Impact of Co-run Contention on Sparse Matrix Format Selection for SpMV on Heterogeneous Consumer Devices

ANONYMOUS AUTHOR(S)

Sparse matrix format is crucial to the performance of SpMV, a pivotal kernel in machine learning, graph computing, scientific solvers, and many domains. Prior studies however have all assumed standalone executions of SpMV, and overlooked the effects of co-run contentions on format selection. Such cases are especially common on consumer devices that feature unified memory between heterogeneous processors. This paper addresses this important oversight by conducting the first known comprehensive empirical study on the effects of contention on sparse format selection. It uncovers a series of novel insights, including the quantified effects of contentions on SpMV and its format selection, the variations across architectures and industrial-grade libraries, the factors to consider for making format selection contention-aware, its potential benefits and catches, and some design guidelines.

1 INTRODUCTION

Sparse Matrix-Vector Multiplication (SpMV) is an important kernel used in scientific computing and in Deep Learning applications. It has motivated a high volume of research for optimization as it is applicable to many domains, such as linear equation solvers, graph connectivity, and sparse LU factorization [4, 40, 54]. Recently, sparsity has gained more traction in compressing deep neural networks [24, 28] and even higher traction in compressing transformer networks [14]. Maximizing the speed of SpMV has been an everlasting objective in high-performance library development, sparse compilations [5, 31–33], runtime adaptations [58, 59], and computer architecture designs [34].

SpMV involves multiplying a sparse matrix with a dense vector. Its performance is sensitive to the storage format of the sparse matrix, that is, how the elements in the sparse matrix are represented and stored in memory. There are numerous formats invented, four of which are illustrated in Figure 1. The speed of SpMV differs by as much as several times when different formats are used. The best format however differ across matrices, as well as towards different applications; no single format fits all [8, 44, 57].

Previous works have proposed to choose the best format using methods such as machine learning. Various methods have been developed, incorporating SVMs, decision trees, and convolutional neural networks [57–59]. These works carefully consider prediction accuracy, as well as the overhead incurred from the runtime prediction and format conversion overhead [9, 58, 59].

However, prior studies have not considered optimal format selection under *co-running environment*. Previous works were grounded under the assumption of a controlled computing environment, where SpMV kernels can run under an isolated hardware environment. But on real-world devices—especially consumer devices, the assumption rarely holds due to users' needs.

Consumer devices are devices for personal use, such as personal computers, smartphones, embedded systems. A consumer device often has multiple processes or apps running simultaneously, for several reasons: (i) System Services: Many background processes are system services essential for the proper functioning of a phone, like syncing data, managing network connections, and providing notifications; (ii) Multitasking: Smartphones are designed for multitasking, which allows apps to stay in the background so they can quickly resume when needed. This enhances the user experience by making app switching faster. For instance, while a social media app is continuously optimizing its ads layout, the user may be interacting with a personal assistant program in the front. (iii) Background Tasks: Apps may perform background tasks such as checking for updates, syncing data, or sending notifications.

The effect of co-run applications on consumer devices cannot be ignored since SpMV is a highly memory-bound kernel, where its arithmetic intensity is bounded by the memory system. So is the best format in a standalone run still the best in the presence of co-runs? How much performance loss may be caused by the isolated execution assumption when there are co-runs? How much potential is there if we make the format predictors contention-aware? What are the implications to the construction and deployment of format selectors? All these questions are crucial for maximizing the performance of SpMV in real-world settings. But to the best of our knowledge, they all remain open: No prior work has systematically examined the effects of co-run on format selection for SpMV. These questions become increasing important as the trend of AI deployment on consumer devices rapidly grows: IDC's latest forecast estimates that AI smartphone shipments will grow 364% year-over-year in 2024, reaching 234.2 million units, growing to 912 million units in 2028 implying a compound annual growth rate (CAGR) of 78.4% for 2023-2028 [1].

This work strives to address that important oversight. Its first part presents a comprehensive empirical study to find out the effects of co-run contention on choosing the best formats for SpMV. It focuses on consumer devices that are eqipped with heterogeneous processors and Unified Memory Architecture (UMA), where CPUs and accelerators share the memory bus and hence are more susceptible to co-run contention influence. Such devices are the norm in modern consumer devices such as laptops and smartphones, and are common in machines with Intel CPU-GPU integrated processors.

The study is comprehensive, covering 600 matrices, 3 industrial-quality SpMV libraries (NVIDIA cuSPARSE for NVIDIA GPUs, ARM Performance Library (ARM PL) for ARM mobile CPUs, and Intel Math Kernel Library (MKL) for Intel CPUs) on 2 models of user devices, and 6 co-run scenarios with 6 co-run contention levels in each. Through over 2 million measurements, the study reports 12 key findings (Section 8), such as (i) contention consistently causes significant degradation (on average 79-92%) performance degradation for SpMV, (ii) its impact on best format selection however varies significantly across libraries and architectures (40-42% matrices change their best formats due to contention on cuSPARSE, but only 3-18% on MKL and 7-11% on ARM PL), (iii) contention-aware prediction has clear advantages, but need additional factors to consider in its development and deployment.

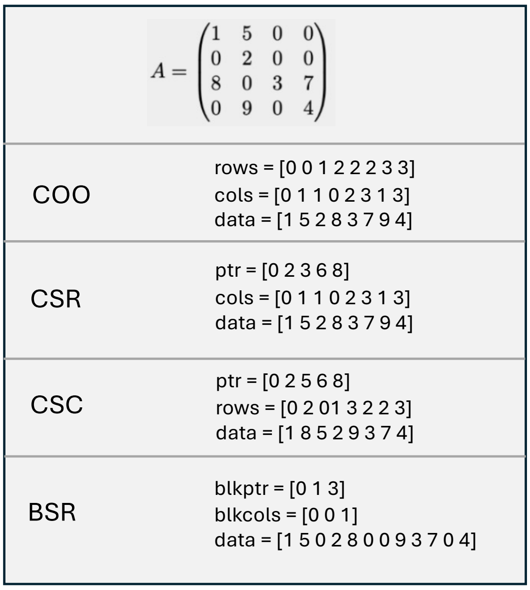
This work further explores the implications of the findings to the constructions of format predictors. It compares two concrete XGBoost-based predictors, with one being contention-oblivious, the other contention-aware, and gains up to 25% performance benefits for being contention-aware. It then goes beyond a concrete predictor, showing up to 45% loss of prediction accuracy for a series of simulated contention-oblivious predictors. It finally discusses the overhead for being contention-aware and potential strategies to deal with that.

Overall, this work makes the following main contributions:

- It gives the first comprehensive study on the impact of co-run contention on SpMV format selection.
- It reveals a set of quantitative and qualitative findings about the impact of co-run contention on SpMV and format selection.
- It shows the implications of the findings to the constructions of format predictors through both concrete and abstract predictors.
- It discusses the issues and gives guidelines for designing and deploying effective contention-aware format selectors.

2 BACKGROUND

This section provides the necessary background to understand SpMV and memory contention.



$$A = \begin{pmatrix} 1 & 5 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 8 & 0 & 3 & 7 \\ 0 & 9 & 0 & 4 \end{pmatrix}$$

COO	rows = [0 0 1 2 2 2 3 3]
	cols = [0 1 1 0 2 3 1 3]
	data = [1 5 2 8 3 7 9 4]

CSR	ptr = [0 2 3 6 8]
	cols = [0 1 1 0 2 3 1 3]
	data = [1 5 2 8 3 7 9 4]

CSC	ptr = [0 2 5 6 8]
	rows = [0 2 01 3 2 2 3]
	data = [1 8 5 2 9 3 7 4]

BSR	blkptr = [0 1 3]
	blkcols = [0 0 1]
	data = [1 5 0 2 8 0 0 9 3 7 0 4]

Fig. 1. Sparse formats and their corresponding arrays. For BSR, 2×2 blocks are used.

2.1 SpMV and Format Selection

Sparse linear algebra is considered one of the seven dwarfs of numerical methods that Colella considered essential for computational science and engineering [16]. The most used sparse linear algebra routine is SpMV (used in graph analytics, differential equations, linear solvers, etc). There are various different formats a sparse matrix can be represented as. Figure 1 shows several commonly used formats: Coordinate (COO), Compressed Sparse Row (CSR), Compressed Sparse Column (CSC), and Blocked Sparse Row (BSR).

COO is the easiest format to build and modify from. It involves three arrays (`row_idx`, `col_idx`, `values`): where `row_idx` is the row indices, `col_idx` is the column indices, and `values` is the value at that index. Each array is a length of *nnz*, the number of non-zeroes in the sparse matrix. This format is commonly used to convert to different formats, rather than perform SpMV.

CSR is the most space-efficient allowing for row indices to be compressed. This format is also represented by three arrays: (`row_ptr`, `col_idx`, `values`). `row_ptr` is the row pointer array, while other arrays follow the same convention as COO. `row_ptr` is the length of $m + 1$, where m is the number of rows in the sparse matrix and it is a cumulative sum of the number of non-zeroes in each row, thus why it's compressed on the row. CSC is the transposed format of CSR, having (`col_ptr`) and (`row_idx`) instead.

BSR is the most compute efficient, storing sub-blocks of matrices. BSR format is represented by three arrays: (`blkptr`, `blkcols`, `data`). `data` is an array holding the values in the blocks. The values are stored in row-major order within each block. The length of this array is the number of non-zero blocks multiplied by the number of elements in a block. `blkptr` and `blkcols` play the same rows as `ptr` and `cols` in CSR but for non-zero blocks. More specifically, `blkcols` is an array storing the column index of each non-zero block; it has a length equal to the number of non-zero blocks in the matrix. `blkptr` is an array of pointers, with each element as the index of the element in `blkcols` that correspond to the first non-zero block in a row.

When the non-zero elements are clustered together in blocks, BSR shines in storage and performance. This is because blocks of zeros are completely ignored. However, if a block is not completely dense, zeroes are included in the `block_values`. Thus, the format does not ignore all the zeroes in its representation. If there exists a non-zero in every sub-block, then it is inefficient to represent it in a BSR format over a dense format, as it needs additionally store the `block_indices` and `block_ptr`). Under optimal scenarios, sub-blocks can be efficiently parallelized and vectorized, making the algorithm have less stress on the memory system due to cache-friendly and contiguous memory accesses.

Previous works have studied how to select the best sparsity format based on the parameters of the sparse matrix. Some are machine learning based and others are heuristic-based [57, 59]. MKL and ARM PL adopts a heuristic called *inspector-executor*, where they schedule and reformat matrices based on runtime and memory parameters to give the best performance for Sparse BLAS routines [3, 29]. In the inspector-executor mode, the API has an analysis stage where it identifies the matrix structure and reformats. Then in the execution stage, subsequent routine calls will reuse and optimize the data for efficient and optimal execution.

2.2 Memory Contention

On user mobile devices, for efficiency on physical space, the CPU and GPU share the same chip die on the SoC. Thus, they will also share the same memory system. This setup will face memory contention, since the memory bus is shared between resources. Recently, more SoCs with UMA are making it into server side and consumer hardware. For example, Apple has transitioned into Apple Silicon Macs, where their laptops and desktops run on Unified Memory, sharing CPU, GPU, and other hardware accelerators. NVIDIA has also released their NVIDIA H100 Superchip which combines an ARM CPU processor and NVIDIA GPU on a high bandwidth unified memory interconnect. How memory contention can affect performance for practitioners and engineers will be insightful for today and future novel hardware architectures.

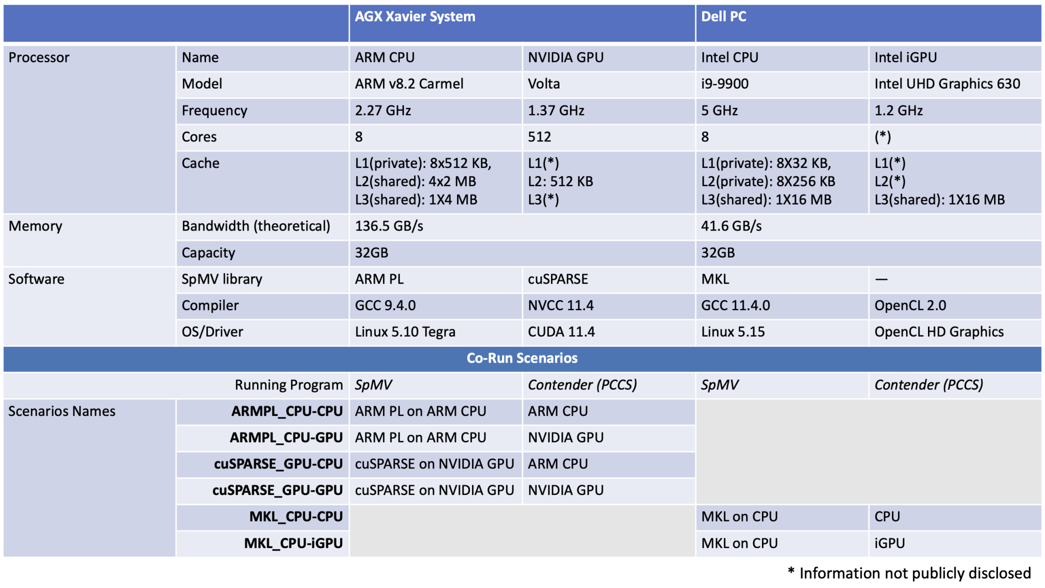
3 RELATED WORK

SpMV Optimizations SpMV is widely known to be highly memory-bound due to its arithmetic intensity [48]. There are many research efforts to analyze and optimize performance bottlenecks of just SpMV itself [11, 22, 42, 47, 53]. Optimizations are not general towards all architectures, and architecture-specific tuning is required for the peak performance of SpMV. Some optimizations are motivated by specific sparse applications, such as deep learning [25]. For example, Gales specifically optimizes sparse tensor algebra for CSR format on GPU to accelerate DL applications that have specific densities and patterns [23].

Some works optimize and automate the sparse tensor algebra optimization through compiler and code generation techniques, such as TACO and Tiramisu [5, 31–33]. AlphaSparse automatically creates novel machine-generated sparse formats to specifically optimize SpMV on a given matrix [19]. Mosaic is an extension of TACO that utilizes vendor libraries to exploit any specialized hardware accelerators that are not accessible to the general public to give the most optimal performance for sparse tensor algebra [6].

With the current state of optimizations and research works in mind, we use vendor libraries from respective hardware backends as they are well-maintained and widely used in academia and industry.

SpMV Format Selection Zhao and others proposed the first DNN-based predictor to statically predict the best sparse format based on the structure of a matrix [57]. Since then, a number of studies have delved into the topic [9, 18, 44, 45, 60]. Some carefully consider the overhead



		AGX Xavier System		Dell PC	
Processor	Name	ARM CPU	NVIDIA GPU	Intel CPU	Intel iGPU
	Model	ARM v8.2 Carmel	Volta	i9-9900	Intel UHD Graphics 630
	Frequency	2.27 GHz	1.37 GHz	5 GHz	1.2 GHz
	Cores	8	512	8	(*)
	Cache	L1(private): 8x512 KB, L2(shared): 4x2 MB L3(shared): 1X4 MB	L1(*) L2: 512 KB L3(*)	L1(private): 8X32 KB, L2(private): 8X256 KB L3(shared): 1X16 MB	L1(*) L2(*) L3(shared): 1X16 MB
Memory	Bandwidth (theoretical)	136.5 GB/s		41.6 GB/s	
	Capacity	32GB		32GB	
Software	SpMV library	ARM PL	cuSPARSE	MKL	—
	Compiler	GCC 9.4.0	NVCC 11.4	GCC 11.4.0	OpenCL 2.0
	OS/Driver	Linux 5.10 Tegra	CUDA 11.4	Linux 5.15	OpenCL HD Graphics
Co-Run Scenarios					
	Running Program	*SpMV*	*Contender (PCCS)*	*SpMV*	*Contender (PCCS)*
Scenarios Names	**ARMPL_CPU-CPU**	ARM PL on ARM CPU	ARM CPU		
	ARMPL_CPU-GPU	ARM PL on ARM CPU	NVIDIA GPU		
	cuSPARSE_GPU-CPU	cuSPARSE on NVIDIA GPU	ARM CPU		
	cuSPARSE_GPU-GPU	cuSPARSE on NVIDIA GPU	NVIDIA GPU		
	MKL_CPU-CPU			MKL on CPU	CPU
	MKL_CPU-iGPU			MKL on CPU	iGPU

* Information not publicly disclosed

Fig. 2. Hardware targets, vendor libraries, and co-run scenarios

during runtime incurred by the predictor as well as format conversion via time-series-based predictors and regression models [58, 59]. More recently, WISE has reached 92% accuracy in a performance prediction model based on sparse matrix attributes and format selection to achieve significant speedups in performance [56]. To the best of our knowledge, no prior work has studied format selection in the presence of co-run contentions, which is especially important for personal, heterogeneous systems.

Memory Contention Many prior performance models have included memory contention into considerations for program performance optimizations [7, 10, 15, 20, 21, 30, 35–37, 39, 41, 43, 50, 51, 61, 62]. Some efforts create performance models to guide hardware designs [2, 27, 49, 63]. Xu and others have proposed a slowdown model named PCCS to accurately predict the performance degradation of applications under unified-memory contention [52]. Gables [27] integrates memory interference into Roofline models. None of the studies aim to uncover the relations between co-run contention and sparse matrix format selection.

PCCS has its own tool to generate synthetic contention kernels, which we utilize in our experiments. PCCS contention kernels are inspired by the Empirical Roofline Toolkit (ERT) that determines the empirical peak of a processor [55]. PCCS re-purposes those kernels into a tunable contender that can be used to generate different lsvels of contention by either strengthening the memory pressure by keeping kernels with low compute intensity and high memory bandwidth, or strengthening the compute pressure by keeping kernels with high compute intensity and low memory bandwidth. Another notable work towards memory contention is Gables, where they incorporate memory interference into the Roofline Model for mobile SoCs [26]. Bubble-Up [38] proposes a methodology that predicts performance degradation from memory contention. Some other work [35] focuses on the interference among DNNs in a multi-tenant deep learning system. None of the studies aim to uncover the relations between co-run contention and sparse matrix format selection.

To our knowledge, there are no prior works that consider contention in real-world applications on user-end devices with unified memory and heterogeneous processors.

4 PRINCIPLES AND CONSTRAINTS

It is worth noting that when the matrix format changes, the SpMV kernel must change as well for its access to the elements in computation would differ. Therefore, in a SpMV library, different matrix formats correspond to a different kernel functions. Format selection naturally leads to the selection of the corresponding kernel function.

One **principle** held in this work is to focus on industrial-quality vendor SpMV libraries, for two reasons. First, efficient SpMV kernels are tricky to develop, requiring effective use of native (vector) instructions, customization to (undisclosed) hardware features, extensive tuning, and so on. Open-source SpMV libraries may perform reasonably well on some matrices, but often fall short on other matrices, trailing in quality and robustness. Second, findings on vendor libraries will directly benefit real-world applications as they are often built on those libraries.

A second **principle** is about the used hardware, which should (i) be real hardware rather than simulators, for accuracy; (ii) have heterogeneous chips with unified memory, for reflecting the trends and common features of mainstream embedded devices; (iii) include multiple kinds, for identifying both general insights and hardware-specific ones.

Focusing on vendor libraries and real hardware give the many important benefits, but also entails some **constraints**. (i) The choices of matrix formats are limited. Even though many formats have been proposed in research papers, those adopted broadly in vendor libraries are only a small portion of them, specifically, COO, CSR, CSC, and BSR formats. It is plausibly due to those formats having shown enough applicability and practical competitiveness over other formats. Our exploration hence focus on only those formats. (ii) Vendor SpMV libraries are mostly closed-source, and profiling tools on embedded system are much less developed than on servers. For instance, although NVIDIA Nsight profiler offers detailed info on server GPUs, the trace it collects on AGX Xavier has almost no details; `nvprof` becomes out of date and incompatible with the recent systems. These constraints make detailed profiling and performance analysis difficult. We give our best-effort analysis, but sometimes have to stop at a level higher than we desire. Despite these constraints, we believe that it is still worth focusing on vendor libraries on real hardware, for the accuracy and practical impact. We next detail our experiment design.

Please note that when the matrix format changes, the SpMV kernel code must change as well. The reason is that the way for the kernel to access the data elements must change when the format of the matrix changes. Therefore, in a SpMV library, different matrix formats correspond to different kernel functions, and format selection naturally leads to the selection of the corresponding kernel function. Format selection and kernel selection are inherently coupled together, and it is impossible to isolate their individual impacts. The differences in kernels (e.g., different vectorizations, register usage, intermediate data organizations) are caused by and also reflections of the format changes. Therefore, the analysis of the impact of formats selection must include the impact of the different kernel implementations, including the vectorizations, register and cache usage, and so on, as the following sections do.

5 EXPERIMENT DESIGN FOR STUDYING THE EFFECTS OF CONTENTION

5.1 Sparse Formats

First, different sparse formats can inherit different performance properties based on the pattern and structure of the sparse matrix itself. The first question we wanted to ask is if memory contention can affect which format representation performs the best or not. We benchmark SpMV with sparse

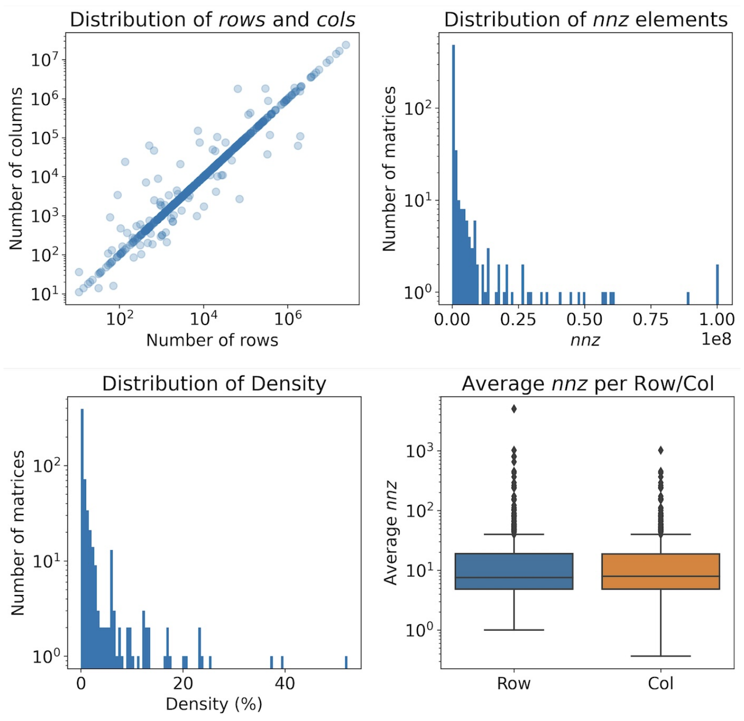


Fig. 3. Distributions of subsampled matrices from SuiteSparse Matrix Collection

formats of COO, CSR, CSC, and BSR with blocksize of 4 (BSR4). These are formats that are supported on all of the vendor sparse libraries that we benchmark on. MKL and ARM PL have an inspector-executor mode for choosing the best format. To measure the performance of each individual format and its respective SpMV algorithm, we directly call the Sparse BLAS API without including calls to `mkl_sparse_optimize` and `armpl_spmv_optimize` that automatically optimize the sparse format [3, 29].

5.2 Dataset

To get a representative dataset of sparse matrices, we evaluate the performance of SpMV on sparse matrices from the SuiteSparse Matrix Collection dataset [17]. This is a SOTA dataset widely used in academia and industry to evaluate the performance of sparse algroithms. The dataset has 2,893 matrices used in a wide field of applications. We sample 25 matrices from each of 24 domains to get a subsample of 600 matrices. We chose a subsample to keep data collection feasible, as we run the benchmarks for 25 trials $\times$ 6 co-run scenarios $\times$ 6 contention levels $\times$ 4 sparse formats $\times$ 600 sparse matrices, totaling over 2 million samples. Our sampling process filters out matrices that can fit under 2GB, as we need some additional space for format conversions, data loading, and contention code. Figure 3 shows the distribution of several features of the subsampled dataset.

5.3 Hardware

Different platforms can have different effects on performance, as not only the computational units and microarchitecture differ, but also the memory system. Contention is especially important on embedded hardware with unified memory, where, processors share the same data path to memory.

Table 1. Effective memory bandwidths per contention level in GB/s

Level	ARM CPU	NVIDIA GPU	Intel CPU
0	0.00	0.00	0.00
1	0.70	20.31	0.58
2	45.78	40.24	16.11
3	84.21	78.99	21.24
4	134.85	147.69	42.33
5	164.20	200.46	63.71

To get a comprehensive analysis, we collect benchmarks from multiple hardware targets. We chose Intel and ARM for CPU and NVIDIA for GPU. Figure 2 shows in detail what hardware systems and their respective software are used to benchmark SpMV. Note that the ARM v8.2 CPU and NVIDIA Volta GPU are both on the same SoC and will run on Unified Memory. We also use an Intel processor, but use the integrated GPU (iGPU) as it shares the L3 cache and main memory with the CPU.

5.4 Co-run Scenarios

Different libraries are utilized to execute SpMV. The libraries are specialized towards one hardware target and are not cross-compatible with others. To show all the configurations and their contenders, Figure 2 outlines the experiments we run with the SpMV libraries under CPU and GPU contention. In total, we have six different configurations: `NVIDIA_GPU-CPU`, `NVIDIA_GPU-GPU`, `MKL_CPU-CPU`, `MKL_CPU-GPU`, `ARMPL_CPU-CPU`, `ARMPL_CPU-GPU`. The code names will be used throughout the rest of the paper to clearly indicate what hardware the SpMV backend runs on and where the contender code runs on.

Co-runners. To systematically observe the effects of different levels of contention on SpMV, it is necessary to have a controllable way to create memory contention by some co-running programs (called *co-runners*). We have explored two strategies for the co-runners. One is to use some other benchmarks (e.g., Rodinia programs [12]), the other is to use a controllable contention generator named PCCS [52]. The PCCS contention generator is inspired by the Empirical Roofline Toolkit (ERT) which is designed for determining the empirical peak performance of a processor [55]. PCCS re-purposes those kernels into a tunable contender that can be used to generate consistent contention of various levels by either strengthening the memory pressure by keeping kernels with low compute intensity and high memory bandwidth, or strengthening the compute pressure by keeping kernels with high compute intensity and low memory bandwidth.

Prior work [52] has shown that on embedded heterogeneous hardware with unified memory (i.e., those used in this study), the impacts from two co-runners on the performance of a program are equivalent if the two different co-runners have the same *memory bandwidth demand* (i.e., the average bytes/sec in a standalone run), regardless of their differences in computing or memory access patterns. The reason is that the main impact from the co-runner is on the shared data path to/from the memory, the contention in which is determined by the memory bandwidth demand. Our experiments on PCCS and Rodinia programs have reproduced their observations. Two properties of PCCS make it especially appealing for studying contention impact: (i) it is easy to precisely control the levels of contention; (ii) the contention level can be maintained throughout the execution of a program. We hence use PCCS as the co-runner to produce various controllable levels of memory contentions in all of our experiments.

Contention levels. For simplicity, we denote the intensities from levels 0 to 5 throughout the rest of the paper. Level 0 is no memory contention; only the SpMV benchmarks run in an isolated setting. Level 1 is when PCCS runs a kernel that is the most compute-bound and reaches the empirical peak of the compute target. This is where PCCS runs as many arithmetic operations as possible within a single memory access; mainly it runs a bunch of repetitive and contiguous `fma` instructions. Level 5 is when PCCS is running a kernel that is the most memory-bound. This is where PCCS runs fewer arithmetic operations per contiguous memory access, and it demands more memory bandwidth. Table 1 shows the empirical numbers of the effective memory bandwidth each contention level consumes under an isolated setting.

5.5 SpMV Libraries and Execution Settings

Figure 2 shows the software support and vendor libraries for Sparse BLAS that were used to give us the best performance on the respective hardware target. Algorithms on how to execute SpMV with various format selections may differ depending on the hardware vendor and how they implemented it, but we ensure that the correctness of the output is the same. For all platforms, core frequencies are pinned to the max.

On CPU, we pin SpMV to run on one core (the first core), while contention code can run on all the other cores. There are two reasons for that choice. (i) The first reason is that in our initial benchmarking of the matrices, we noticed multi-threaded performance did not scale well as we added more threads; in some cases the performance of each matrix worsened for both MKL and ARM PL. For ARM PL on 4 cores, we observed slowdowns of 0.44X, 0.3X, 1.0X respectively for CSR, CSC, and COO compared against single-core performance. The phenomenon is attributed to the memory-bound property of SpMV and multi-thread communication overhead. (ii) The other reason is that MKL SpMV implementations for COO and CSC formats are not multi-threaded. Thus, when simulating format selection, it would be an unfair advantage to CSR and BSR since they will always outperform other formats under multiple threads and it wouldn't give us any interesting analysis between a variety of formats. It is worth noting that SciPy, which relies on sparse vendor libraries, also sticks to single-core performance [46].

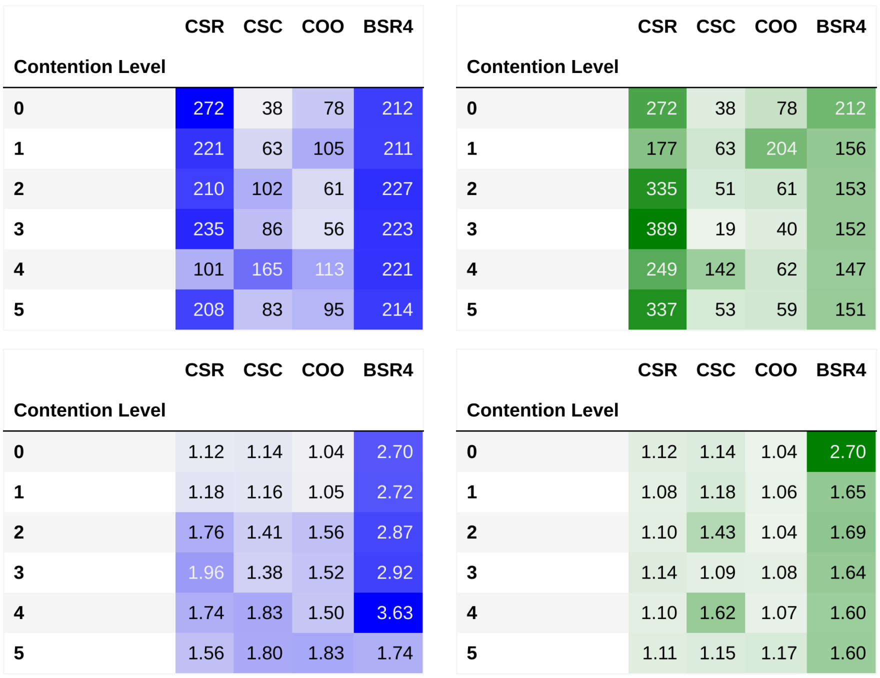
On the Intel system, we make sure to disable hyperthreading. When the SpMV benchmark runs on GPU, there is still a CPU control process, which is also pinned to the first CPU core. For cuSPARSE, there is no option to concurrently run two GPU kernels via two different CUDA Streams. There is also no option to partition the GPU hardware on AGX platform. Thus, we concurrently run the host processes of both GPU programs. This means that the performance of SpMV is time-shared and performance will either: remain roughly the same; drastically drop off due to SpMV process being preempted after a certain time-quanta.

6 OBSERVATIONS AND ANALYSIS

For each backend, we primarily show three different results. The first set of results shows the number of matrices for each format that perform the best out of all the others in each contention level. This gives us an idea of what the distribution of the best format is, and how the favorite formats of the matrices change with the contention levels. For each plot, we use shades of blue to indicate `library-CPU` setting, while shades of green help depict `library-GPU` setting.

The second set of results is an analysis of results where we plot the geometric mean of the speedups over the second-best format for each matrix. This set of results will give us insight into how much a format is more efficient than another sub-optimal format.

The last set of results plots the performance and throughput in GFLOPS grouped by density (geometric mean) to observe any general trends between contention levels.



Contention Level	CSR	CSC	COO	BSR4
0	272	38	78	212
1	221	63	105	211
2	210	102	61	227
3	235	86	56	223
4	101	165	113	221
5	208	83	95	214

Contention Level	CSR	CSC	COO	BSR4
0	272	38	78	212
1	177	63	204	156
2	335	51	61	153
3	389	19	40	152
4	249	142	62	147
5	337	53	59	151

Contention Level	CSR	CSC	COO	BSR4
0	1.12	1.14	1.04	2.70
1	1.18	1.16	1.05	2.72
2	1.76	1.41	1.56	2.87
3	1.96	1.38	1.52	2.92
4	1.74	1.83	1.50	3.63
5	1.56	1.80	1.83	1.74

Contention Level	CSR	CSC	COO	BSR4
0	1.12	1.14	1.04	2.70
1	1.08	1.18	1.06	1.65
2	1.10	1.43	1.04	1.69
3	1.14	1.09	1.08	1.64
4	1.10	1.62	1.07	1.60
5	1.11	1.15	1.17	1.60

Fig. 4. Observations on cuSPARSE. Top left: Matrix counts per best format under CPU contention (cuSPARSE_GPU-CPU); Top right: Matrix counts under GPU contention (cuSPARSE_GPU-GPU); Bottom left: Speedups over the use of second-best format under CPU contention (cuSPARSE_GPU-GPU); Bottom right: Speedups under GPU contention (cuSPARSE_GPU-CPU)

The custom Linux kernel and CUDA drivers used on AGX Xavier were missing access to hardware counters and incompatible with profiling software. Thus, we profiled for a deeper analysis, only if it was possible.

6.1 NVIDIA cuSPARSE

We benchmark SpMV on the NVIDIA AGX Xavier Volta GPU. We use the same GPU for creating contention from the GPU side (`cuSPARSE_GPU-GPU`) and use the ARM CPU on the NVIDIA AGX Xavier for CPU contention (`cuSPARSE_GPU-CPU`).

CPU contention. From Figure 4, we can see that CSR and BSR4 are the most preferable formats in NVIDIA cuSPARSE. In the CPU contention case, as the contention level increases, the distribution of matrices shifts. Many matrices favoring CSR at zero contention shift to other formats when there is contention. The most significant change happens at contention level 4, where a lot of CSC and CSR formats interchange, as they are close to each other in implementation; CSC is the transposed implementation of CSR. On the other hand, the matrices favoring BSR4 mostly stick with that format regardless of the contention; the observation is confirmed by the more detailed distribution shifts shown in Figure 5. The reason is that most matrices favoring BSR4 have a higher density, as seen in the group average performances in Figure 6. Higher density matrices are more likely to have patches of dense sub-blocks, making BSR4 a better option. Because BSR4 runs each patch as a dense matrix, it is more compute-bound than memory-bound for those cases. That explains it is less sensitive to memory contention than other formats.

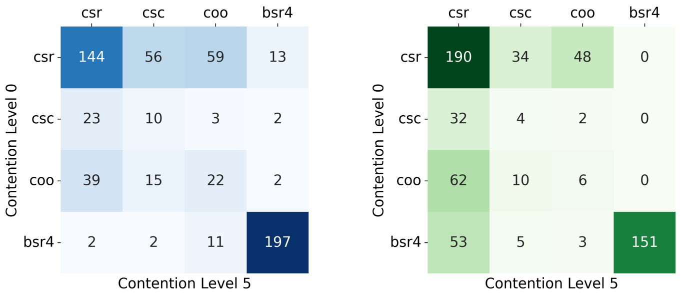


Fig. 5. Distribution shifts for best sparse format for cuSparse from no contention to contention level 5; left: cuSPARSE_GPU-CPU, right: cuSPARSE_GPU-GPU.

Overall, 40% matrices have changes in their best formats under memory contention. From Figure 4, we can see that in most cases, the average performance gap between the use of the best format and the second best format becomes significantly more pronounced in the presence of contention. It grows from 1.12× to 1.96× for CSR, 1.14× to 1.83× for CSC, 1.04× to 1.83× for COO, and 2.70× to 3.63× for BSR4. That indicates the more importance for format selection when there is contention. The plots are in log-scale, so there is a very significant drop off after level 1, and then it plateaus for all formats. This shows that memory contention has high sensitivity of performance degradation between levels 0-2.

GPU contention. In the GPU contention case, CSR and BSR4 are also the most preferred formats. Shifts in the best format caused by contention are even more substantial than in the CPU contention case. Unlike the CPU contention case, many matrices favoring BSR4 at no contention also see changes in their best formats when there is contention. It is because in the GPU case, the contender program and SpMV time-share GPUs, which affects the availability of computing resource for SpMV. The execution time of some matrices, in the presence of contention, exceed the preemption timeframe; they show 59-63% slowdowns on average.

The performance gap between the best format and the second best format are not as pronounced as in the CPU contention case, but are still substantial, especially for the CSC and BSR4 formats, ranging from 1.14-1.62× and 1.60-2.70×.

Overall, the significant trends and shifts in both the CPU contention and GPU contention cases show that for cuSPARSE, the best sparse format perceived in a contention oblivious setting is not truly the best in a contention-aware setting, with 38% and 42% of matrices interchanging between different formats under CPU and GPU contention respectively.

6.2 Intel Math Kernel Library

For this setup, we use the Intel i9 processor to run SpMV benchmarks from MKL Sparse library. We use the same CPU for CPU contention (`MKL_CPU-CPU`) and iGPU for GPU contention (`MKL_CPU-GPU`).

Figure 7 shows the observations. MKL is observed to be more heavily tuned for CSR, where 67-75% of the matrices prefer it under various contention levels. This is evident when sampling counts for vector instructions, where we observed that COO utilizes no AVX2 arithmetic instructions, while CSC, CSR, and BSR4 did. For a small subset of matrices during profiling, we observed CSR uses 1.8× less vector instructions than CSC and 35.9× less for BSR4. When contention level is high (level-5), more matrices start preferring CSC and BSR4 formats. A closer look shows that 62% of the matrices remain in CSR as the best format as contention increases from level 0 to 5. 58 matrices changed from CSR to CSC, while 19 changed from CSC to CSR. Speedups jumps 1.4× more for

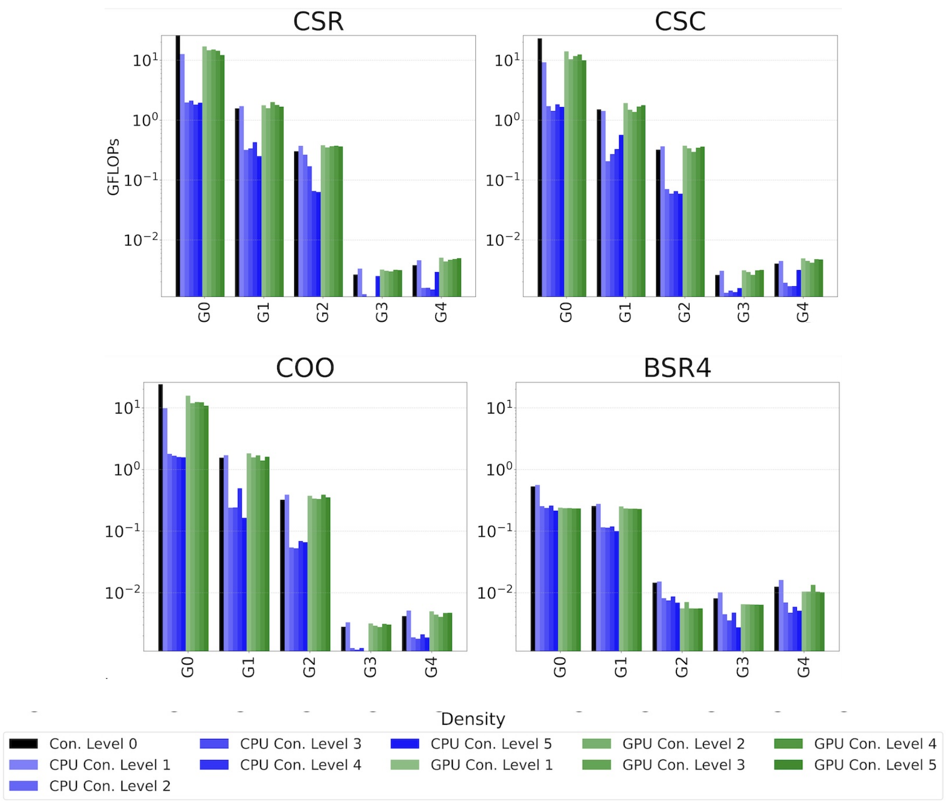


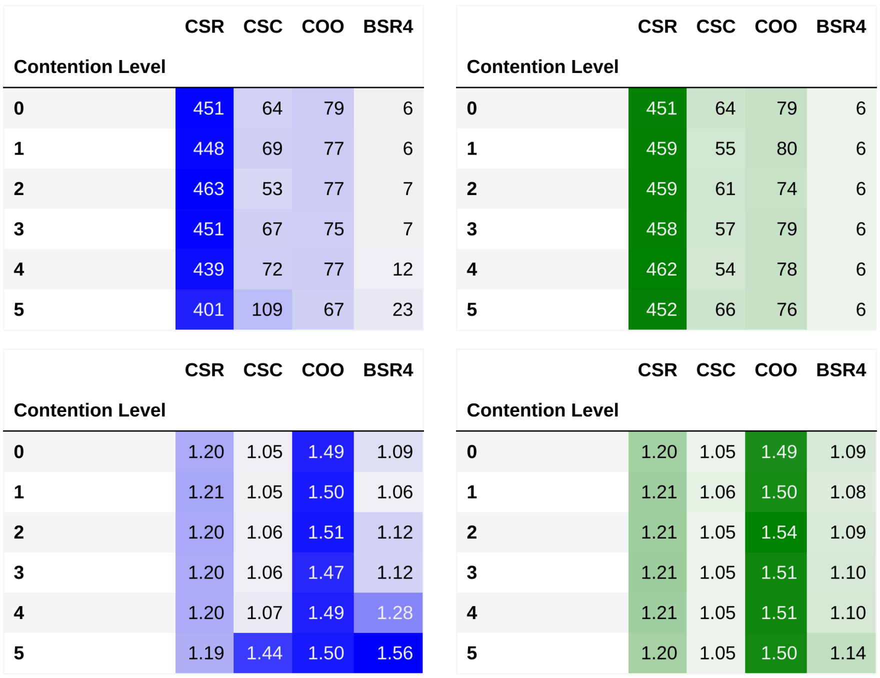
Fig. 6. cuSPARSE performance benchmarks grouped by density in log scale. Density bins: G0 (0%, 10%]; G0 (0%, 10%]; G1 (10%, 21%]; G2 (21%, 31%]; G3 (31%, 42%]; G4 (42%, 52%]

CSC, as 58 matrices from CSR and 6 matrices from COO switch to CSC. The interchange between CSR and CSC are the most common, and that is due to the algorithms being similar, as previously observed with cuSPARSE.

The speedups from CSC and BSR4 formats over other formats become larger as contention level increases. From level 0 to level 5, the speedups increase from 1.05× to 1.44× (CSC) and 1.09× to 1.56× (BSR4), indicating the even higher importance for format selection in the presence of contention. On the other hand, GPU contention (`MKL-CPU_GPU`) has little to no effect to MKL, mainly because the iGPU of the Intel processor is very weak.

Figure 8 shows the average group performances of the matrices grouped by matrix density. CPU contention degrades the performance of SpMV algorithms significantly across the board. But less on COO than on other formats. We attributes it to the fact that COO in MKL uses no vector arithmetic instructions, meaning that the implementation does not require higher bandwidth to deliver data (utilization of vector makes memory bandwidth the bottleneck).

Compared against cuSPARSE, the trends of SpMV algorithms on CPU differ. On average, CSR is the best performing for lower densities, such as G0, G1, and G2. However, COO performs better for higher densities like G3 and G4. The only difference between COO and CSR is that CSR does $2D$ loop iteration over indirect memory locations on the compressed row, while COO just does $1D$ loop over the whole matrix. Considering that COO only uses scalar arithmetic instructions,



Contention Level	CSR	CSC	COO	BSR4
0	451	64	79	6
1	448	69	77	6
2	463	53	77	7
3	451	67	75	7
4	439	72	77	12
5	401	109	67	23

Contention Level	CSR	CSC	COO	BSR4
0	451	64	79	6
1	459	55	80	6
2	459	61	74	6
3	458	57	79	6
4	462	54	78	6
5	452	66	76	6

Contention Level	CSR	CSC	COO	BSR4
0	1.20	1.05	1.49	1.09
1	1.21	1.05	1.50	1.06
2	1.20	1.06	1.51	1.12
3	1.20	1.06	1.47	1.12
4	1.20	1.07	1.49	1.28
5	1.19	1.44	1.50	1.56

Contention Level	CSR	CSC	COO	BSR4
0	1.20	1.05	1.49	1.09
1	1.21	1.06	1.50	1.08
2	1.21	1.05	1.54	1.09
3	1.21	1.05	1.51	1.10
4	1.21	1.05	1.51	1.10
5	1.20	1.05	1.50	1.14

Fig. 7. Observations on Intel MKL. Top left: Matrix counts per best format under CPU contention (MKL_CPU-CPU); Top right: Matrix counts under GPU contention (MKL_CPU-GPU); Bottom left: Speedups over second-best format under CPU contention (MKL_CPU-CPU); Bottom right: Speedups under GPU contention (MKL_CPU-GPU))

for larger densities, CSR suffers due to indirect accesses and packing values into vectors. And this performance degrades even more for CSR than COO as memory contention level increases.

To see the effects of the memory system at a fine-grain level, we have also profiled the memory accesses on the Intel platform using the hardware performance counters available via `perf`. Figure 9 shows the miss rate of the caches. We make sure to only collect hardware counter results on the pinned core that SpMV benchmark is running on to see the effect of memory contention impacting SpMV coming from other cores. We chose the `TSOPF_RS_b162_c3.mtx` matrix as it is a representative case showing significant differences in performances when interchanging from CSR to CSC.

From the benchmarks, CSR performance is the best at 261 μs milliseconds on average under no contention, outperforming CSC format with 346 μs. But under the highest level of contention, CSC format outperforms CSR with 1.26 milliseconds while CSR itself gets 2.12 milliseconds.

We sampled hardware performance counters via `perf` to see how SpMV cache miss rates were affected under contention. The Intel CPU has private L1 and L2 cache and a shared L3 cache. We see that the L1 miss rate stays constant, and this behavior is reasonable due to the cache being private. And the L3 cache miss rate increases when contention increases since the other cores which are running the contending code will compete for the memory bus and resources. However, the L2 cache rate also increases, even though it is private. We expected for it to stay constant even under contention, similar to the behavior of L1 miss rate trends. However, the cache microarchitecture has additional features, such as a line fill buffer connection with L3 cache, that affects the L2 miss rates.

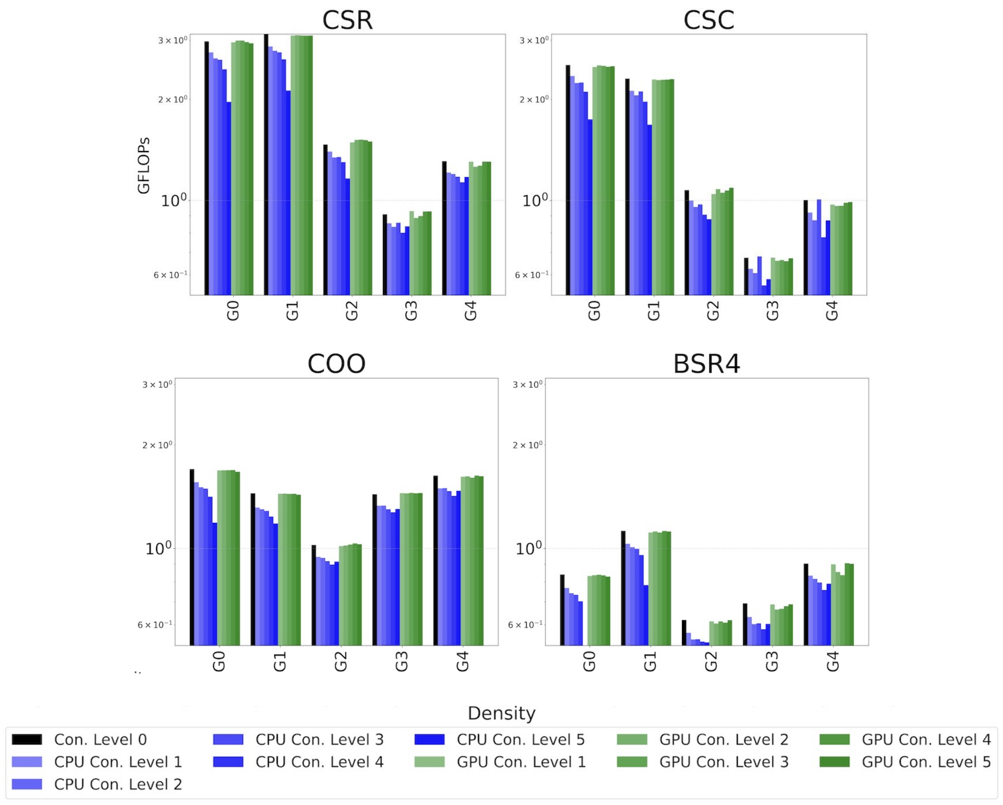
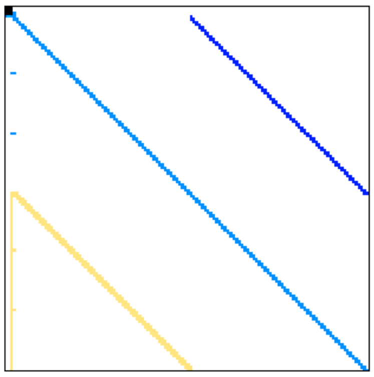


Fig. 8. MKL Sparse performance benchmarks grouped by density in log scale. Density bins: G0 (0%, 10%]; G0 (0%, 10%]; G1 (10%, 21%]; G2 (21%, 31%]; G3 (31%, 42%]; G4 (42%, 52%]



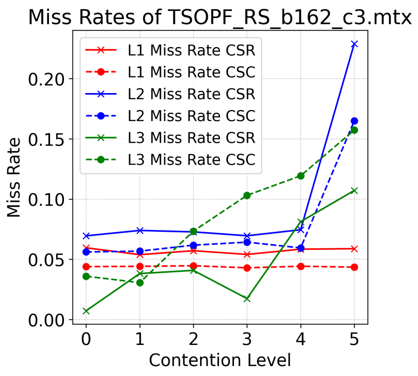


Fig. 9. Left: Spyplot of TSOPF_RS_b162_c3.mtx; Right: Cache miss rates of SpMV on that matrix on Intel CPU when contention comes from the integrated GPU.

How this correlates to the structure is interesting, as the matrix appears to be highly symmetric, where CSC and CSR are close in performance. However, there is a small patch of dense columns, shown in yellow, where CSC actually could be marginally preferable. When memory contention is

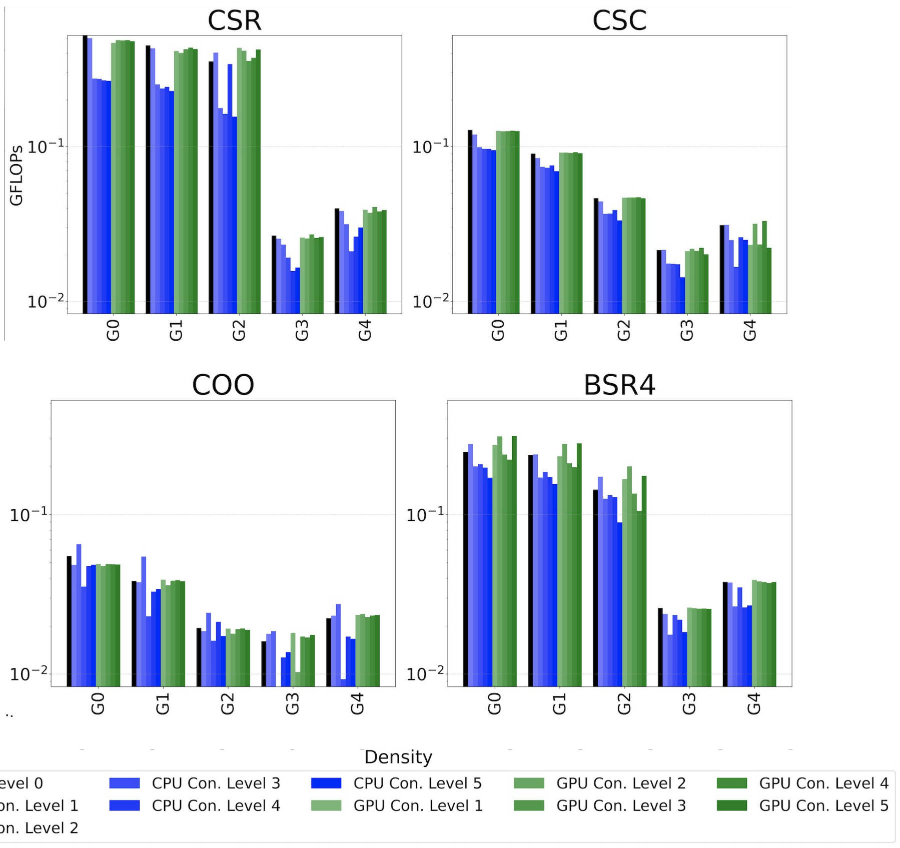


Fig. 10. ARM PL Sparse performance benchmarks grouped by density in log scale. Density bins: G0 (0%, 10%]; G0 (0%, 10%]; G1 (10%, 21%]; G2 (21%, 31%]; G3 (31%, 42%]; G4 (42%, 52%]

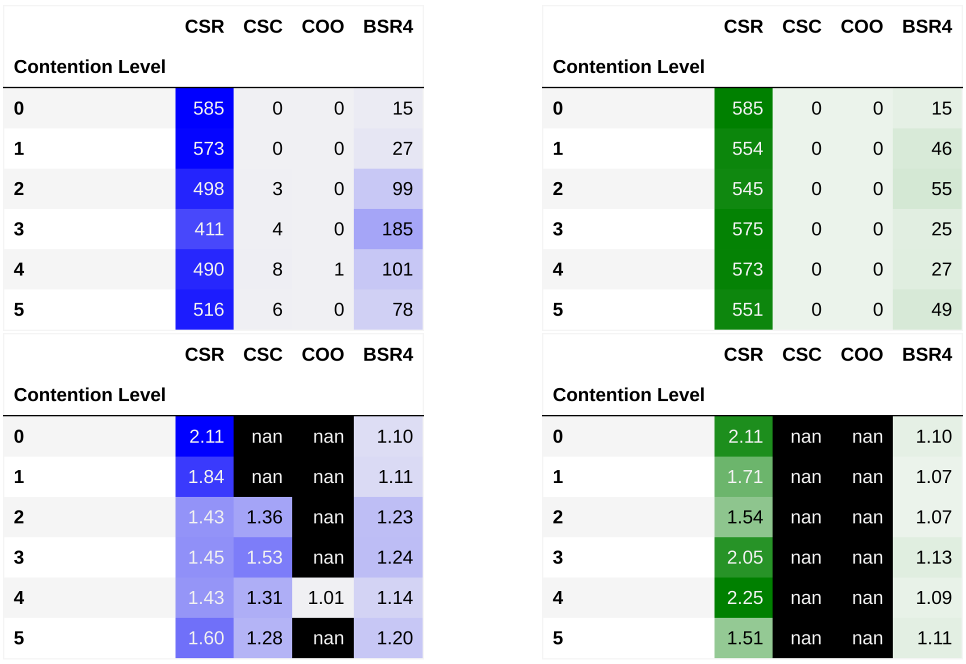
applied, the magnitude of performance degradation is higher by 68% in CSR than CSC. Most of the interchanges between CSR and CSC for MKL are in this performance trend.

6.3 ARM Perforance Library

We use the ARM v8 CPU to run SpMV benchmarks from ARM PL library. The same CPU is used for CPU contention (`ARMPL_CPU-CPU`) and NVIDIA AGX Xavier Volta GPU for GPU contention (`ARMPL_CPU-GPU`).

Figure 11 indicates that the CSR format has been heavily tuned and optimized in ARM PL, much akin to MKL. The best format mostly interchanges between CSR and BSR4, with more matrices that have CSR as the best format switch over to BSR4. A similar observation from cuSPARSE was that compute efficient formats can excel as contention increases.

Figure 10 shows the average group performances of the matrices. The degradation of performance due to CPU contention is significant across the board. On COO, some groups show even higher performance in the presence of contention. We attribute that exception to memory clock frequency. The hardware does not allow control of memory clock frequency. So memory frequency could be slightly boosted when co-running tasks are demanding high memory bandwidth and the effects are pronounced for some matrices in those groups.



Contention Level	CSR	CSC	COO	BSR4
0	585	0	0	15
1	573	0	0	27
2	498	3	0	99
3	411	4	0	185
4	490	8	1	101
5	516	6	0	78

Contention Level	CSR	CSC	COO	BSR4
0	585	0	0	15
1	554	0	0	46
2	545	0	0	55
3	575	0	0	25
4	573	0	0	27
5	551	0	0	49

Contention Level	CSR	CSC	COO	BSR4
0	2.11	nan	nan	1.10
1	1.84	nan	nan	1.11
2	1.43	1.36	nan	1.23
3	1.45	1.53	nan	1.24
4	1.43	1.31	1.01	1.14
5	1.60	1.28	nan	1.20

Contention Level	CSR	CSC	COO	BSR4
0	2.11	nan	nan	1.10
1	1.71	nan	nan	1.07
2	1.54	nan	nan	1.07
3	2.05	nan	nan	1.13
4	2.25	nan	nan	1.09
5	1.51	nan	nan	1.11

Fig. 11. Observations on ARM PL. Top left: Matrix counts per best format under CPU contention (ARMPL_CPU-CPU); Top right: Matrix counts under GPU contention (ARMPL-CPU-GPU); Bottom left: Speedups under CPU contention (ARMPL_CPU-CPU); Bottom right: Speedups under GPU contention (ARMPL_CPU-GPU)). When no matrix has CSC or COO as the best format, speedups are denoted as nan

GPU contention seems to affect the performance minimally with little to no performance degradation. This is most likely due to the memory bus not competing for a lot of bandwidth as the throughput in GFLOPS for ARM PL's SpMV are low to begin with (6× slower than MKL).

We could not profile the CPU either for these settings, as mentioned before. Instead, we manually did an `objdump` of the ARM PL library, and observed whether microkernels had vector instructions or not (the symbols were descriptive enough to pinpoint which microkernels we utilized). We found that only CSR and BSR4 microkernels have ARM NEON instructions (primarily indicated by `fmla.4s` and any other vector instructions that correlate to $y = \alpha Ax + \beta y$), while COO and CSC only use scalar registers and instructions. These results are consistent with the distributions and speedups. Optimized implementations via vectorization are more sensitive to memory contention, as seen with CSR at density groups G0-G2. Similar to cuSPARSE, there is a significant dropoff of performance degradation for some formats after level 1. This is also due to the sensitivity of memroy contention at lower levels seen at CSR G0, G1, and G2.

7 CONTENTION-AWARE SPARSE FORMAT SELECTION

The results indicate some clear impact of contention on best format selection. This section discusses how to leverage it for improving SpMV format selection and what the potential is. To that end, we implement a concrete contention-aware sparse format predictor and compare it with a contention-oblivious counterpart. We further study the general potential of being contention-aware across a set of abstract predictors of various accuracy. We in addition discuss the overhead and strategies to

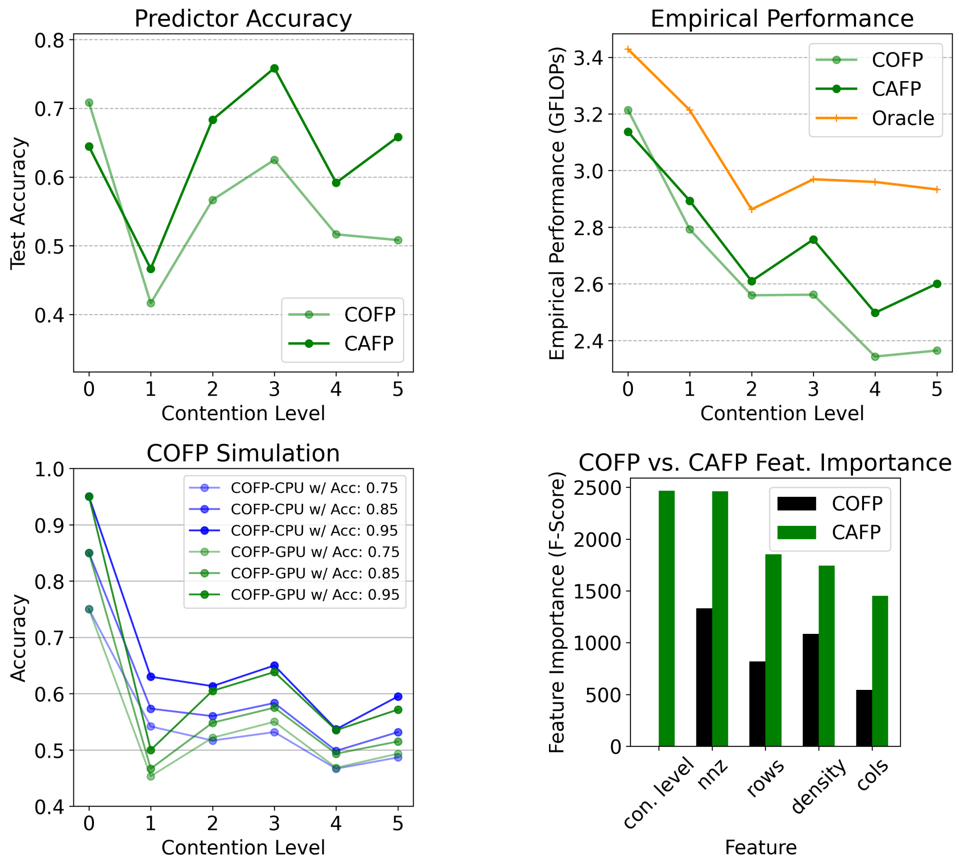


Fig. 12. Top left: Accuracy of our CAFP and COFP for cuSPARSE-GPU_GPU. Top right: Geometric mean of empirical performances of COFP, CAFP, Oracle. Bottom left: Our simulated concrete predictor COFP. Bottom right: Feature Importance of COFP and CAFP.

deal with it. We denote COFP as Contention Oblivious Format Predictor and CAFP as Contention Aware Format Predictor throughout the rest of this section.

7.1 Comparison on Predictors

For comparison, we used gradient-boosted decision tree (XGBoost) [13] to construct two concrete format predictors. XGBoost rather than more popular Deep Neural Networks (DNN) was chosen for two reasons. (i) XGBoost had proven effective in outperforming some DNN-based predictors on modest-sized datasets [58, 59]; (ii) It is agile, incurring much less runtime prediction overhead, which makes it a preferred choice than the slow DNNs when the prediction needs to happen at runtime [58, 59]. The prediction by a DNN can be as much as tens of iterations of SpMV while XGBoost takes only a range of 1-22% of SpMV. Nevertheless, our goal here is not to build the most accurate models, but to see the potential of being contention-aware for format prediction. The insight gained from our analysis can be transferable to other settings as well.

Our models are built on the data collected for the `cuSPARSE-GPU_GPU` setting on the 600 matrices in the SuiteSparse Matrix Collection dataset as described in Section 5.2. The hardware is the

AGX Xavier machine shown in Figure 2. The co-runners are the PCCS contention generator which generates contentions at a range of levels (Sec 5.4). Data are partitioned in 8:2 for training and testing. We created two tabular datasets. One that is contention-aware, and one that is contention-oblivious. Both datasets share features, including number of rows, cols, and *nnz*. The contention-aware dataset is the same except that it contains the contention level as an additional feature. The labels are the best sparse format. The training process and hyper-parameters of the two models are identical. The design ensures that the only difference between the two models is the inclusion of the contention level as an input factor. It is worth noting that for both contention-oblivious and contention-aware predictors, we considered more sophisticated features (e.g., symmetry, kind, group, and structure of the matrices). We didn't use them in the final predictors because getting those features would add substantial run-time overhead, outweighing the extra benefits they could bring. We try to keep the prediction agile.

Figure 12 reports a series of metrics that report the the models' accuracy and empirical performance when choosing the models' prediction, as well as a comparison against a simulated static predictor. The accuracy of COFP is about 71% when there is no contention, but drops sharply to 42–62% in the presence of contention. The regrets are substantially reduced by the CAFP; at some contention levels (e.g., 76% at level-3), the prediction accuracy is even higher than the start accuracy (0-contention accuracy). The higher accuracy translates to better performance of SpMV, as shown in the second graph. Also included is what a 100% accurate predictor performance (Oracle) would get, that is, the full potential of a CAFP, leaving up to 9–25% performance improvement.

The third graph in Figure 12 shows the accuracies of abstract COFPs at several extra levels of zero-contention accuracies: 75, 85, and 95%. They are artificially set to make accurate predictions to exactly that portion of matrices at zero-contention without changing their predictions with contentions. It shows that using more accurate contention-oblivious predictors still suffers significant drop in accuracy due to large number of format exchanges at higher contention levels. The last graph shows the models' feature importance (the higher, the more important a feature is). In COFP model, the *nnz* has the highest importance. However, when CAFP model is trained, contention level has the highest importance alongside *nnz*, confirming its usefulness.

7.2 Overhead and Deployment Strategy

This section discusses factors to consider when implementing and deploying a CAFP on user devices. As in prior studies[9, 57–59], the discussion assumes SpMV is invoked repeatedly on the same sparse matrix (but different vectors), a typical scenario in scientific simulation and deep learning (e.g., weight matrix).

Overhead is a main factor to consider, in both time and space. Like in the contention-oblivious case, there are three sources of time overhead: (i) attaining the features of a matrix, (ii) making format prediction, and (iii) converting the matrix format.

The first one is a one-time operation. But for contention-aware predictors, it would also include the overhead in getting contention level periodically, which can be done quickly by reading performance counters. The hardware performance counters to read to find out the memory throughput are specific to processors; one needs to use the appropriate counters when implementing the contention-aware prediction. On Intel processors, for instance, the relevant counters are as follows: LLC_MISSES: Counts last level cache (L3) misses; UNC_M_RPQ_INSERTS: Counts the number of read requests inserted into the read queue of the memory controller; UNC_M_WPQ_INSERTS: Counts the number of write requests inserted into the write queue of the memory controller. Hardware performance counters are special registers; the time taken to read them is negligible. Even if one reads the counters as often as once every SpMV call, the time overhead is still negligible (one or several register readings versus executions of tens of thousands of instructions). The time overhead of (i) is hence negligible.

The second source of overhead depends on the predictor; it is marginal for lightweight models: The XGBoost (which resides on the CPU) used in our work takes 8 μs to do inference on one sample, which is only a small fraction of one SpMV run on non-trivial matrices.

The third source of overhead is the most interesting one. Format conversion takes non-trivial time compared to SpMV time. In the contention-oblivious case, the conversion can be done once for a given matrix before execution. But because the contention level may change during execution, to adapt to the contention levels, the matrix may need to be repeatedly switched to other formats during an execution; the overhead hence could be a runtime concern.

The appropriate strategy to treat overhead depends on the context. If the contention level is stable in the system, one-time prediction and format creation would be sufficient, no much difference from the deployment of contention-oblivious predictors.

When the contention may change in an execution, we see three possible schemes. (i) Scheme-1: It keeps only one copy of the sparse matrix in the most recently used format. It invokes the predictor and creates the needed format whenever the contention level shows a substantial shift. (ii) Scheme-2: This scheme keeps a copy of the matrix in each of the formats, calls the predictor and switches to the predicted format at a contention level shift. (iii) Scheme-3: This scheme is a middle ground. It saves the most likely to be used (multiple) formats, and creates other formats when necessary.

Scheme-1 incurs the largest time overhead but no extra space overhead. Our measurement shows that on Xavier GPU, it takes between 0.25 and 2.8 ms (562 μs on average) to convert a matrix from CSR to BSR4, which ranges from 394-702% conversion overhead for the corresponding calls to SpMV in BSR4 format. The implication is that the times for adapting the formats need to be well spaced out to control the overhead. Making the format conversion asynchronous could help—that is, converting the format in the background while SpMV-application keeps running.

Scheme-2 incurs negligible time overhead but the largest space overhead. We analyze the space overhead of the four formats as follows. COO has an upper bound of $O(3nnz)$ space complexity, while CSR and CSC share an upper bound of $O(2nnz + m)$ (m is the row# or column#). For BSR, it is highly dependent on the block size and structure of the sparse matrix which is $O(n_{blocks} nnz_{blocks} + m)$, where n_{blocks} is the number of blocks and nnz_{blocks} is the number of nnz within a block. If a matrix cannot be efficiently packed into dense patches of blocks, then extra 0s will be stored, making this overhead potentially higher than COO, CSR, and CSC. This scheme fits the case where space is not an issue but the contention level changes frequently.

Scheme-3 offers a tradeoff between the first two schemes. It could be made elastic by offering knobs to tune the frequency in adapting formats and the number of formats to save. Like Scheme-1, it could also use asynchronous format conversion to hide some time overhead. We leave empirical explorations of the schemes to future work.

8 SUMMARY OF INSIGHTS

This paper presents the first-known comprehensive analysis of SpMV performance in different storage formats in the presence of various levels of memory contention, and provides a set of insights on the significance of co-run contention on sparse format selection:

(1) The degree of the impact of contention on format selection varies across libraries and architectures. For some (e.g., ARM PL), contention-oblivious selection is sufficient, but for others, it is not. For the latter, contention-oblivious selection may result in significant (as much as 48%) performance degradation.

(2) Libraries belonging to the former category tend to have some of the following properties: (i) SpMV has a relatively low throughput even in stand-alone runs. (ii) If one format dominates in contention-free scenarios, it is likely to stay dominating in the co-runs. (iii) The contention program runs on a weak co-processor.

(3) On CPU, CSR tends to work well, but in the cases when COO works better (e.g., G3 & G4 in Figure 8), the speedups COO brings tend to be significant.

(4) On embedded GPUs, the format preferences tend to show some substantial diversity among matrices. As contention increases, many matrices switch their preferred formats.

(5) With contention, the performance gaps between the best format and the other formats become even more pronounced; format selection is more important.

(6) The higher the performance and memory bandwidth a specific format for SpMV algorithm demands (e.g., due to indexing or vector instructions), the higher the degradation of performance can be under high contention.

(7) The environment of where contention happening can affect performance.

(8) When dealing with contention in sparse format selection, it is necessary to discern which category the library and architecture belongs to. Simple treatment (e.g., simply using the dominating format in the contention-free case) could be sufficient for some on an architecture. As that can avoid format prediction and conversion, it can be appealing for production use. For other cases, contention-aware format selection is necessary.

(9) Contention-aware format selection can offer substantial performance gains over contention-oblivious selection on user devices.

(10) Because the best formats of a matrix may change with contention, if the contention changes, the format selector may need to switch the format of a matrix. It is a space-time tradeoff and can be optimized based on the context (e.g., through the three schemes in the previous section).

(11) On CPUs, there is a largely monotonic relation between contention degree and SpMV performance degradation, which suggests the potential for contention-aware performance prediction, useful for runtime task scheduling. The complicated relation on embedded GPUs suggest the difficulties for such predictions, but positive effects of contention on such GPUs in certain contexts could offer some novel opportunities for the scheduler to take advantage in scheduling tasks.

(12) The design of format selection shall be cautious of the performance cliffs of some matrices in the presence of certain levels of contention, switching the format when possible at the cliffs;

(13) On systems with adaptive memory clock rate, the format selector should take into consideration the impact of the memory clock rate adaptation.

9 CONCLUSIONS

This paper presents the first-known comprehensive analysis of SpMV performance in different storage formats in the presence of various levels of memory contention, and provides a set of insights on the significance of co-run contention on sparse format selection, the influence of various factors, and the promise of co-run aware sparse format prediction. These findings and insights offer a deeper understanding of the performance and optimizations of sparse computations on heterogeneous consumer devices. They could help enhance the development of sparse libraries, optimization techniques, and runtime systems, to better meet the increasing demands of high performance of sparse computations in deep learning and other emerging domains.

REFERENCES

[1] The future of next-gen ai smartphones.

[2] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[3] ARM. Sparse linear algebra introduction, 2023.

[4] Krste Asanović, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[5] Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Del Sozzo, Abdurrahman Akkas, Yunming Zhang, Patricia Suriana, Shoaib Kamil, and Saman P. Amarasinghe. Tiramisu: A polyhedral compiler for expressing fast and portable code. *CoRR*, abs/1804.10694, 2018.

[6] Manya Bansal, Olivia Hsu, Kunle Olukotun, and Fredrik Kjolstad. Mosaic: An interoperable compiler for tensor algebra. *Proc. ACM Program. Lang.*, 7(PLDI), jun 2023.

[7] Rajkishore Barik, Naila Farooqui, Brian T Lewis, Chunling Hu, and Tatiana Shpeisman. A black-box approach to energy-aware scheduling on integrated cpu-gpu systems. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, pages 70–81. ACM, 2016.

[8] Akrem Benatia, Weixing Ji, Yizhuo Wang, and Feng Shi. Sparse matrix format selection with multiclass svm for spmv on gpu. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 496–505, 2016.

[9] Akrem Benatia, Weixing Ji, Yizhuo Wang, and Feng Shi. Bestsf: A sparse meta-format for optimizing spmv on gpu. *ACM Trans. Archit. Code Optim.*, 15(3), sep 2018.

[10] David Black-Schaffer, Nikos Nikoleris, Erik Hagersten, and David Eklov. Bandwidth bandit: Quantitative characterization of memory contention. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 1–10. IEEE Computer Society, 2013.

[11] Aydin Buluç, Samuel Williams, Leonid Oliker, and James Demmel. Reduced-bandwidth multithreaded algorithms for sparse matrix-vector multiplication. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 721–733, 2011.

[12] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)*, pages 44–54. Ieee, 2009.

[13] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, August 2016.

[14] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019.

[15] Younghyun Cho, Florian Negele, Seohong Park, Bernhard Egger, and Thomas R Gross. On-the-fly workload partitioning for integrated cpu/gpu architectures. In *PACT*, pages 21–1, 2018.

[16] Phil Colella. Defining software requirements for scientific computing, 2004.

[17] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), dec 2011.

[18] Sunidhi Dhandhania, Akshay Deodhar, Konstantin Pogorelov, Swarnendu Biswas, and Johannes Langguth. Explaining the performance of supervised and semi-supervised methods for automated sparse matrix format selection. In *50th International Conference on Parallel Processing Workshop*, ICPP Workshops '21, New York, NY, USA, 2021. Association for Computing Machinery.

[19] Zhen Du, Jiajia Li, Yinshan Wang, Xueqi Li, Guangming Tan, and Ninghui Sun. Alphasparse: Generating high performance spmv codes directly from sparse matrices. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22. IEEE Press, 2022.

[20] Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N Patt. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. *ACM Sigplan Notices*, 45(3):335–346, 2010.

[21] Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N Patt. Fairness via source throttling: A configurable and high-performance fairness substrate for multicore memory systems. *ACM Transactions on Computer Systems (TOCS)*, 30(2):7, 2012.

[22] Goran Flegar and Hartwig Anzt. Overcoming load imbalance for irregular sparse matrices. In *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms*, IA3'17, New York, NY, USA, 2017. Association for Computing Machinery.

[23] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2020.

[24] Scott Gray, Alec Radford, and Diederik P. Kingma. Gpu kernels for block-sparse weights. 2017.

[25] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[26] Mark Hill and Vijay Janapa Reddi. Gables: A roofline model for mobile socs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 317–330, 2019.

[27] Mark Hill and Vijay Janapa Reddi. Gables: A roofline model for mobile socs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 317–330. IEEE, 2019.

[28] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(1), jan 2021.

[29] Intel. Inspector-executor sparse blas routines, 2023.

[30] Min Kyu Jeong, Mattan Erez, Chander Sudanthi, and Nigel Paver. A qos-aware memory controller for dynamically balancing gpu and cpu bandwidth use in an mpsoc. In *Proceedings of the 49th Annual Design Automation Conference*, pages 850–855. ACM, 2012.

[31] Fredrik Kjolstad. *Sparse Tensor Algebra Compilation*. Ph.d. thesis, Massachusetts Institute of Technology, Cambridge, MA, Feb 2020.

[32] Fredrik Kjolstad, Stephen Chou, David Lugato, Shoaib Kamil, and Saman Amarasinghe. taco: A tool to generate tensor algebra kernels. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 943–948, Oct 2017.

[33] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. The tensor algebra compiler. *Proc. ACM Program. Lang.*, 1(OOPSLA):77:1–77:29, October 2017.

[34] Zhiyao Li, Jiaxiang Li, Taijie Chen, Dimin Niu, Hongzhong Zheng, Yuan Xie, and Mingyu Gao. Spada: Accelerating sparse matrix multiplication with adaptive dataflow. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 747–761, New York, NY, USA, 2023. Association for Computing Machinery.

[35] Zihan Liu, Jingwen Leng, Zhihui Zhang, Quan Chen, Chao Li, and Minyi Guo. Veltair: Towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 388–401, New York, NY, USA, 2022. Association for Computing Machinery.

[36] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 450–462. ACM, 2015.

[37] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pages 248–259. ACM, 2011.

[38] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 248–259, 2011.

[39] Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. Contention aware execution: online contention detection and response. In *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, pages 257–265. ACM, 2010.

[40] Timothy G. Mattson, Carl Yang, Scott McMillan, Aydin Buluç, and José E. Moreira. Graphblas c api: Ideas for future versions of the specification. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2017.

[41] Nikita Mishra, John D Lafferty, and Henry Hoffmann. Esp: A machine learning approach to predicting application interference. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 125–134. IEEE, 2017.

[42] Bor-Yiing Su and Kurt Keutzer. Clspmv: A cross-platform opencl spmv framework on gpus. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, page 353–364, New York, NY, USA, 2012. Association for Computing Machinery.

[43] Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu. The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 62–75. IEEE, 2015.

[44] Qingxiao Sun, Yi Liu, Ming Dun, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. Sptfs: Sparse tensor format selection for mttkrp via deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2020.

[45] Qingxiao Sun, Yi Liu, Hailong Yang, Ming Dun, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. Input-aware sparse tensor storage format selection for optimizing mttkrp. *IEEE Transactions on Computers*, 71(8):1968–1981, 2022.

[46] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero,

Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[47] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pages 1–12, 2007.

[48] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, apr 2009.

[49] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[50] Yuejian Xie and Gabriel Loh. Dynamic classification of program memory behaviors in cmps. In *the 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2008.

[51] Shizhen Xu, Yuanchao Xu, Wei Xue, Xipeng Shen, Fang Zheng, Xiaomeng Huang, and Guangwen Yang. Taming the" monster": Overcoming program optimization challenges on sw26010 through precise performance modeling. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 763–773. IEEE, 2018.

[52] Yuanchao Xu, Mehmet Esat Belviranli, Xipeng Shen, and Jeffrey Vetter. Pccs: Processor-centric contention-aware slowdown model for heterogeneous system-on-chips. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 1282–1295, New York, NY, USA, 2021. Association for Computing Machinery.

[53] Shengen Yan, Chao Li, Yunquan Zhang, and Huiyang Zhou. Yaspmv: Yet another spmv framework on gpus. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '14, page 107–118, New York, NY, USA, 2014. Association for Computing Machinery.

[54] Carl Yang, Aydın Buluç, and John D. Owens. Graphblast: A high-performance linear algebra-based graph framework on the gpu. *ACM Trans. Math. Softw.*, 48(1), feb 2022.

[55] Charlee Yang. Cs roofline toolkit, 2020.

[56] Serif Yesil, Azin Heidarshenas, Adam Morrison, and Josep Torrellas. Wise: Predicting the performance of sparse matrix vector multiplication with machine learning. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '23, page 329–341, New York, NY, USA, 2023. Association for Computing Machinery.

[57] Yue Zhao, Jiajia Li, Chunhua Liao, and Xipeng Shen. Bridging the gap between deep learning and sparse matrix format selection. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '18, page 94–108, New York, NY, USA, 2018. Association for Computing Machinery.

[58] Yue Zhao, Weijie Zhou, Xipeng Shen, and Graham Yiu. Overhead-conscious format selection for spmv-based applications. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 950–959, 2018.

[59] Weijie Zhou, Yue Zhao, Xipeng Shen, and Wang Chen. Enabling runtime spmv format selection through an overhead conscious method. *IEEE Transactions on Parallel and Distributed Systems*, 31(1):80–93, 2020.

[60] Gangyi Zhu and Gagan Agrawal. Sampling-based sparse format selection on gpus. In *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 198–208, 2021.

[61] Haishan Zhu and Mattan Erez. Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems. *ACM SIGARCH Computer Architecture News*, 44(2):33–47, 2016.

[62] Qi Zhu, Bo Wu, Xipeng Shen, Li Shen, and Zhiying Wang. Co-run scheduling with power cap on integrated cpu-gpu systems. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 967–977. IEEE, 2017.

[63] Tsahee Zidenberg, Isaac Keslassy, and Uri Weiser. Multiamdahl: How should i divide my heterogenous chip? *IEEE Computer Architecture Letters*, 11(2):65–68, 2012.