# Performance Analysis of Array Database Systems in Non-Uniform Memory Architecture

Simone Dominico, Eduardo C. de Almeida, Marco A. Z. Alves
*Federal University of Paraná*
(sdominico, eduardo, mazalves)@inf.ufpr.br

Jorge A. Meira
*University of Luxembourg*
jorge.meira@uni.lu

***Abstract*—Array Database Management Systems (Array databases) support query processing over multi-dimensional data. Data storage is implemented with non-linear structures to mitigate the shortcomings of the relational model when dealing with raw binary data, such as images, time series, and others. Due to data-hungry nature of multi-dimensional data applications, array databases must ideally provide a linear speedup when using a multi-processing system. When dealing with Non-Uniform Memory Access (NUMA) machines, array databases may require massive data movement across the nodes resulting in a severe performance impact, depending on the user operation. In this paper, we analyze the performance impact of the NUMA architecture in the SAVIME and SciDB array databases running five different well-known static thread pinning strategies. Our experiments showed a maximum speedup of these different strategies by 2.49x for SAVIME and up to 1.40x for SciDB. We also observed that these static strategies only yield 48% from the potential speedup (and 26% of the energy reduction), opening a new research topic.**

***Index Terms*—NUMA, Array databases, Thread placement.**

## I. Introduction

Relational Database Management Systems (RDBMSs) are widely used in many applications due to the efficient storage scheme provided by the relational model, which allows reducing the storage space and easing data maintenance. This model also allows users to combine and fetch data flexibly by using a declarative query language. However, the relational model is not ideal for storing and analyzing scientific data due to its limitations and incompatibility to work directly with the data format generated in scientific simulations and experiments (i.e., data array) [1], [2]. The incompatibility between the array and the relational model requires a series of data transformations to execute queries in order to analyze scientific data [3].

Array databases implement multidimensional data models to serve many data-hungry applications that do not fit the traditional relational model, such as scientific simulations, data exploration, machine learning, biological structures, to name but a few [4], [5]. Usually, such areas quickly produce large binary data volumes to be processed and analyzed at once. Array query languages provide specific multidimensional operations, such as geometric and linear algebra operations (e.g., data slices, array transpose, addition, subtraction, opposite array) and use parallelism to reduce query response time.

This work was partially supported by the Serrapilheira Institute (grant number Serra-1709-16621), CAPES and CNPq (Brazilian Government).

At the same time, high performance systems use NUMA architectures to leverage the high number of CPU cores within the same shared memory. In NUMA, the memory access latency can vary depending on the address being accessed (e.g. access to near or far-away computing nodes). In this scenario both, RDBMSs and Array databases may present different performance depending on how query threads are pinned and how data mapping occurs across the nodes.

Query processing models used by array databases are similar to the ones used by RDBMS. For example, SAVIME [6] uses the materialization query model, whereas SciDB [3] uses the iterator query model. Besides, by default, the array databases rely on the Operating System (OS) to map query threads to CPU cores, as observed in traditional RDBMS. The OS uses load balance strategies to spread the threads all over the cores, without considering specific characteristics of the interaction between operations running in the database and the multi-core processor architecture. However, previous work on RDBMS [7] observed that the OS attempts to load balance between NUMA nodes generated a negative impact on performance, once the data locality unawareness of the OS increased the interconnection traffic between nodes. Eventually, one could ask if this negative impact holds when using NUMA architecture to support other database systems models.

In this paper, we evaluate the performance of array databases executing in a NUMA architecture. We use multiple well-known static thread pinning strategies to measure potential improvements in query performance. As far as we know, we are the first to evaluate such impact on Array databases. In our experiments, we have chosen the SAVIME and SciDB state-of-the-art systems that implement a multidimensional array data model from scratch (i.e., no adaptations to the relational model). Our main contributions in this paper are:

**Traditional techniques comparison:** We analyze the speedup and energy impact of five different thread pinning strategies for NUMA systems when executing SAVIME and SciDB. Using different strategies we could observe a maximum speedup of $2.49\times$ ($2.09\times$ less energy) with $5\times$ less remote memory accesses for SAVIME. For SciDB, we observed a speedup of up to $1.40\times$ ($1.47\times$ less energy) and reduction on the remote memory access by $4.1\times$.

**Maximum performance analysis:** We show that traditional techniques for distributing threads across NUMA cores are still far from a perfect point of improvement. Our experiments

show that, on average, 52% performance and 74% energy improvements are still available to be collected by newer and improved techniques.

## II. Array Database Systems

In this section, we briefly describe the data model of Array databases and the query operator to slice multidimensional data that we used in our evaluations.

The array data model represents data using $n$ named dimensions, and each of them has contiguous indexes. The multiple dimensions simplify the data access and analysis through different views. In this model, each cell belonging to the array contains attributes with the same data type. Figure 1 illustrates the array data model with three dimensions being respectively $latitude$, $longitude$, and $year$[1]. The values are accessible through a set of indexes. Experts can quickly analyze, in the example of Figure 1, the change in temperature over the years.

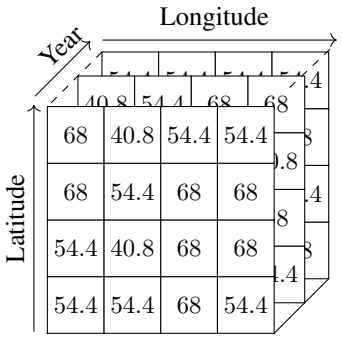


Fig. 1: Model of a multidimensional array: temperature on many latitudes, and longitudes over the years.

Along with the array data model concept, a wide range of Array database has emerged, such as RasdMan [8], ArrayStore [9], SciDB [3], SciQL [10] and SAVIME [6]. In this paper, we focus on two full-stack databases, SAVIME [6] and SciDB [3]. They implemented the array model from scratch without adaptations in the relational model. In scalable multi-processing machines, this allows the distribution of data chunks on separated machine nodes. Chunk is a physical representation of an array, and both systems split the arrays into chunks according to the stored data types. The chunk size and format depend on the density of the array. For dense arrays, all the chunks will have the same size. On the other hand, when arrays are sparse, chunks may have different sizes and formats. Non-regular chunks (i.e. sparse arrays) are prone to be non-uniformly distributed, causing penalties in query processing.

Querying an Array database extracts data from arrays using nested functions calls. Chunks are pipelined through query operators. The array data structure makes faster access to a set of cells using the indexes. Also, if a query frequently uses a chunk for faster access, the chunk is kept in memory [11].

The SAVIME and SciDB databases support an Array Functional language (AFL) with a series of operators defined as functions [2], [5]. In this paper, we focus on the operator classified by SciDB as the selection operator. The same operation has different names in the Array database. In SAVIME is $subset$, and in SciDB, it is $subarray$. From this point, we refer to both operations as $subarray$. The $subarray$ operator uses dimension indexes to fast data access, in both SAVIME and SciDB, it selects data in a range. The $subarray$ operator creates a copy of the data within a range.

Array databases implement $subarray$ operators in different ways. SAVIME finds cells between the range using the filter and generates many chunks with different sizes as a result. SciDB decodes the compressed binary data for the chunks and redistributes data to produce a new chunking configuration for the results that are within the range of interest. The $subarray$ is a simple operation in an array data model. However, according to the query selectivity, we may require processing all the chunks.

Considering that Array database systems must use multiple threads to provide scalable performance, in the next section we present the NUMA architecture and its interference on such multi-threaded systems.

## III. NUMA Architectures

NUMA architecture is a common design to provide high throughput with a unified memory view. Ideally, these architectures provide high performance by maximizing computing power and data sharing. Such characteristics suit perfectly to support an Array database.

NUMA systems are typically formed by multi-core processors grouped in nodes that share the memory with non-uniform access latency. NUMA architecture has node-to-node communications links, which provide high bandwidth with separate memory controllers per node. In these NUMA nodes, multiple cache levels and memory sharing schemes among the nodes compose the memory hierarchy. Each of the referred nodes can access both local memory bank and remote memory banks from neighboring nodes (see Figure 2).

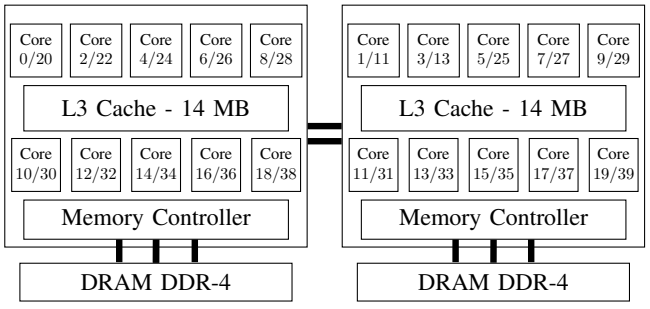


Fig. 2: Example of a NUMA architecture with 2-nodes inspired on Intel Xeon Silver 4114.

NUMA effects can cause significant performance problems. Considering that different data and thread allocation within a NUMA system may provide different latencies, it becomes essential to analyze such mapping for each specific application.

[1] geoserver.geo-solutions.it/edu/en/multidim/netcdf/netcdf$_basics.html$

Previous research proposed several thread pinning strategies to provide high performance depending on the application behavior [12]–[19].

Nonetheless, the critical task of mapping the threads over the available NUMA-cores is usually OS managed. For example, Linux focus on thread mapping in a load balance approach. Also, the OS allocates memory pages next to the node in which the first access to the page occurs. Altogether, the first-touch can be exploited to initialize parallel data between NUMA nodes. If threads are moved to maintain load balance, the standard OS strategy will not be the best scenario, and it will also affect the Array database performance. Since the query processing threads move around different NUMA nodes, the OS generates unnecessary thread migration and latency penalties due to remote data access. The result is a potentially significant impact on Array database performance.

Thus, our challenge is pinning threads on specific cores to benefit from the local memory, and to avoid the migration caused by OS attempts to keep load balance.

### A. Thread Pinning Strategies

We now present five well known thread pinning strategies that are used in our experiments. Hence, we aim to analyze the impact of these strategies when compared to the baseline (i.e. OS scheduling strategy). We use for Savime and SciDB, OpenMP (version 4.5) [20] thread affinity and $taskset$[2] respectively, to pinning threads.

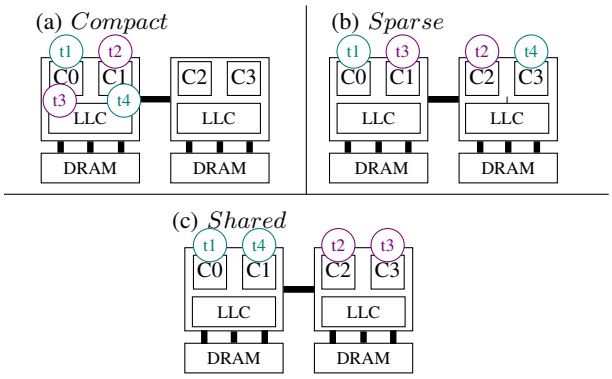


Fig. 3: Compact, Sparse and shared thread pinning strategies on a two NUMA node with four threads.

**Baseline:** To provide a commonly used baseline, we decided to measure the OS thread pinning performance without user interference. The OS uses a load balance to distribute threads over all the NUMA nodes, to maximize the usage of cores.

**Strategy 1 - Compact:** In Figure 3(a) we present the $compact$ strategy. The thread pinning follows the order of threads creation and only uses one NUMA node. This strategy can provide workload benefits with high data reuse. However, data size is bounded to the memory available on the node.

**Strategy 2 - Sparse:** The $sparse$ strategy distributes the threads equally among the nodes, one thread per node. For instance, $t1$ is allocated in node 0, $t2$ in node 1, and so on, as we show in Figure 3(b). Hence, our goal is to measure the performance when scattering the threads in the nodes and pinning them in one core.

**Strategy 3 - Shared:** The $shared$ strategy aims to pin sets of threads that work on the same chunk to a single NUMA node. These threads share the Last-Level Cache (LLC) node, as shown in Figure 3(c). We use this strategy to analyze if data reuse has a positive impact on minimizing the effects of the NUMA architecture.

**Strategy 4 - Petri net:** In this strategy, we use the same strategy presented in [7]. The dynamic mechanism implements an abstract model based on the Petri net. The goal of this mechanism is to maintain a local optimal number of cores to tackle the current database workload. The authors model performance states that must be satisfied to trigger the allocation of cores in the NUMA architecture. The assumption is that a minimum number of cores maintain performance. The primary metric to achieve the optimal number of cores is the CPU load.

**Strategy 5 - Random:** Here, the Array database threads are pinned randomly through the cores in all NUMA nodes. We generated 20 random allocations to verify if workloads have different behaviors. We tried to avoid identical core pinning, as in the compact and sparse strategies, when the threads are pinned in the same corresponding cores. The random strategy intends to identify if there is a thread pinning that improves performance compared to the other strategies evaluated. Results only show the best case concerning this strategy.

## IV. Experimental Evaluation

We now evaluate the performance of SAVIME and SciDB Array databases in a NUMA architecture using the previously described strategies of thread pinning.

The NUMA machine used in our experiments has two nodes, each node with an Intel Xeon Silver 4114 (with Skylake microarchitecture). Each Xeon socket has ten cores with private L1(I+D) cache (32 KB), private L2 cache (1 MB), and a shared L3 cache (14 MB). The two NUMA nodes are interconnected by a Quick Path Interconnect (QPI) [21] link 4.x with 21.5 GB/S bandwidth. The machine includes 128 GB DDR-4 main memory and 14 TB of disk running the Ubuntu OS in version 18.04.01 LTS for SAVIME and 14.04.6 LTS for SciDB. The different OS versions were used according to the Array database documentation. We run an unmodified Linux kernel, version $4.15.0-121-generic$. To measure the hardware performance, we used the Intel Performance Counter Monitor (PCM) [22]. We used the SAVIME version (v.1.0) and SciDB version (v.19.11.5).

In our experiments, we set the maximum number of threads available for query execution to 20, which corresponds to the number of available physical cores. The workload has dense array based on data from the HPC4e BSC seismic benchmark [23]. We present the results considering the average over 10 executions for each thread pinning strategy and $baseline$.

[2]https://man7.org/linux/man-pages/man1/taskset.1.html

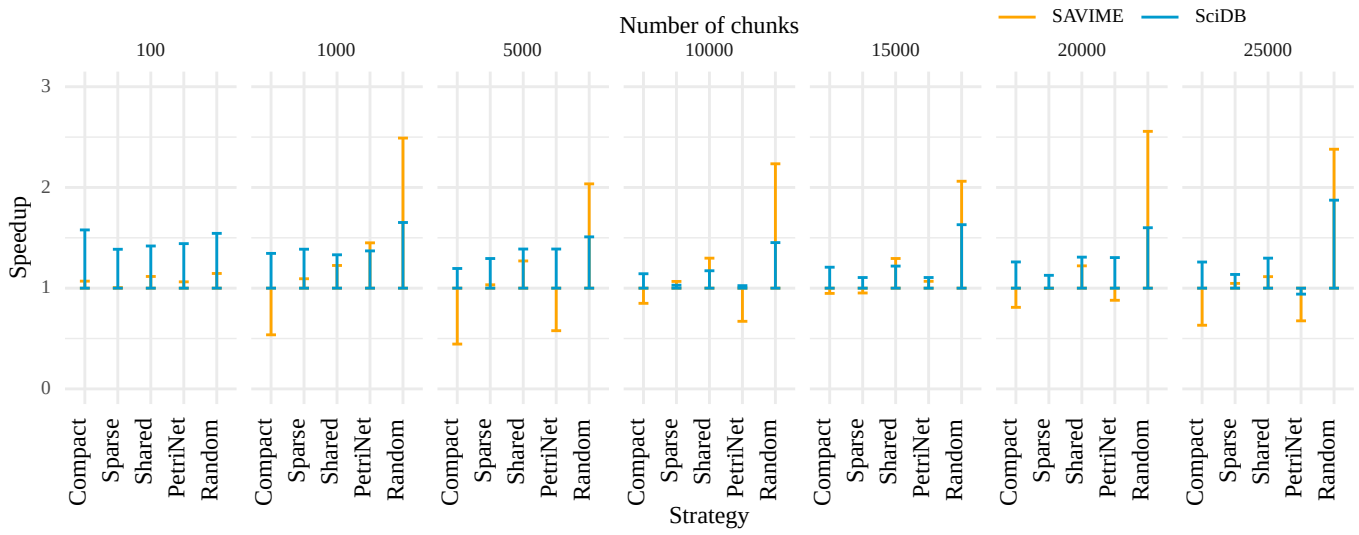


Fig. 4: Performance comparison of *subarray* operator in 1 GB database using different numbers of chunks in Array database.

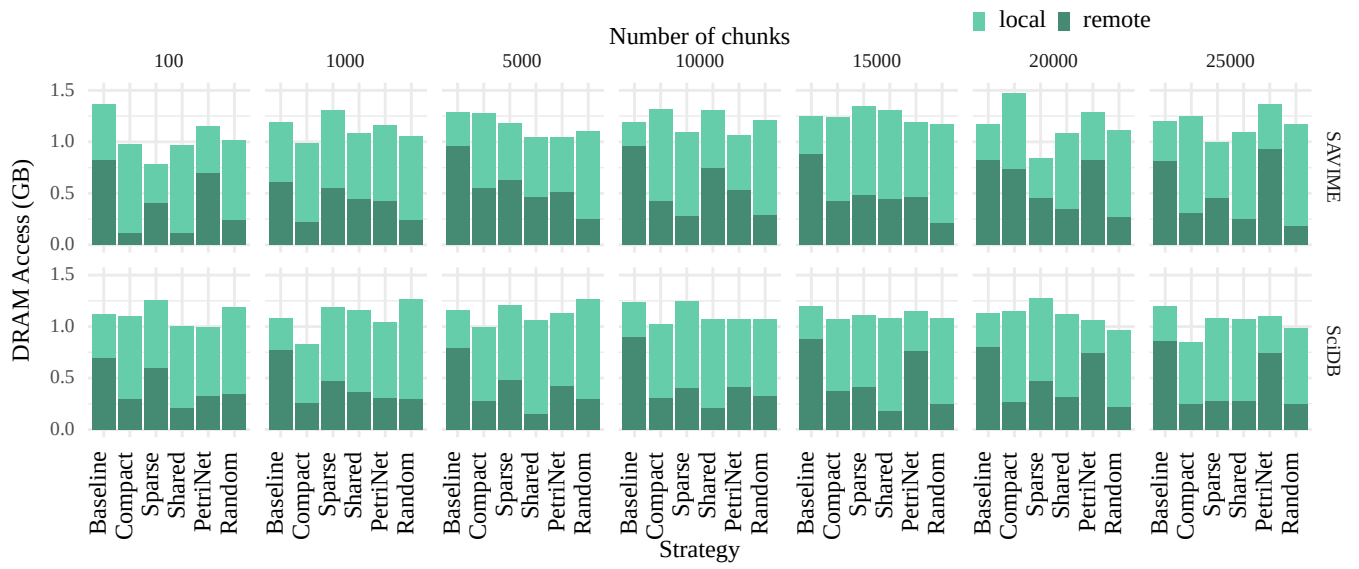


Fig. 5: Remote and local memory accesses in *subarray* operation in a 1 GB database using different numbers of chunks in Array database.

### *A. Subarray Operation Analysis*

We focus our analysis on the *subarray* operator. The *subarray* operator picks a data subset of a multidimensional array. This operator was chosen due to its simple memory access behavior. Such behavior moves large amounts of data in such a way that might have a great influence on NUMA systems. Besides, this operator has considerable application, since it is ones of the most studied one [2], [24]. This operation has a coalescing memory access pattern, and no cache data reuse due to its streaming data behavior. Thus, this memory access pattern generates a poor use of the memory hierarchy in NUMA architecture. Processing of *subarray* operation by several threads in parallel leads to data movement between NUMA nodes and a direct impact on Array database performance.

*1) Impact of the number of chunks:* This subsection focuses on the impact of NUMA architecture in an Array database for multiple numbers of chunks in a dataset of 1 GB. It means that, whenever we increase the number of chunks, the size of each chunk decreases.

Figure 4 depicts the results for SAVIME and SciDB in terms of speedup per amount of chunks when using different thread

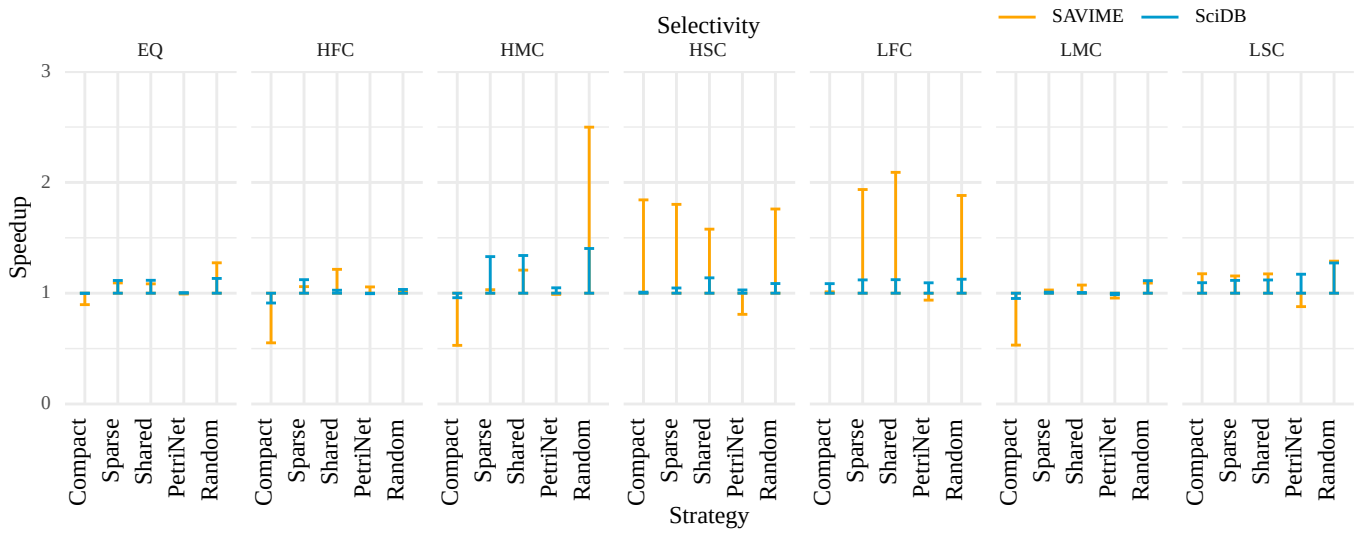


Fig. 6: Performance comparison of $subarray$ operator in a 50 GB database varying the operator selectivity in Array database.

pinning strategies compared to the OS scheduler ($baseline$). We can observe that some thread pinning strategies improve Array database performance. We notice that results associate the best performance to a larger number of chunks. This means that, when the threads work in a smaller chunk, exploiting spatial locality. Consequently, there is a higher thread pinning impact, which leads to, in this case, the best results.

Regarding the different thread pinning strategies we can see that $random$ strategy provides the most significant improvements. The $random$ strategy shows the best result among the 20 $random$ configurations that were tested. In the case of $random$ strategy reached a maximum acceleration of up to $2.55\times$ in SAVIME and $1.87\times$ in SciDB. This result indicates the need for a dynamic approach because, among some $random$ thread pinning, we found better performance in relation to the other static strategies. For example, the shared strategy achieves 47% of maximum performance.

Figure 5 shows the results of remote and local memory access during the execution of a $subarray$ operator. The local memory access has lower latency compared to the remote memory access. It means that the higher the number of local memory access, the lower are NUMA effects on the Array database. The variation in total access to DRAM memory through the use of thread pinning strategies shows that not all strategies have achieved a good use of cache memories. This creates a need to search for data in DRAM more times, and with greater latency. To better understand the relation between the speedup and the DRAM accesses results, please observe the 100 number of chunks (left most set on the figures). We can see a clear relationship between the total number of DRAM accesses and the final performance. It happens due to the high latency for getting data from DRAM memories, and it indicates a poor use of cache memories. Besides, the rise in the number of remote accesses plays a second hole in reducing the final speedup. The amount of remote accesses also indicates how well the scheduler pinned the threads on NUMA cores.

It is noticeable the performance scaling when the amount of chunks increases in the $subarray$ operation. This means that, by using smaller chunk sizes, the thread pinning can efficiently exploit the NUMA architecture. Here, we can observe in Figure 5 remote accesses. The $random$ strategy shows a reduction in remote access of $4.9\times$ for SAVIME chunks and $3.5\times$ for SciDB.

On the other hand, observing the $compact$ strategy, the thread pinning showed variations in speedup, delivering the worst performance compared to the $baseline$ (which does not pin threads). When looking at the remote and local accesses in Figure 5, we notice that $compact$ strategy reduced the number of remote access for SAVIME by 50%. The reason is that, the $subarray$ operation performs chunks most of the time, and each thread performs different chunks. This fact increases the probability of having many chunks being processed by threads at the same time, which also increase the contention on cache memories. Another relevant result is that $PetriNet$ did not show as much acceleration as we observed in an RDBMS. $PetriNet$ helps the OS allocating cores through the evaluation of CPU load. Unlike an RDBMS, the CPU load in the Array database maintained a high CPU usage [25]. Consequently, $PetriNet$ allocated all the cores of the NUMA architecture.

*2) Impact of Selectivity:* We now evaluate the impact of different query selectivity using a 50 GB dataset. To conduce this evaluation, we used a fixed number of 210 chunks. Here, we expected faster executions as we increased the selectivity, and only a small fraction of data should be gathered. We evaluated high (70%) and low (20%) selectivity queries on three chunk setups. Using just one chunk (HSC and LSC), on 20% of the chunks (HFC and LFC), on all chunks (HMC and LMC). We also present the exact query (EQ) with a $subarray$.

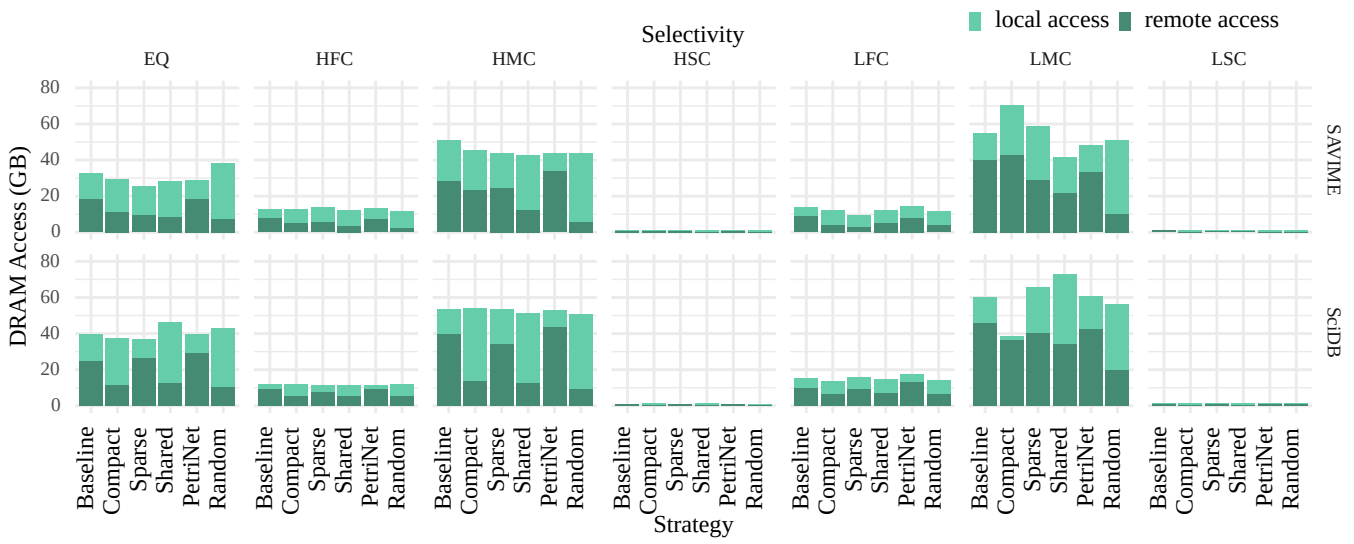


Fig. 7: Remote and local memory accesses in $subarray$ operation in a 50 GB database using different operator selectivity in the Array database.

Figure 6 presents the speedup, whereas Figure 7 brings the number of remote/local memory access. We noticed behavior differences in the $subarray$ operator when selectivity varied. When the $subarray$ operator has high selectivity, it needed to retrieve a small part of the data. Yet, low selectivity transferred more data in the memory hierarchy and generates more access in DRAM. This process indicates that when a there are many chunks and high selectivity, the data movement can be high despite the increase in selectivity. Hence, thread pinning guarantees more efficient use of the cache. Through the analysis of high selectivity results, the NUMA architecture presented a higher impact with queries that touched more chunks, with $2.49\times$ and $1.40\times$ speedup in SAVIME and SciDB, respectively. This occurred since the $random$ thread pinning strategy distributed the threads in such a way that it caused less remote memory access (with a reduction of $5\times$ in SAVIME and $4.1\times$ in SciDB). We can observe a decrease in the number of total accesses to DRAM memory with the use of the random strategy, and such reduction implies a good use of cache memory. One of the characteristics of $subarray$ operation is low data reuse, which indicates that the $random$ strategy could find a balance in threads, and did not compete for memory space in the NUMA architecture. Nevertheless, the $random$ strategy took advantage of the data spatial location. Also, shared strategy only yield 48% from the potential speedup achieved by the $random$ strategy.

Notice that, when the query is in only one chunk the compact strategy presented the best result with speedup of $1.84\times$ for SAVIME, with a query of high selectivity. In this case, as the query uses only one chunk, all threads are working in the same memory space. Therefore, these threads reuse data already loaded by other threads. Also, our experimental evaluation showed again that, when the number of chunks increased, the $compact$ strategy did not present a significant acceleration, disregarding selectivity. The $sparse$ and $shared$ strategies, on the other hand, showed similar speedup and remote accesses to memory. The $shared$ strategy tried to pin similar threads that work under the same chunk together in the same node. However, during the running query, each thread changed chunks. This behavior makes the $compact$ and $shared$ strategies similar because, from this moment on, the $shared$ strategy assumes the $sparse$ behavior and data may or may not be on same the NUMA node.

*3) Energy evaluation:* To measure the hardware performance, we used the Intel PCM [22]. Intel PCM tool provides the total power consumption of the main memory. Figure 8 shows the results of energy consumption in the DRAM memory (normalized by baseline) regarding experiments with 50 GB. We can observe that the presented behavior is similar to speedup results. The $random$ strategy reduced DRAM energy by 52% in SAVIME with high selectivity in many chunks ($HMC$). However, the $compact$ strategy increases energy consumption in the same experiment ($HMC$). For SciDB, the energy saving within the use of $random$ strategy was 31.86% with high selectivity in many chunks. The worst result in SciDB was from the use of the $compact$ strategy in $HMC$.

*4) NUMA effect on Array database:* During our evaluations of $subarray$ operator, we notice that the number of chunks has an impact on the Array database atop in the NUMA architecture. The results imply that the higher the number of chunks, the higher is data movement between the NUMA nodes, which explained remote accesses variation. Moreover, in our experiments, we used dense arrays, and the chunks are mapped closer in memory. In this case, as expected, the distribution of threads to all the NUMA nodes, performed

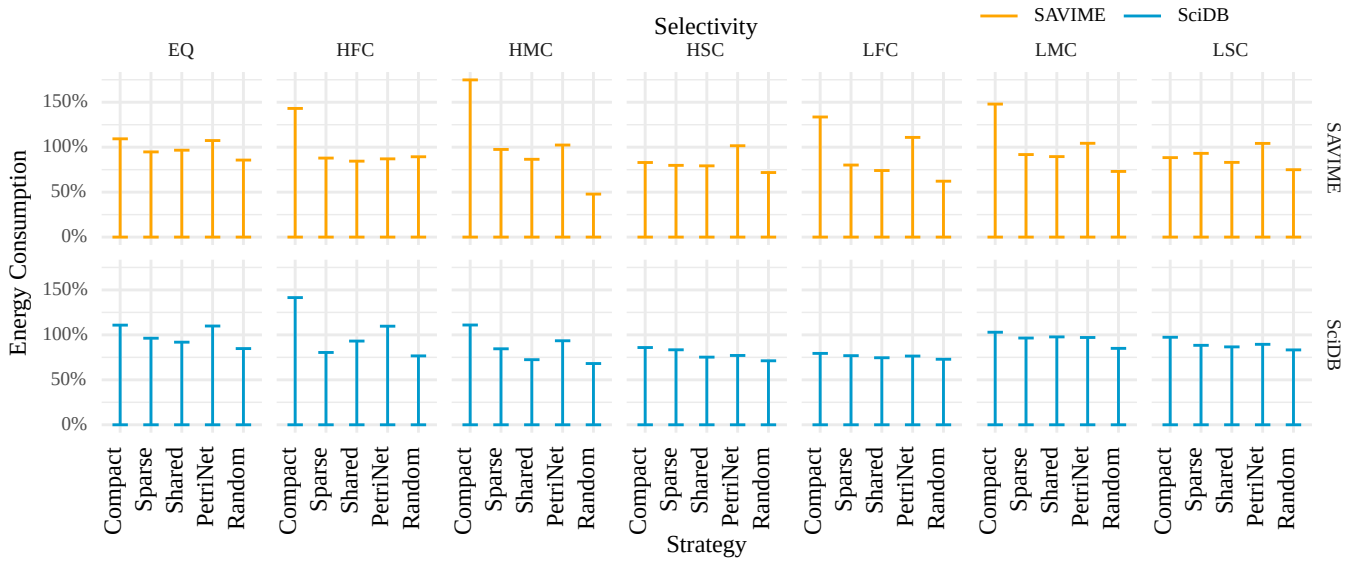


Fig. 8: Energy consumption in DRAM with 50 GB database using different operator selectivity in the Array database.

by the OS scheduler, led to an underuse of the architecture's full potential by the Array database. We conclude, therefore, that it is not the best scenario for Array database. Also, the thread pinning strategies provide lower energy consumption when presented best results in speedup. Finally, based on the experimental evaluation results, we can observe that simple thread pinning strategies provide moderate speedup, but the results obtained with the *random* strategy indicated the need for more sophisticated strategies for an Array database. Also, the finding best thread mapping is a difficult and complex task, since workload in Array database has different with patterns of memory access. The thread mapping and cannot directly be observed using a simple *random* strategy without incurring a high probe time and many combinations.

## V. Related Work

The impact of NUMA architecture on performance have motivated several recent works in different areas. In computer architecture, these researches employ different thread mapping techniques to minimize the effects of NUMA architecture and study the effects of NUMA under different views of architecture. For instance, the work presented in [12]–[18], [26] focuses on thread placement techniques based on memory access patterns and communication cost between nodes. However, these studies analyze generic applications without considering the specific operation that each application is running.

Researches that mitigate the effects of NUMA architecture in database systems gain momentum in the database research community. These researches have mainly focused on RDBMS on particular query operators or thread/data placement. In [27], [28] the authors present techniques to improve the performance of join in the NUMA architecture by directing the operation threads on specific NUMA nodes to mitigate the effects of the data movement.

There is related work focusing on thread/data placement use different techniques to designate the threads on the NUMA node where the data is allocated. In [29] Online Transaction Processing (OLTP) threads are pinned in hardware islands created through different separations of NUMA architecture nodes. In [30]–[35] both works focus in data partitioning, the data is allocated on the NUMA nodes and the threads are positioned statically where the data is allocated.

In contrast, other work [36], [37] use hardware configurations to mitigate the effects of NUMA architecture. New kernel to query processing with custom thread allocation policies are also proposed [38]. Besides, the proposal of a new database scheduler to control the dispatching of query fragments, is called "morsels" [39]. The "morsels" are statically pinned in specific cores to take advantage of the data location and avoid data movement between nodes.

In work [7], the authors present a multi-core allocation technique that reduced the effects of the NUMA architecture on RDBMS, reducing the number of cores that the OS could use to allocate the threads. A similar strategy presented by [33] uses OS policies to designate the number of resources needed and creates a communication between the OS and the RDBMS in execution.

Considering the database work, we observed that the main focus is RDBMS. In contrast, our research investigates the impact of the NUMA architecture on Array database. Also, we analyzed whether thread placement improves the performance of Array database on NUMA architectures. The reason why related approaches may not provide benefits for an Array database is related to the type of workload and operations performed that differ from a RDBMS.

## VI. Conclusions and Future Work

Based on the fact that RDBMS and Array database adopt similar strategies when using multi-thread parallelism, only the former has been extensively studied in terms of performance behavior when using NUMA architectures. In this paper, we evaluated the speedup and energy consumption impact of NUMA in two Array databases, SAVIME, and SciDB.

By using different thread pinning strategies on the evaluated Array databases, we showed how each strategy behaved. Our results support that NUMA architecture severly affects performance of $subarray$ operation in a Array database. Furthermore, we conclude that traditional techniques are still far from the maximum possible gains. Although several studies show results for traditional RDBMS [7], [27]–[33], [33]–[39]. As far as we know, we are the first to study the NUMA architecture effects in Array database.

Our next steps include understanding NUMA effects in other Array database operators and designing an array database scheduler that finds the best thread pinning.

## References


[1] S. Blanas, K. Wu, S. Byna, B. Dong, and A. Shoshani, "Parallel data analysis directly on scientific file formats," in *Proc. ACM SIGMOD Int. Conf. on Manag. of Data*, 2014, pp. 385–396.

[2] H. Lustosa and F. Porto, "SAVIME: A multidimensional system for the analysis and visualization of simulation data," *CoRR*, vol. abs/1903.02949, 2019.

[3] P. G. Brown, "Overview of scidb: large scale array storage, processing and analysis," in *Proc. ACM SIGMOD Int. Conf. on Manag. of data*, 2010, pp. 963–968.

[4] P. Baumann and S. Holsten, "A comparative analysis of array models for databases," in *Database theory and application, bio-science and bio-technology*, 2011, pp. 80–89.

[5] S. Kim, S. G. Sohn, T. Kim, J. Yu, B. Kim, and B. Moon, "Selective scan for filter operator of scidb," in *Proceedings of the 28th Int. Conf. on Scientific and Statistical Database Manag.*, 2016, pp. 1–4.

[6] H. Lustosa, N. Lemus, F. Porto, and P. Valduriez, "Tars: An array model with rich semantics for multidimensional data," in *Proc. of the ER Forum 2017 and the ER 2017 Demo Track*, 2017, pp. 114–127.

[7] S. Dominico, E. C. de Almeida, J. A. Meira, and M. A. Z. Alves, "An elastic multi-core allocation mechanism for database systems," in *IEEE 34th Int. Conf. on Data Eng. (ICDE)*, 2018, pp. 473–484.

[8] P. Baumann, P. Furtado, R. Ritsch, and N. Widmann, "The rasdaman approach to multidimensional database management," in *ACM Symp. on Applied Computing*, 1997, pp. 166–173.

[9] E. Soroush, M. Balazinska, and D. Wang, "Arraystore: a storage manager for complex parallel array processing," in *Proc. ACM SIGMOD Int. Conf. on Manag. of data*, 2011, pp. 253–264.

[10] Y. Zhang, M. Kersten, and S. Manegold, "Sciql: array data processing inside an rdbms," in *Proc. ACM SIGMOD Int. Conf. on Manag. of Data*, 2013, pp. 1049–1052.

[11] L. Gerhardt, C. Faham, and Y. Yao, "Accelerating scientific analysis with scidb," *Journal of Physics: Conf. Series*, vol. 664, no. 7, 2015.

[12] E. H. M. da Cruz, M. A. Z. Alves, A. Carissimi, P. O. A. Navaux, C. P. Ribeiro, and J.-F. Méhaut, "Memory-aware thread and data mapping for hierarchical multi-core platforms," *Int. Journal of Networking and Computing*, vol. 2, no. 1, pp. 97–116, 2012.

[13] E. H. Cruz, M. Diener, L. L. Pilla, and P. O. Navaux, "Hardware-assisted thread and data mapping in hierarchical multicore architectures," *ACM Trans. on Archit. and Code Optimization*, vol. 13, no. 3, pp. 1–28, 2016.

[14] F. Song, S. Moore, and J. Dongarra, "Analytical modeling and optimization for affinity based thread scheduling on multicore systems," in *Int. Conf. on Cluster Computing and Workshops*, 2009, pp. 1–10.

[15] B. Lepers, V. Quéma, and A. Fedorova, "Thread and memory placement on numa systems: Asymmetry matters," in *Proc. of the 2015 USENIX Conf. on Usenix Annual Technical Conf.*, 2015, pp. 277–289.

[16] P. Virouleau, F. Broquedis, T. Gautier, and F. Rastello, "Using data dependencies to improve task-based scheduling strategies on numa architectures," in *Euro-Par 2016*, 2016, pp. 531–544.

[17] G. C. Chasparis, M. Rossbory, and V. Janjic, "Efficient dynamic pinning of parallelized applications by reinforcement learning with applications," in *Euro-Par 2017*, 2017, pp. 164–176.

[18] I. Sánchez Barrera, M. Moretó, E. Ayguadé, J. Labarta, M. Valero, and M. Casas, "Reducing data movement on large shared memory systems by exploiting computation dependencies," in *Int. Conf. on Supercomputing*, 2018, pp. 207–217.

[19] M. Popov, A. Jimborean, and D. Black-Schaffer, "Efficient thread/page/parallelism autotuning for numa systems," in *Int. Conf. on Supercomputing*, 2019, pp. 342–353.

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

[21] Intel, "Maximizing multicore processor performance," Jan. 2019. [Online]. Available: https://www.intel.com/content/www/us/en/io/quickpath-technology/quickpath-technology-general.html

[22] F. P. Willhalm Thomas, Dementiev Roman, "Intel performance counter monitor," Dec. 2012. [Online]. Available: https://software.intel.com/en-us/articles/intel-performance-counter-monitor

[23] B. S. Center, "Hpc4e seismic test suite to increase the space of development of new modelling." Apr. 2016. [Online]. Available: https://www.bsc.es/news/bsc-news/new-hpc4e-seismic-test-suite-increase-the-pace-development-new-modelling-and-imaging

[24] S. Papadopoulos, K. Datta, S. Madden, and T. Mattson, "The tiledb array data storage manager," *Proc. VLDB Endow.*, vol. 10, no. 4, 2016.

[25] H. Lustosa, F. Porto, P. Blanco, and P. Valduriez, "Database system support of simulation data," *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1329–1340, 2016.

[26] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky, "A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous hpc platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 10, pp. 2176–2190, 2018.

[27] M.-C. Albutiu, A. Kemper, and T. Neumann, "Massively parallel sort-merge joins in main memory multi-core database systems," *Proc. VLDB Endow.*, vol. 5, pp. 1064–1075, 2012.

[28] C. Balkesen, G. Alonso, J. Teubner, and M. T. Özsu, "Multi-core, main-memory joins: Sort vs. hash revisited," *Proc. VLDB Endow.*, vol. 7, no. 1, pp. 85–96, 2013.

[29] D. Porobic, I. Pandis, M. Branco, P. Tözün, and A. Ailamaki, "Oltp on hardware islands," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 1447–1458, 2012.

[30] S. Bellamkonda, H.-G. Li, U. Jagtap, Y. Zhu, V. Liang, and T. Cruanes, "Adaptive and big data scale parallel execution in oracle," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1102–1113, 2013.

[31] T. Kissinger, T. Kiefer, B. Schlegel, D. Habich, D. Molka, and W. Lehner, "Eris: A numa-aware in-memory storage engine for analytical workloads," *Proc. VLDB Endow.*, vol. 7, no. 14, pp. 1–12, 2014.

[32] D. Porobic, E. Liarou, P. Tözün, and A. Ailamaki, "Atrapos: Adaptive transaction processing on hardware islands," in *2014 IEEE 30th Int. Conf. on Data Engineering (ICDE)*, 2014, pp. 688–699.

[33] J. Giceva, G. Alonso, T. Roscoe, and T. Harris, "Deployment of query plans on multicores," *Proc. VLDB Endow.*, vol. 8, no. 3, 2014.

[34] M. Gawade and M. Kersten, "Numa obliviousness through memory mapping," in *Int. Workshop on Data Manag. on New Hardware*, 2015.

[35] O. Ozturk, U. Orhan, W. Ding, P. Yedlapalli, and M. T. Kandemir, "Cache hierarchy-aware query mapping on emerging multicore architectures," *IEEE Trans. on Computers*, vol. 66, no. 3, pp. 403–415, 2016.

[36] S. R. Agrawal, S. Idicula, A. Raghavan, E. Vlachos, V. Govindaraju, V. Varadarajan, C. Balkesen, G. Giannikis, C. Roth, N. Agarwal *et al.*, "A many-core architecture for in-memory data processing," in *Int. Symp. on Microarchitecture*, 2017, pp. 245–258.

[37] M. Dreseler, T. Kissinger, T. Djürken, E. Lübke, M. Uflacker, D. Habich, H. Plattner, and W. Lehner, "Hardware-accelerated memory operations on large-scale numa systems." in *ADMS@ VLDB*, 2017, pp. 34–41.

[38] J. Giceva, G. Zellweger, G. Alonso, and T. Rosco, "Customized os support for data-processing," in *Int. Workshop on Data Manag. on New Hardware*, 2016, pp. 1–6.

[39] V. Leis, P. Boncz, A. Kemper, and T. Neumann, "Morsel-driven parallelism: a numa-aware query evaluation framework for the many-core age," in *Proc. ACM SIGMOD Int. Conf. on Manag. of Data*, 2014, pp. 743–754.