memory bandwidth is only a fraction of the maximum sustainable bandwidth of the micro-architecture.

Since the *Nor* application has a low retirement rate and is placed below the L3 cache roof, to boost application execution it is necessary to further improve the memory accesses. This task is performed by introducing cache blocking techniques, improving the memory access pattern, and resulting in the *Block* version of the application, which attained a speedup around 3.6x when compared to the *Baseline* version. As it can be observed in Figure 6b, in MaRM, the *Block* version is placed between the compute roofs 2RS and 3RS, hinting its compute-bound nature.

The *Block* version of the application only contains scalar instructions and it is limited in the MaRM by the higher retirement roofs. Thus, it is recommended to vectorize the application, which is performed through the utilization of AVX512 intrinsics, allowing it to attain a time speedup of 5.1x compared to the *Baseline* version. MaRM (Figure 6c) places the *AVX512* version on top of the 2RS roof, indicating that it is completely limited by the retirement units. It is important to notice that MaRM is able to accurately characterize this kernel due to its modeling approach that considers all the instructions retired by the application. It is also possible to observe a drop in the IPC between *Block* and *AVX512* versions. This effect results from the limited number of ports that support AVX512 instruction in Skylake-SP micro-architecture. Hence, it is possible to conclude that the *AVX512* algorithm is able to attain the maximum retirement of the micro-architecture for AVX512 instructions. Moreover, since the *AVX512* version attained the maximum IPC allowed by AVX512 instructions, MaRM hints that the vectorization attained close to maximum efficiency.

Finally, since the single-threaded algorithm is already vectorized and completely limited by the retiring roofs in MaRM, the application is parallelized by using the OpenMP programming model. This parallel version is limited by the 2RS roof in MaRM (Figure 6d) and attains a speedup of 18.5x compared to the AVX512 version and 94x when compared to the baseline version. The super-linear speedup between the *AVX512* and *OpenMP* versions results from the higher utilization of the L3 cache in the *OpenMP* version.

3.2 COMDETECTIVE AND REUSETRACKER

With the increase of their usage in HPC and other domains, there are also higher needs for accurate and low-overhead profiling tools that can identify performance bottlenecks in multi-threaded applications. One attractive method to achieve accurate and low overhead profiling is to use the precise event sampling facility,[20].[21] This facility makes it possible for performance monitoring

[20]Intel. *Intel Microarchitecture Codename Nehalem Performance Monitoring Unit Programming Guide*. https://software.intel.com/sites/default/files/m/5/2/c/f/1/30320-Nehalem-PMU-Programming-Guide-Core.pdf. 2010.

[21]Paul J. Drongowski. *Instruction-Based Sampling: A New Performance Analysis Technique for AMD Family 10h Processors*. https://pdfs.semanticscholar.org/5219/4b43b8385ce39b2b08ecd409c753e0efafe5.pdf. 2007.

units (PMUs) to precisely attribute the sampled hardware events to the actual instructions that trigger those events. Though there have been several precise event sampling-based profiling techniques proposed for AMD machines,[22],[23],[24],[25],[26],[27],[28],[29],[30],[31] none of them captures inter-thread communication and measures reuse in the context of multi-threaded execution. Considering that data movement is a major factor that prevents parallel applications from reaching their theoretical peak performance,[32] there is an urgent need for a profiling technique that detects inter-thread communications.

To detect inter-thread communications and measure data locality in multi-threaded code with low overheads, we substantially extend ComDetective$_{\text{orig}}$,[33] a tool that captures inter-thread communications in the forms of communication matrices, and ReuseTracker$_{\text{orig}}$,[34] another tool that measures reuse distance in multi-threaded applications, to work in AMD machines. We introduced these tools in our previous works and developed them to interface with Intel PEBS in sampling memory accesses. In this project, we extend them to leverage the Instruction Based Sampling (IBS) facility when running on AMD machines to sample memory loads and stores in detecting communications and measuring reuse distance. We call the new tools ComDetective and ReuseTracker. Considering the wide adoption of AMD processors in HPC systems and data centers, our extension to the profiling tools is very timely, relevant, and helpful to the

[22]Jennifer M. Anderson et al. "Continuous Profiling: Where Have All the Cycles Gone?" *ACM Trans. Comput. Syst.* 15.4 (1997), pp. 357–390. ISSN: 0734-2071. DOI: 10.1145/265924.265925. URL: https://doi.org/10.1145/265924.265925.

[23]Paul J. Drongowski. *An introduction to analysis and optimization with AMD CodeAnalyst™ Performance Analyzer*. Tech. rep. Advanced Micro Devices, Inc., 2008.

[24]Inc. Advanced Micro Devices. *AMD uProf*. Accessed: 2021-05-14.

[25]Collin McCurdy and Jeffrey Vetter. "Memphis: Finding and fixing NUMA-related performance problems on multi-core platforms". *2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*. 2010, pp. 87–96. DOI: 10.1109/ISPASS.2010.5452060.

[26]Renaud Lachaize, Baptiste Lepers, and Vivien Quéma. "MemProf: A Memory Profiler for NUMA Multicore Systems". *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*. Boston, MA: USENIX Association, 2012, p. 5.

[27]Xu Liu and John Mellor-Crummey. "Pinpointing Data Locality Problems Using Data-Centric Analysis". *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. IEEE Computer Society, 2011, pp. 171–180.

[28]Xu Liu and John Mellor-Crummey. "A Data-Centric Profiler for Parallel Programs". *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. Denver, Colorado: Association for Computing Machinery, 2013. DOI: 10.1145/2503210.2503297. URL: https://doi.org/10.1145/2503210.2503297.

[29]Xu Liu and John Mellor-Crummey. "A Tool to Analyze the Performance of Multithreaded Programs on NUMA Architectures". *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 259–272. DOI: 10.1145/2555243.2555271. URL: https://doi.org/10.1145/2555243.2555271.

[30]Xu Liu and Bo Wu. "ScaAnalyzer: a tool to identify memory scalability bottlenecks in parallel programs". *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2015, pp. 1–12. DOI: 10.1145/2807591.2807648.

[31]Probir Roy and Xu Liu. "StructSlim: A Lightweight Profiler to Guide Structure Splitting". *Proceedings of the 2016 International Symposium on Code Generation and Optimization*. Barcelona, Spain: Association for Computing Machinery, 2016, pp. 36–46. DOI: 10.1145/2854038.2854053. URL: https://doi.org/10.1145/2854038.2854053.

[32]D. Unat et al. "Trends in Data Locality Abstractions for HPC Systems". *IEEE Transactions on Parallel and Distributed Systems* 28.10 (2017), pp. 3007–3020.

[33]Muhammad Aditya Sasongko et al. "ComDetective: A Lightweight Communication Detection Tool for Threads". *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Denver, Colorado: Association for Computing Machinery, 2019. DOI: 10.1145/3295500.3356214. URL: https://doi.org/10.1145/3295500.3356214.

[34]Muhammad Aditya Sasongko et al. "ReuseTracker: Fast Yet Accurate Multicore Reuse Distance Analyzer". *ACM Trans. Archit. Code Optim.* 19.1 (2021). ISSN: 1544-3566. DOI: 10.1145/3484199. URL: https://doi.org/10.1145/3484199.

application programmers.

Even though the original communication detection and reuse distance algorithms remain unchanged, extending the Intel-based profiling tools to the AMD multicores is not a straightforward task and comes with many challenges. AMD IBS is very different from Intel PEBS in hardware design as it can be programmed to count and sample only instruction fetches or executed micro-operations. To target only specific events such as memory accesses for profiling, software-level filtering is needed to choose only memory accesses among all micro-operation samples to be taken as inputs for the profiling tool. Furthermore, AMD IBS requires certain BIOS software that is not widely available,[35] especially in the cloud machines. As an alternative solution, we relied on a Linux kernel module[36] that allows us to program IBS hardware. Originally this kernel module is designed to count hardware samples and produce a log of samples. As the kernel module has a simplistic workflow, we extensively modified the kernel module in order to introduce additional functionalities necessary for supporting our profiling tools.

To demonstrate the usage of ComDetective and ReuseTracker, we profiled and code-refactored several benchmarks. By leveraging these tools, we also show their impacts and usefulness in improving data locality and thus application performance. Using ComDetective and ReuseTracker, we profiled the data mining benchmarks from the MineBench benchmark suite.[37] Among these benchmarks, we detected false sharing using ComDetective, which also corresponds to the spatial reuses in L3 caches as detected by ReuseTracker, in *kmeans*, *ScalParC*, *SVM-RFE*, and *Utility_Mining*. Since the false sharing in *kmeans*, *ScalParC*, and *SVM-RFE* occurs between elements of large arrays, we could not remove the false sharing by adding padding between adjacent elements as it will bloat the memory consumption. For *Utility_Mining*, the false sharing happens between attributes of a struct data structure, and therefore, we could add padding between them without bloating the memory consumption. Due to this code refactoring, we gained 15% speedup from the modified *Utility_Mining* benchmark.

By using only ReuseTracker, we also identified long-distanced reuses whose distances exceed the size of each local L2 cache in *kmeans*, *PLSA*, *RSEARCH*, *ScalParC*, *SVM-RFE*, and *Utility_Mining*. In order to confirm the impact of this data locality problem on memory access latency, we modified ReuseTracker to also report the cache miss latency of each load sample that misses in the L1 data cache, which is also a piece of information recorded by IBS. In the profiled benchmarks, the use-reuse pairs that cause long-distance reuses mostly happen across different function calls, which makes it difficult to modify the code using simple approaches like loop transformation. However, in the *ScalParC* benchmark, we found out that a huge portion of use-reuse pairs occur in the same function and a huge portion of high latency cache misses also happen in this function. Based on this information, we improved the data locality in this function by replacing each pair of statements that perform increment and assignment operations separately on the same global array element into a single statement. This simple modification yields 29% performance gain from the modified *ScalParC*.

[35] Joseph L. Greathouse. *Re: Error : IBS profiling is disabled in your BIOS*. https://community.amd.com/t5/general-discussions/error-ibs-profiling-is-disabled-in-your-bios/td-p/55043. AMD Community; Joseph L. Greathouse. *Re: IBS not available on EPYC 7451 ?* https://community.amd.com/t5/server-gurus-discussions/ibs-not-available-on-epyc-7451/m-p/258228. AMD Community.

[36] Joseph L. Greathouse. *AMD Research Instruction Based Sampling Toolkit*. https://github.com/jlgreathouse/AMD_IBS_Toolkit. 2017.

[37] Ramanathan Narayanan et al. "MineBench: A Benchmark Suite for Data Mining Workloads". *2006 IEEE International Symposium on Workload Characterization*. 2006, pp. 182–188. doi: 10.1109/IISWC.2006.302743.

3.3 MATRIX AND TENSOR GENERATORS

In order to support ML-based prediction models, we devise a matrix and tensor generator. A smart sparse matrix generator is implemented that takes the significant features of matrices into consideration. Our aim is to mitigate real-world matrices by investigating and using their features. Our generator utilizes a set of given metrics to create a matrix having these features. With our generator, one can create matrices obeying structural or value-dependent restrictions. Structural properties include size, density, coefficient of variation for row lengths, given bandwidth for square matrices, and given profile for symmetric matrices. Value-dependant properties allow users to choose whether the matrix is diagonally dominant and/or positive definite.

Similar to the matrix generator, we propose a smart sparse tensor generator that takes the significant features of tensors into consideration. For this, we investigate real tensors and mitigate them using their features.

Our generator utilizes a set of given metrics to create a tensor having these features. These are (i) sizes, and (ii) density of the tensor; (iii) density of fibers, i.e., the ratio of nonzero fibers over all fibers; (iv) coefficient of variation of nonzero counts per slice; and (v) coefficient of variation of fiber lengths. The reason behind using the coefficient of variation here is to have a better intuition regarding the distribution of nonzeros, independent of the size of the tensor.

For ease of expression, we describe the construction of a 3-mode tensor. First, nz_per_slice array is constructed for mode-0 slices, according to the average and standard deviation for mode-0 slices using the Box-Muller method.[38] Then for each slice i, we construct the nz_per_fiber array for mode-(0,1) fibers as follows. The number of nonzero fibers in slice i is determined proportional to the number of nonzeros in slice i. According to that count, the nonzero fiber indices in slice i are selected randomly. For these nonzero fiber indices, we find their nonzero counts according to the average and standard deviation for fibers again using the Box-Muller method. Finally, we fill the nonzeros within these fibers accordingly, by randomly deciding the nonzero indices.

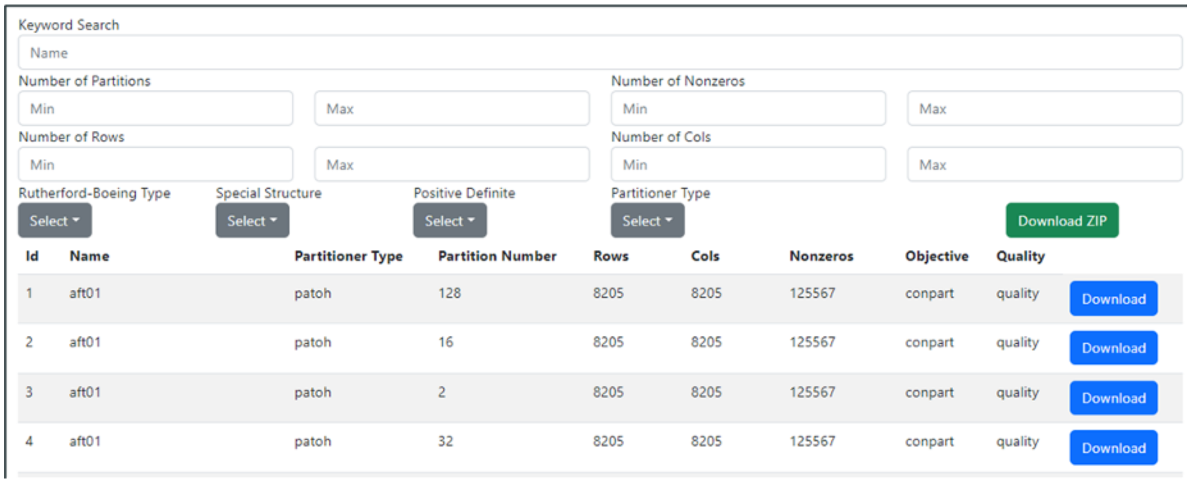
3.4 PARTITIONING UTILITY API

A state-of-the-art (hyper)graph partitioning utility has been developed using METIS and PaToH. This utility was utilized to generate partitioning results for some of the sparse matrices in SparCity's open repository of sparse problem instances. The objective was to standardize performance comparisons and benchmarking for parallel sparse matrix operations and to help researchers save time and computational resources while ensuring the reproducibility of research results. Specifically, all real matrices in the SuiteSparse collection with over 100,000 nonzeros were partitioned, resulting in 638 matrices being partitioned. For each dataset, partitioning vectors were generated for 2, 4, 8, 16, 32, 64, and 128 parts, using the partitioning objectives of minimizing edge-cut and communication volume for METIS, and connectivity and cut-net metrics for PaToH. The partitioning information, such as cutsize, imbalance, and partitioning time usage, was also recorded as a part of the repository meta-data for reference.

The main objective of designing an API for Sparse Matrix Utility is to allow users to filter and view partitions generated by various partitioners. We initially started with METIS and PaToH partitioners to build the web interface but plan to expand it to reordering algorithms for sparse matrices. The web interface designed for this purpose enables users to download individual partitions or collections of selected partitions (as a zip file) after filtering. On the backend side of the web interface, Spring Boot framework of Java is used while on the frontend side ReactJs

[38]ER Golder and JG Settle. "The Box-Müller Method for Generating Pseudo-Random Normal Deviates". *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 25.1 (1976), pp. 12–20.

library of JavaScript is used. In order to filter partitions dynamically, the website utilizes from a relational database that is using Java H2 driver. In this model, a partition is a super entity (class) that can be extended by different sub-entities. In the current model, partitions can be either METIS partitions or PaToH partitions. However, in the future, this model can be extended for different partitioners. Each record in the database has a unique id and path that are used for downloading. The name attribute of the partition represents the name of the matrices that are partitioned. The columns, rows, and non-zeros attributes represent the size and shape of the matrices. The group and kind attributes represent institutions and areas of the matrices that are used. Furthermore, the number of partitions and objective attributes are attributes generated by partitioners.



Id	Name	Partitioner Type	Partition Number	Rows	Cols	Nonzeros	Objective	Quality	
1	aft01	patoh	128	8205	8205	125567	conpart	quality	Download
2	aft01	patoh	16	8205	8205	125567	conpart	quality	Download
3	aft01	patoh	2	8205	8205	125567	conpart	quality	Download
4	aft01	patoh	32	8205	8205	125567	conpart	quality	Download

Figure 7 *Frontend side of the Partitioning Utility Webpage*

On the frontend, the users can filter partitions based on the attributes they have in the database. Additionally, users can filter partitioners based on the partitioner type and some special attributes that belong to a specific partitioner. In general, the website is using the Representational State Transfer (REST) API architecture. Based on the user-entered inputs via the frontend, the corresponding attributes are filtered in the backend side of the webpage and sent back to the frontend side by `http` get requests. Figure 7 demonstrates the user interface of the website that is designed so far. The website is currently running on a local server, but it is going to be deployed to a web server in a few weeks and become publicly available.

3.5 A64FX CACHE PARTITIONING PROFILER

We have developed a profiling tool able to model the effect of cache partitioning in programs. Cache partitioning allows dividing a cache into multiple partitions. The profiling tool uses Intel PIN[39] for dynamic binary instrumentation. It records reuse distance histograms for each of the program's data objects (array or pointers) to estimate the number of cache misses in case a data object was assigned to a separate cache partition.

Fujitsu's A64FX processor is equipped with a way-based hardware cache partitioning mechanism named *sector cache*.[40] It enables partitioning the L1D and L2 caches and assigning a

[39]Chi-Keung Luk et al. "Pin: building customized program analysis tools with dynamic instrumentation". *Acm sigplan notices* 40.6 (2005), pp. 190–200.

[40]*A64FX Microarchitecture Manual*. Version 1.5. Fujitsu Limited. 2021. URL: https://github.com/fujitsu/A64FX/blob/master/doc/.

program's data objects to partitions. The tool is not specifically tailored to the A64FX processor or SpMV programs. However, we can optimize a program and assess the effect of cache partitioning in SpMV on the A64FX using the profiling results.

Listing 1: Cache partitioning profiler usage.

```
# run profiling for <program>:
$ pin -t ./pincpt/obj-intel64/pincpt.so -- <program>
# generates <profling data>

# run post-processing with profiling data:
$ python pincpt.py <profiling data>

# output for example program:
region              ellgemv    # a function in <program>
cache level               2
nways                    12    # partition size
misses sc           1394574    # cache misses w/ cache partitioning
misses nosc         1511264    # cache misses w/o cache partitioning
reduction [%]      7.721351    # cache miss reduction
    ellgemv: isolate object "x" in 12 cache ways of L2 cache
```

Listing 1 shows an example of using the profiling tool with a program. The profiling tool runs the target program and generates profiling data that can be analyzed in a post-processing step. The output of the post-processing is a list of functions that benefit from cache partitioning as well as the suggested partition sizes and data objects to be isolated in a partition separate from the rest of the program data. Additionally, the output includes the estimated number of cache misses with and without using cache partitioning and the corresponding reduction in cache misses.

Listing 2: Fujitsu C compiler cache partitioning compiler directives.

```
#pragma procedure scache_isolate_way     L2=12
#pragma procedure scache_isolate_assign  x
```

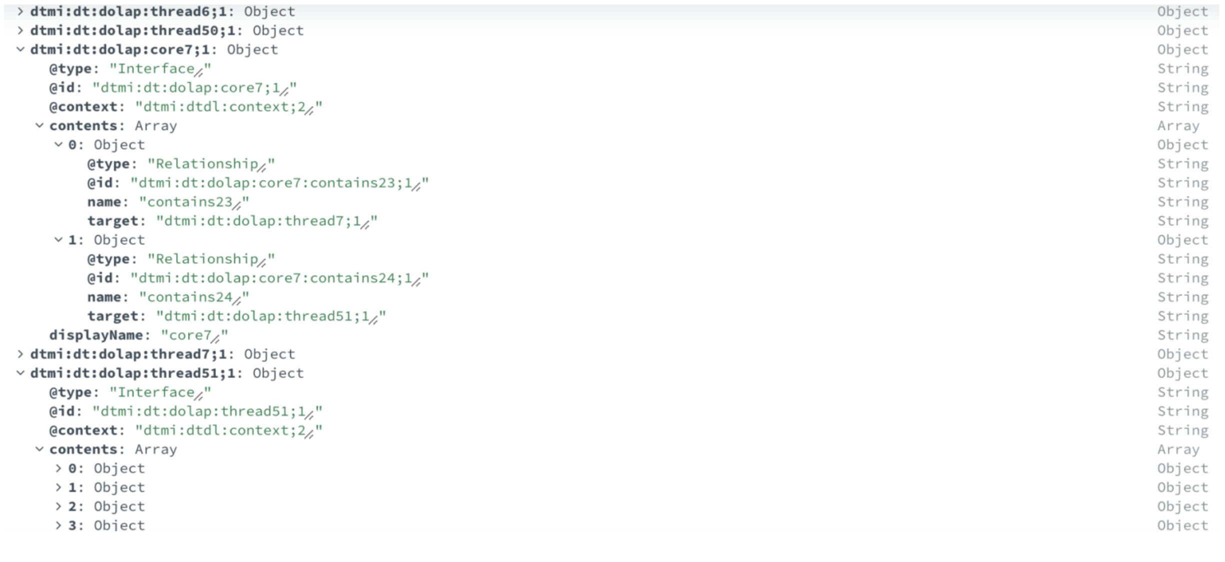
In the end, we can use that output information to apply the cache partitioning. Using Fujitsu's C compiler, this can be implemented by adding the source code lines shown in Listing 2 in the function `ellgemv` within the example program. Applying this optimization resulted in a speedup of 1.18 and a measured cache miss reduction of 7.36 % (estimate: 7.72 %) for the example program execution on the A64FX processor.

3.6 SUPERTWIN

SuperTwin is designed to achieve comprehensive monitoring and profiling by collaboratively using state-of-the-art digital twin description, data extraction, and visualization tools. To this end, an altered version of the Azure Digital Twin Description Language (DTDL) is derived with respect to the requirements of a digital twin of computational environments instead of IoT devices, and SuperTwin Description (STD) is defined. STD encodes complete system information for a target system on its components, topology, and metrics that each component can report and benchmark results. STD also enriches this data with time-series database pointers and profiling records that are used to generate structured queries, dashboards, and linked-data structures. Moreover, to be able to provide a robust interface to STD, a SuperTwin class is implemented whose one instance can probe a target system automatically by using a combination of widely available feature extraction tools such as `cpuid`, `lshw`, `likwid-topology`, and `libpfm4`.

The SuperTwin class manages metadata and time-series data using the mentioned pointers. A SuperTwin object uses MongoDB to register STDs and InfluxDB to store time-series data acquired by sampling. To facilitate sampling from target systems, SuperTwin uses Performance Co-Pilot (PCP), which is developed by RedHat. However, SuperTwin does not simply launch PCP collectors on the target system. Instead, leveraging the information acquired during probing, SuperTwin configures PCP collectors for both software metrics and hardware performance events when desired. Since the STDs are tree-like data structures, homogenous in sub-trees and methods for all target systems, they are easily comparable to each other. SuperTwin class also implements methods to generate linked-data structures via structured queries on the systems that are on the same network or, STDs with attached profiling reports to allow linked-data structures among performance events that take place in systems that operate in different closed networks.

Since STDs are tree-like data structures with well-defined sub-trees, they are perfectly scalable for larger systems. The difference between representing a single compute node and a computing cluster with SuperTwin is that the latter is achievable by only connecting node graphs to a new root which is the root node of cluster twin. After this step, the same methods for node-level twins can be used with more depth in recursion. On top of probing, knowledge representation, automated profiling, and linked-data structures, SuperTwin operates a visualization pipeline that is modular and recursive in compatible with STD. Therefore templated or custom dashboards can be created on-the-fly for each component or computation in STD. Those dashboards are also interlinked and comply with linked-data structures.



```
> dtmi:dt:dolap:thread6;1: Object                                  Object
> dtmi:dt:dolap:thread50;1: Object                                 Object
v dtmi:dt:dolap:core7;1: Object                                    Object
    @type: "Interface"                                             String
    @id: "dtmi:dt:dolap:core7;1"                                   String
    @context: "dtmi:dtdl:context;2"                                String
  v contents: Array                                                Array
    v 0: Object                                                    Object
        @type: "Relationship"                                      String
        @id: "dtmi:dt:dolap:core7:contains23;1"                    String
        name: "contains23"                                         String
        target: "dtmi:dt:dolap:thread7;1"                          String
    v 1: Object                                                    Object
        @type: "Relationship"                                      String
        @id: "dtmi:dt:dolap:core7:contains24;1"                    String
        name: "contains24"                                         String
        target: "dtmi:dt:dolap:thread51;1"                         String
    displayName: "core7"                                           String
> dtmi:dt:dolap:thread7;1: Object                                  Object
v dtmi:dt:dolap:thread51;1: Object                                 Object
    @type: "Interface"                                             String
    @id: "dtmi:dt:dolap:thread51;1"                                String
    @context: "dtmi:dtdl:context;2"                                String
  v contents: Array                                                Array
    > 0: Object                                                    Object
    > 1: Object                                                    Object
    > 2: Object                                                    Object
    > 3: Object                                                    Object
```

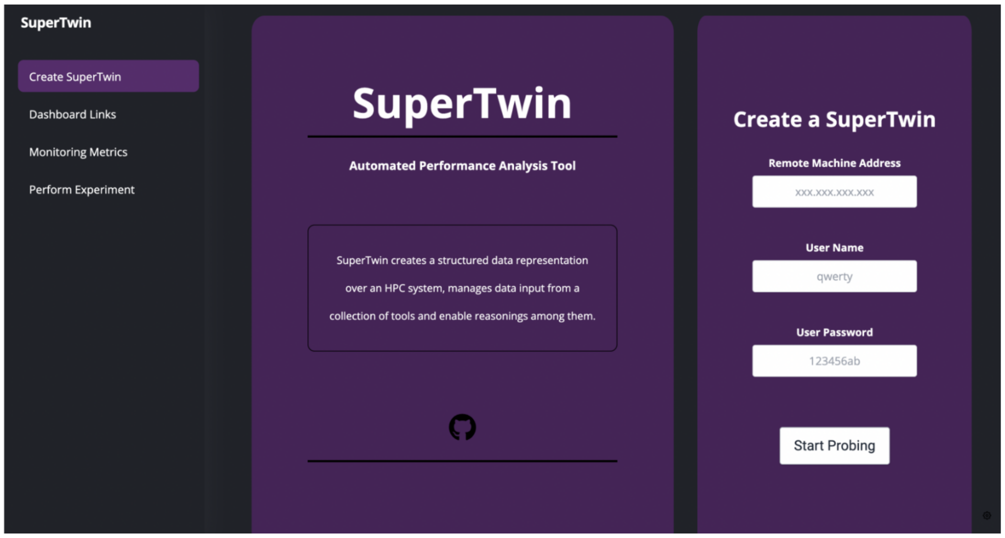
Figure 8 *An STD sample from the document store database. In this sample, how the topology of a system is encoded into STD is shown. The "contains" relations between two components can be combined as a tree of all components that shows the hierarchical structure of the target system. When combined with the database pointers and metric metadata, structured queries, analysis pipelines, and interlinked dashboards are generated.*

SuperTwin samples target systems with two different approaches; one is creating a monitoring framework with low-frequency sampling to provide a general picture of the target system's state.

The other is configuring the PMUs before executing a computation and sampling performance events with high frequency. In this approach, SuperTwin interferes with the process launch, generates shell scripts that launch the desired kernels, sets affinity leveraging previously queried system topology, and records the measurements during execution. For all operations, SuperTwin performs the operations other than sampling on a host system that is (preferably) different from the target system and creates minimal overhead during measurements.

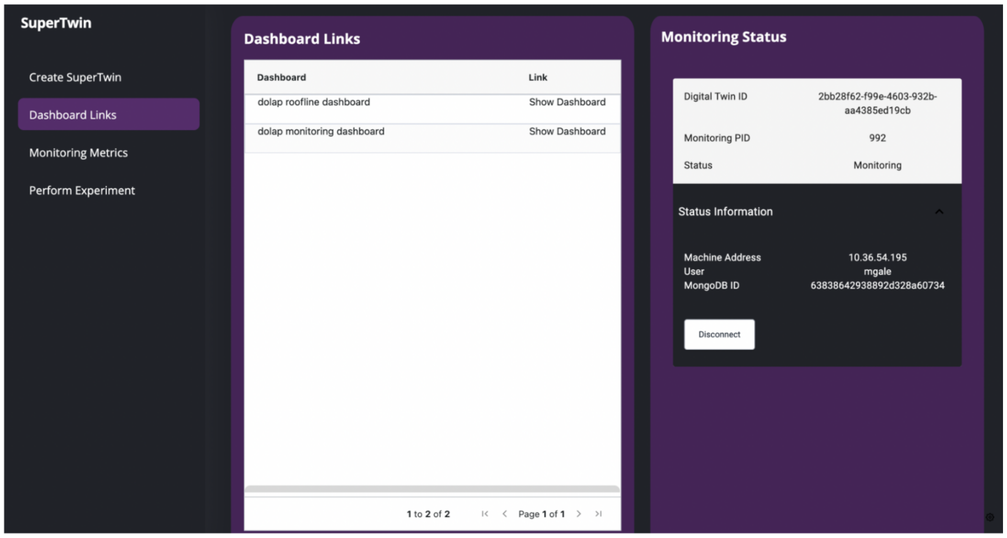
3.6.1 SUPERTWIN WEB

SuperTwin can be run as a console application or via using SuperTwin API. However, since it can extract more than 1000 metrics, it can be overwhelming for users to scan and select the desired metrics to monitor. To address this issue, the tool has a web application that is built with `React.js` and `Flask` that is connected to SuperTwin API to improve the user experience.



SuperTwin creates a structured data representation over an HPC system, manages data input from a collection of tools and enable reasonings among them.

Figure 9 *The login page of the SuperTwin application: when logged in with a user account and an IP address, a probing phase on the target system starts. This probing can also include multi-threaded CARM, STREAM, and HPCG benchmarks that are tailored for the system topology. If the probing phase is done before and STD is generated, login simply re-instantiates SuperTwin using the STD recorded in the database.*


SuperTwin

- Create SuperTwin
- Dashboard Links
- Monitoring Metrics
- Perform Experiment

Dashboard Links

Dashboard	Link
dolap roofline dashboard	Show Dashboard
dolap monitoring dashboard	Show Dashboard

1 to 2 of 2 Page 1 of 1

Monitoring Status

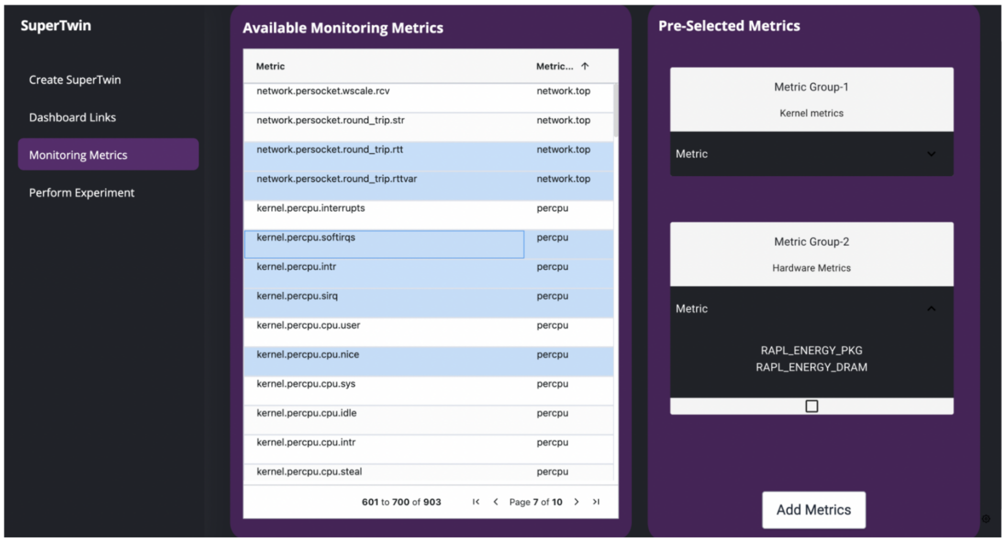
Digital Twin ID	2bb28f62-f99e-4603-932b-aa4385ed19cb
Monitoring PID	992
Status	Monitoring

Status Information

Machine Address	10.36.54.195
User	mgale
MongoDB ID	63838642938892d328a60734

Disconnect

Figure 10 *Monitoring and performance model dashboards are generated once the probing phase is completed. Those dashboards are encoded in STD and can be accessed at any time afterward. The dashboard links page also shows the digital twin metadata and monitoring framework status, and also provides the capability to disconnect or restart monitoring agents. If more events are observed and more dashboards are generated, those dashboards are also accessible from this page.*


SuperTwin

- Create SuperTwin
- Dashboard Links
- Monitoring Metrics
- Perform Experiment

Available Monitoring Metrics

Metric	Metric...
network.persocket.wscale.rcv	network.top
network.persocket.round_trip.str	network.top
network.persocket.round_trip.rtt	network.top
network.persocket.round_trip.rttvar	network.top
kernel.percpu.interrupts	percpu
kernel.percpu.softirqs	percpu
kernel.percpu.intr	percpu
kernel.percpu.sirq	percpu
kernel.percpu.cpu.user	percpu
kernel.percpu.cpu.nice	percpu
kernel.percpu.cpu.sys	percpu
kernel.percpu.cpu.idle	percpu
kernel.percpu.cpu.intr	percpu
kernel.percpu.cpu.steal	percpu

601 to 700 of 903 Page 7 of 10

Pre-Selected Metrics

Metric Group-1

Kernel metrics

Metric

Metric Group-2

Hardware Metrics

Metric

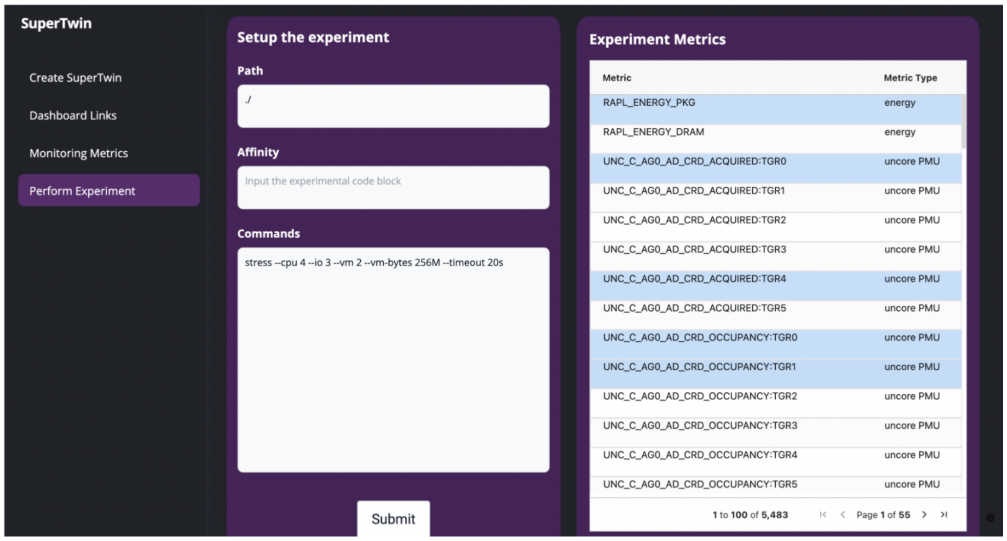
RAPL_ENERGY_PKG
RAPL_ENERGY_DRAM

Add Metrics

Figure 11 *The monitoring framework is reconfigurable via SuperTwin API and/or SuperTwin application. On the configuration page, the recommendations for pre-determined metric groups and a complete list of (mostly software) metrics that have meaningful information on the system state are provided to the user. A metric list is pre-processed with respect to their components and categories and search functionality is provided. Changes made on sampled metrics from this page have an immediate effect on the monitoring dashboard that is listed in Figure 10.*

3.6.2 CONCLUSION

SuperTwin, at the moment, is available for generating automated performance models using CARM and provides baselines with STREAM and HPCG benchmarks. The research on the effect of affinity on the benchmark results is still going on. SuperTwin is also capable of generating monitoring dashboards for single and multiple socket systems and dashboards for execution profilings using metrics collected from PMUs. Dashboards that provide a comparison between different executions on the same host, or on different hosts are also ready to use. Those automated probing, monitoring, profiling, and knowledge retrieval capabilities can be used via SuperTwin class that is implemented in Python. In addition, the web application can be used to easily scan and filter the target system's measurement capabilities. SuperTwin is designed to provide completely interlinked dashboards that are generated automatically for every component of a computational system. This capability is once implemented in the previous versions, however, it is now removed from the source code and under work for major improvement. At the moment, SuperTwin is capable of constructing single-node twins. However, as mentioned previously, (heterogeneous) compute clusters are the main target of SuperTwin and it is designed mainly for this purpose. The current aim is to further enrich the probed data and knowledge graph generation capabilities with the integration of `yloc` into SuperTwin. We are also working on building a visualization pipeline on top of this structure while scaling SuperTwin for clusters. When this integration is complete, a completely modular and interlinked visualization helper class will be released as part of SuperTwin.



Path: ./

Affinity: Input the experimental code block

Commands:

```
stress --cpu 4 --io 3 --vm 2 --vm-bytes 256M --timeout 20s
```

Metric	Metric Type
RAPL_ENERGY_PKG	energy
RAPL_ENERGY_DRAM	energy
UNC_C_AG0_AD_CRD_ACQUIRED:TGR0	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR1	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR2	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR3	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR4	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR5	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR0	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR1	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR2	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR3	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR4	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR5	uncore PMU

1 to 100 of 5,483 |< < Page 1 of 55 > >|

Figure 12 *The Observation framework is also reconfigurable using SuperTwin API and/or SuperTwin application. During the probing phase, Model Specific Register (MSR) and Performance Monitoring Units (PMUs) are queried and the performance events they can measure are encoded within the STD. This information is later provided to the user to select events of interest when launching a computation. Similarly to monitoring metrics, hardware events are pre-processed and a search functionality over categories/names is provided to the user. When an executable on the target system wanted to be launched, SuperTwin asks for its path, parameter, and CPU affinity. CPU affinity can be indicated of raw strings using `likwid-pin` or `numactl`, or can be generated by SuperTwin to user needs. For example, when "numa compact 4" is stated, it means that two threads on each NUMA node that use the same L1 cache are launched. When "balanced 4" is given, it means that four threads that do not share the L1 cache are launched without considering the NUMA topology. Upon generating the affinity strings, the system topology acquired during the probing phase is leveraged. SuperTwin then resolves those strings and monitors performance events from requested hardware threads. When the execution is complete, the generated dashboard is available on the Dashboard Links page.*

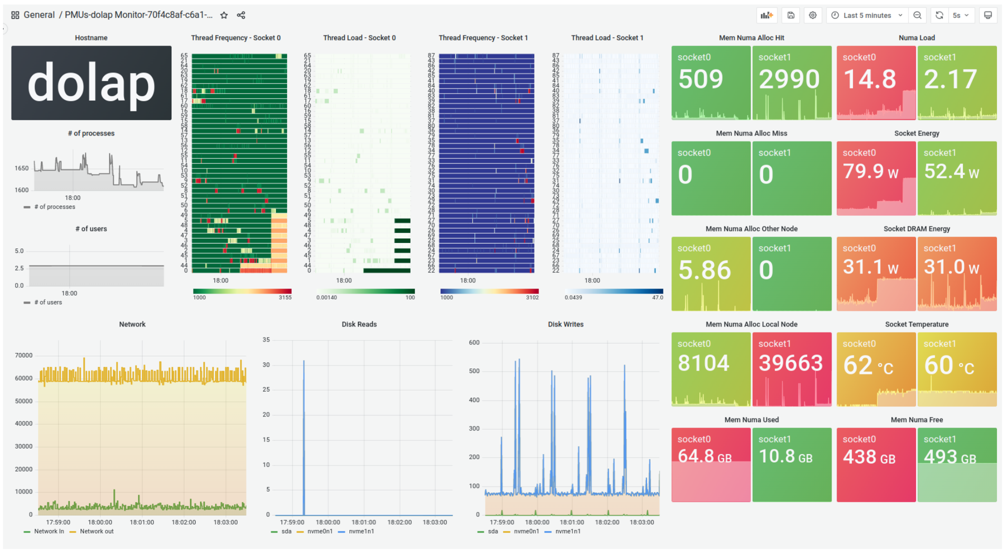


Figure 13 *The Monitor dashboard for a single-node host Dolap. In the socket panels, threads sharing the same L1 cache are plotted consecutively, leveraging STD. At the time of the screenshot, a computation that is just launched in NUMA socket 0 but anomalously allocates memory from NUMA socket 1. This dashboard is accessible via link in Figure 10 and configurable by choosing metrics as in Figure 11.*

4 SPARCITY LIBRARIES

4.1 YLOC

Yloc is a C++ shared library that represents a compute system topology in a graph. It is built on top of the Boost Graph Library (BGL)[41] and uses several modules, for example, external libraries such as Hwloc,[42] as topology information sources. Hardware components are represented as graph vertices whereas their topological relationship is represented with graph edges in Yloc. For example, two directly interconnected GPUs are connected by an edge between the two vertices representing the GPUs in the graph. In contrast, when GPUs are interconnected only via a PCIe bus, the shortest path between those GPUs will be via edges to PCIe vertices. Yloc makes it possible to query properties of system components, for example, the current load of a GPU, or to run graph algorithms, such as finding the shortest path between two vertices.

4.1.1 USING YLOC – AN EXAMPLE WITH GPUS

Listing 3 shows an example application using Yloc to find the GPUs in a node of the system and query the current GPU load. First, the topology graph is generated with `yloc::init`. The topology graph is accessible via `yloc::graph` and is a BGL graph. The graph can be used with the BGL algorithms. To find GPU vertices in the topology graph, we can filter the graph with `boost::make_filtered_graph` for vertices that are of type `yloc::GPU`. Vertex properties can be accessed with `graph[vertex].get<T>(name)`, where `name` is the name of the property and `T` its return type. Because hardware components have a heterogeneous set of properties, the get-function returns a `std::optional` object with a possibly empty value. The example code in Listing 3 is portable across different GPU vendor architectures. AMD and NVIDIA GPUs are

[41] Boost. *Boost C++ Libraries*. http://www.boost.org/.

[42] François Broquedis et al. "hwloc: A generic framework for managing hardware affinities in HPC applications". *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE. 2010, pp. 180–186.

currently supported in Yloc.

Listing 3: Using Yloc to query a (dynamic) property of GPUs.

```
int main(int argc, char *argv[])
{
    yloc::init();
    yloc::Graph &graph = yloc::graph();
    // filter GPUs:
    auto filtered_graph = boost::make_filtered_graph(
        graph,
        boost::keep_all{}, // keep all edges, filter GPU vertices
        std::function{[&](yloc::vertex_descriptor_t v) {
            return graph[v].type->is_a<yloc::GPU>();
    }});
    // query property 'load' of the GPUs:
    for (auto gpu_vertex : boost::make_iterator_range(
            boost::vertices(filtered_graph))) {
        uint64_t load = graph[gpu_vertex].get<uint64_t>("load").value();
    }
    yloc::finalize();
}
```

4.1.2 Using Yloc – An Example with MPI

Another use case for Yloc is finding distances between (remote) MPI processes in a system topology. Listing 4 shows an example application to generate a distance matrix between MPI processes in a distributed MPI program. The distance matrix is transparently generated with the function `yloc::distance_matrix` which uses the BGL graph algorithms internally (breadth first search). The required number of hops from one MPI process to another is used as a distance metric (`boost::on_tree_edge`. However, in principle another metric calculated from a combination of edge and vertex properties, such as latency and bandwidth, could be used instead.

The reported distance depends on the locality of MPI processes. For example, the distance between two MPI processes on the same compute node with affinity to the same NUMA domain is lower than the distance between two MPI processes with affinity to different NUMA domains. The distance to MPI processes on remote nodes depends on the system interconnect topology, but is always higher than the distance to local MPI processes. Such topology information is useful in optimizing communication algorithms in distributed systems and will be used e.g. in an ongoing development of a task migration algorithm developed at LMU.

Listing 4: Using Yloc to generate a distance matrix between MPI processes in a distributed MPI program.

```
int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);
    yloc::init();

    yloc::Graph &graph = yloc::graph();
    auto filtered_graph = boost::make_filtered_graph(
        graph,
        boost::keep_all{}, // keep all edges, filter MPI process vertices
        std::function{[&](yloc::vertex_descriptor_t v) {
            return graph[v].type->is_a<yloc::MPIProcess>();
    }});

    // create distance matrix for vertices from 'filtered_graph'
    // with #hops as distance metric:
    auto matrix = yloc::distance_matrix(filtered_graph);

    // print distance matrix for 'filtered_graph'
    std::for_each(matrix.begin(), matrix.end(),
        [&](std::pair<int, std::vector<int>> &rank_distances)
    {
        std::cout << "distance from rank "
                  << rank_distances.first << '\n';
        for(auto v : boost::make_iterator_range(
                boost::vertices(filtered_graph))) {
            std::cout << "to " << graph[v].get<int>("mpi_rank").value()
                      << ": " << rank_distances.second[v] << '\n';
        }
    });

    yloc::finalize();
    MPI_Finalize();
}
```

4.2 SPARSEBASE

The purpose of SparseBase is to create a library that can efficiently handle sparse data structures, with a focus on High-Performance Computing (HPC) applications. The library can work on tensors and objects made up of these tensors, such as graphs and hypergraphs. Our goal with this library is to offer a comprehensive solution for sparse data processing. As such, we have included various modules within SparseBase that allow for easy representation of sparse data, input/output operations, preprocessing, and feature extraction. These modules work in conjunction to provide a straightforward yet robust API that we believe will greatly enhance the research experience when working with sparse data.

To prioritize HPC applications, SparseBase offers templates for extensive type support. This allows developers to avoid overflow issues and prevent unnecessary memory usage when dealing with data types that are too large. By using templates, the kernel implementations can be optimized for the exact data type of the input data. It is worth noting that the level of granularity in the templates provided by SparseBase is high. For instance, a single format representing a matrix can have different data types assigned to its row and column IDs, the variables that store the number of non-zeros in the matrix, and the variables that store the matrix elements themselves.

The three data types used in SparseBase are referred to as `IDType`, `NNZType`, and `ValueType`. Figure 14 depicts a typical scenario involving SparseBase. First, the developer creates a graph Object and reads a connectivity structure stored in a matrix market file (`.mtx`). Next, they create an object for RCM reordering and use it to reorder the graph object. SparseBase handles this seamlessly through the `GetReorder` function call. It automatically determines the appropriate `Formats` types for the specific reordering task, detects that the input is not in one of these formats, converts the input to the correct format, and performs the kernel operation.

```
Graph<int, int, int> graph;
graph.ReadConnectivityFromMTXToCOO(filename);
Format * connectivity = graph.get_connectivity();
    RCMReorder<int, int, int> reordering;
int * order = reordering.GetReorder(connectivity);
    // hidden conversion from COO to CSR
```

Figure 14 *CRTP structure of the format system*

Formats are the foundational data structures used by SparseBase, with tensors, matrices, and graphs all using them as their underlying structures. The format system aims to maintain an HPC mindset while remaining as abstract and extensible as possible.SparseBase implements conversions between sparse data representations and features numerous preprocessing algorithms that can operate on any format without restrictions. This is primarily because certain formats may be more efficient for specific algorithms or because the integration of existing code written in a particular format may be necessary. Consequently, SparseBase includes a sophisticated conversion system that allows all algorithms to be used with all formats, provided that conversion between the formats is feasible and that the relevant conversion functions are registered with the library.

Preprocessing operations in SparseBase involve `Format` objects such as reordering, partitioning, graph coarsening, matrix factorization, and tensor decomposition. Preprocessing is carried out using preprocessing objects, each of which performs a specific operation and can contain multiple functions that handle various Formats. Each instance includes a matching and conversion mechanism for these functions. Preprocessing simplifies tasks such as reordering, where developers can add a new reordering implementation (e.g., RCM ordering) by adding a function that works on a single format. Even if the input format type is different, users can still use this implementation. The library will convert the input to the appropriate format and execute the function.

The Feature extraction API in SparseBase aims to provide a convenient and easy-to-use interface for extracting features from mathematical representations such as graphs, matrices, and tensors. The design is based on two main ideas: first, users should be able to retrieve all necessary features with a single function call, and second, the library should switch to more efficient versions of the same computation if available. The first idea makes the API simple to use, while the second idea is in line with the library's HPC focus.

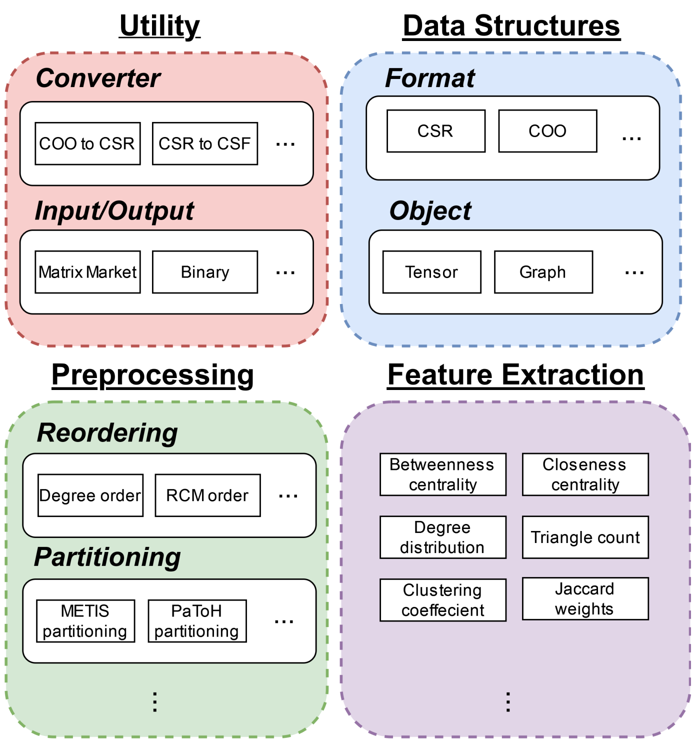


Figure 15 *The different components making up the* SparseBase *library.*

The newest version of SparseBase includes an experimentation pipeline, which enables users to test the usefulness of different storage formats and preprocessing techniques in their downstream tasks. To use this feature, users need to define their tasks using a well-defined interface and specify the formats and techniques they wish to experiment with. The library will then execute the tasks with the selected configurations and provide the user with timing information to choose from.

SparseBase also focuses on visualization as an essential aspect of comprehending sparse data. The goal is to provide convenient and intuitive data visualization tools and to achieve this, a visualization prototype has been developed. The prototype combines Python visualization libraries with the experimentation pipeline in SparseBase to illustrate how the library's experimentation pipeline can be complemented with visualization techniques. Figure 16 shows the prototype in action. The library is actively working on adding more visualization mechanisms, and the visualizations presented in the prototype will eventually be producible directly from C++ through library calls.

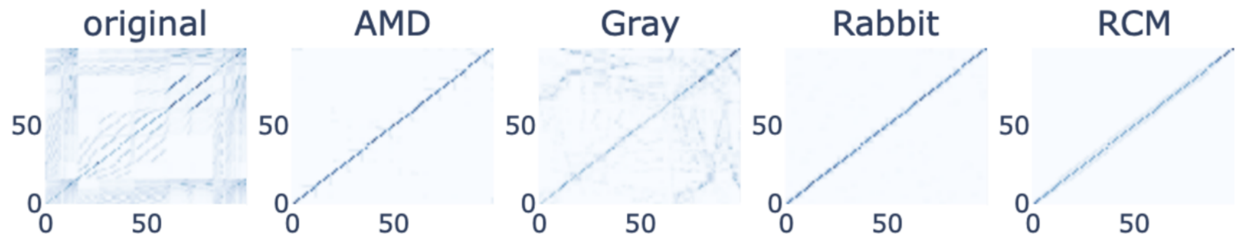


Figure 16 *The matrix Dzienkoski/gsm_106857 from SuiteSparse*[43] *is reordered using 4 different reordering algorithms, namely Approximate Minimum Degree, Gray, Rabbit, and RCM reordering. The sparsity pattern of the matrix with each reordering is shown as a heatmap.*

SparseBase aims to be a powerful library that offers a wide range of functionalities to its users. In addition to its built-in features, it also integrates with several third-party tools and file formats that are commonly used in the field. These include popular partitioning tools such as Metis, Pulp, and PaToH, as well as reordering algorithms such as AMD and Rabbit. The library supports various sparse data representations such as CSR, COO, CSC, and dense array formats, allowing users to work with different types of data efficiently. Furthermore, SparseBase integrates with various I/O file formats such as `mtx`, `edge list`, `PIGO`, and `binary` formats, enabling seamless data exchange between different applications and systems. By incorporating these tools and file formats, SparseBase makes it easier for users to work with sparse data and streamlines the development process, ultimately enhancing their productivity and effectiveness in their respective fields.

5 CONCLUSIONS

During the second year of the SparCity project, the research results have started to take shape. These include (1) new methods for understanding the achievable performance of typical kernels of sparse computation and parallelization; (2) new stand-alone software tools for performance characterization, profiling, sparse data generation, and GUI-supported characterization monitoring/visualization of hardware; (3) new software libraries that can streamline the programming effort of representing/extracting the system topology of supercomputers and handling/transforming the different storage formats and kernel operations of sparse data.

The usefulness of these components of the SparCity framework has been demonstrated and validated by a number of realistic examples relevant to sparse computations. Several of the components have already found their way into some of the real-world applications of SparCity (WP5). During the remaining time of the project, an extensive effort will be made to further improve the SparCity framework and deliver substantial benefits to real-world applications.

REFERENCES

A64FX Microarchitecture Manual. Version 1.5. Fujitsu Limited. 2021. URL: https://github.com/fujitsu/A64FX/blob/master/doc/.

Advanced Micro Devices, Inc. *AMD uProf*. Accessed: 2021-05-14.

Anderson, Jennifer M. et al. “Continuous Profiling: Where Have All the Cycles Gone?” *ACM Trans. Comput. Syst.* 15.4 (1997), pp. 357–390. ISSN: 0734-2071. DOI: 10.1145/265924.265925. URL: https://doi.org/10.1145/265924.265925.

Azad, Ariful et al. “The reverse Cuthill-McKee algorithm in distributed-memory”. *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2017, pp. 22–31.

Bienz, Amanda, William D. Gropp, and Luke N. Olson. “Improving Performance Models for Irregular Point-to-Point Communication”. *Proceedings of the 25th European MPI Users' Group Meeting*. 2018, pp. 1–8. DOI: 10.1145/3236367.3236368.

Bienz, Amanda, William D Gropp, and Luke N Olson. “Reducing communication in algebraic multigrid with multi-step node awar e communication”. *The International Journal of High Performance Computing Applications* 34.5 (2020), pp. 547–561.

Boost. *Boost C++ Libraries*. http://www.boost.org/.

Broquedis, François et al. “hwloc: A generic framework for managing hardware affinities in HPC applications”. *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE. 2010, pp. 180–186.

Cuthill, Elizabeth. “Several Strategies for Reducing the Bandwidth of Matrices”. *Sparse Matrices and their Applications*. Springer, 1972, pp. 157–166.

Dhandhania, Sunidhi et al. “Explaining the Performance of Supervised and Semi-Supervised Methods for Automated Sparse Matrix Format Selection”. *50th International Conference on Parallel Processing Workshop*. 2021, pp. 1–10.

Drongowski, Paul J. *An introduction to analysis and optimization with AMD CodeAnalyst™ Performance Analyzer*. Tech. rep. Advanced Micro Devices, Inc., 2008.

— *Instruction-Based Sampling: A New Performance Analysis Technique for AMD Family 10h Processors*. https://pdfs.semanticscholar.org/5219/4b43b8385ce39b2b08ecd409c753e0efafe5.pdf. 2007.

Golder, ER and JG Settle. “The Box-Müller Method for Generating Pseudo-Random Normal Deviates”. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 25.1 (1976), pp. 12–20.

Greathouse, Joseph L. *AMD Research Instruction Based Sampling Toolkit*. https://github.com/jlgreathouse/AMD_IBS_Toolkit. 2017.

— *Re: Error : IBS profiling is disabled in your BIOS*. https://community.amd.com/t5/general-discussions/error-ibs-profiling-is-disabled-in-your-bios/td-p/55043. AMD Community.

— *Re: IBS not available on EPYC 7451 ?* https://community.amd.com/t5/server-gurus-discussions/ibs-not-available-on-epyc-7451/m-p/258228. AMD Community.

Gropp, William, Luke N. Olson, and Philipp Samfass. “Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test”. *Proceedings of the 23rd European MPI Users' Group Meeting*. 2016, pp. 41–50. DOI: 10.1145/2966884.2966919.

Ilic, Aleksandar, Frederico Pratas, and Leonel Sousa. “Cache-aware Roofline model: Upgrading the loft”. *IEEE Computer Architecture Letters* 13.1 (2013), pp. 21–24.

Intel. *Intel Microarchitecture Codename Nehalem Performance Monitoring Unit Programming Guide*. https://software.intel.com/sites/default/files/m/5/2/c/f/1/30320-Nehalem-PMU-Programming-Guide-Core.pdf. 2010.

Intel Corporation. *VTune Profiler*. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html. [Online; visited June-2022].

Kolodziej, Scott P et al. "The suitesparse matrix collection website interface". *Journal of Open Source Software* 4.35 (2019), p. 1244.

Lachaize, Renaud, Baptiste Lepers, and Vivien Quéma. "MemProf: A Memory Profiler for NUMA Multicore Systems". *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*. Boston, MA: USENIX Association, 2012, p. 5.

Langguth, Johannes et al. "Parallel performance modeling of irregular applications in cell-centered finite volume methods over unstructured tetrahedral meshes". *Journal of Parallel and Distributed Computing* 76 (2015), pp. 120–131. DOI: 10.1016/j.jpdc.2014.10.005.

Liu, Xu and John Mellor-Crummey. "A Data-Centric Profiler for Parallel Programs". *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. Denver, Colorado: Association for Computing Machinery, 2013. DOI: 10.1145/2503210.2503297. URL: https://doi.org/10.1145/2503210.2503297.

— "A Tool to Analyze the Performance of Multithreaded Programs on NUMA Architectures". *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 259–272. DOI: 10.1145/2555243.2555271. URL: https://doi.org/10.1145/2555243.2555271.

— "Pinpointing Data Locality Problems Using Data-Centric Analysis". *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. IEEE Computer Society, 2011, pp. 171–180.

Liu, Xu and Bo Wu. "ScaAnalyzer: a tool to identify memory scalability bottlenecks in parallel programs". *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2015, pp. 1–12. DOI: 10.1145/2807591.2807648.

Luk, Chi-Keung et al. "Pin: building customized program analysis tools with dynamic instrumentation". *Acm sigplan notices* 40.6 (2005), pp. 190–200.

Marques, Diogo, Aleksandar Ilic, and Leonel Sousa. "Mansard Roofline Model: Reinforcing the Accuracy of the Roofs". *ACM Trans. Model. Perform. Eval. Comput. Syst.* 6.2 (2021). ISSN: 2376-3639. DOI: 10.1145/3475866. URL: https://doi.org/10.1145/3475866.

McCurdy, Collin and Jeffrey Vetter. "Memphis: Finding and fixing NUMA-related performance problems on multi-core platforms". *2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*. 2010, pp. 87–96. DOI: 10.1109/ISPASS.2010.5452060.

Merrill, Duane and Michael Garland. "Merge-Based Sparse Matrix-Vector Multiplication (SpMV) Using the CSR Storage Format". *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2016.

Murphy, Richard C et al. "Introducing the Graph 500". *Cray Users Group (CUG)* 19 (2010), pp. 45–74.

Narayanan, Ramanathan et al. "MineBench: A Benchmark Suite for Data Mining Workloads". *2006 IEEE International Symposium on Workload Characterization*. 2006, pp. 182–188. DOI: 10.1109/IISWC.2006.302743.

Nobre, Ricardo et al. "Exploring the Binary Precision Capabilities of Tensor Cores for Epistasis Detection". *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2020, pp. 338–347.

Roy, Probir and Xu Liu. "StructSlim: A Lightweight Profiler to Guide Structure Splitting". *Proceedings of the 2016 International Symposium on Code Generation and Optimization*. Barcelona, Spain: Association for Computing Machinery, 2016, pp. 36–46. DOI: 10.1145/2854038.2854053. URL: https://doi.org/10.1145/2854038.2854053.

Sasongko, Muhammad Aditya et al. "ComDetective: A Lightweight Communication Detection Tool for Threads". *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Denver, Colorado: Association for Computing Machinery, 2019. DOI: 10.1145/3295500.3356214. URL: https://doi.org/10.1145/3295500.3356214.

Sasongko, Muhammad Aditya et al. "ReuseTracker: Fast Yet Accurate Multicore Reuse Distance Analyzer". *ACM Trans. Archit. Code Optim.* 19.1 (2021). ISSN: 1544-3566. DOI: 10.1145/3484199. URL: https://doi.org/10.1145/3484199.

Thune, Andreas et al. "Detailed Modeling of Heterogeneous and Contention-Constrained Point-to-Point MPI Communication". *IEEE Transactions on Parallel and Distributed Systems* 34.5 (2023), pp. 1580–1593. DOI: 10.1109/TPDS.2023.3253881.

Unat, D. et al. "Trends in Data Locality Abstractions for HPC Systems". *IEEE Transactions on Parallel and Distributed Systems* 28.10 (2017), pp. 3007–3020.

Wang, Endong et al. "Intel math kernel library". *High-Performance Computing on the Intel® Xeon Phi™*. Springer, 2014, pp. 167–188.

Williams, Samuel, Andrew Waterman, and David Patterson. "Roofline: An Insightful Visual Performance Model for Multicore Architectures". *Commun. ACM* 52.4 (2009), pp. 65–76. ISSN: 0001-0782. DOI: 10.1145/1498765.1498785.

Yasin, Ahmad. "A top-down method for performance analysis and counters architecture". *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. 2014, pp. 35–44.

6 HISTORY OF CHANGES

Version	Author(s)	Date	Comment
0.1	Xing Cai	14.03.2023	Initial draft skeleton for PO and reviewers
0.2	Aleksandar Ilic	23.03.2023	Major changes in the tool section
0.3	Didem Unat	24.03.2023	Major changes in the tool section
0.4	Xing Cai	24.03.2023	Major changes in the method section
0.5	Johannes Langguth	26.03.2023	Major changes in the method section
0.6	Kamer Kaya	28.03.2023	Proofread

Table 2 *Document History of Changes*