electronics

MDPI

Article

Performance Analysis of Sparse Matrix-Vector Multiplication (SpMV) on Graphics Processing Units (GPUs)

Sarah AlAhmadi [1], Thaha Mohammed [2], Aiiad Albeshri [3], Iyad Katib [3] and Rashid Mehmood [4,*]

[1] Department of Computer and Information Sciences, Taibah University, Medina 42353, Saudi Arabia; Shahmdi@taibahu.edu.sa
[2] Department of Computer Science, Aalto University, 02150 Espoo, Finland; Thaha.Mohammed@aalto.fi
[3] Department of Computer Science, King Abdulaziz University, Jeddah 21589, Saudi Arabia; AAAlbeshri@kau.edu.sa (A.A.); IAKatib@kau.edu.sa (I.K.)
[4] High Performance Computing Center, King Abdulaziz University, Jeddah 21589, Saudi Arabia
* Correspondence: RMehmood@kau.edu.sa

Received: 15 September 2020; Accepted: 6 October 2020; Published: 13 October 2020

Abstract: Graphics processing units (GPUs) have delivered a remarkable performance for a variety of high performance computing (HPC) applications through massive parallelism. One such application is sparse matrix-vector (SpMV) computations, which is central to many scientific, engineering, and other applications including machine learning. No single SpMV storage or computation scheme provides consistent and sufficiently high performance for all matrices due to their varying sparsity patterns. An extensive literature review reveals that the performance of SpMV techniques on GPUs has not been studied in sufficient detail. In this paper, we provide a detailed performance analysis of SpMV performance on GPUs using four notable sparse matrix storage schemes (compressed sparse row (CSR), ELLAPCK (ELL), hybrid ELL/COO (HYB), and compressed sparse row 5 (CSR5)), five performance metrics (execution time, giga floating point operations per second (GFLOPS), achieved occupancy, instructions per warp, and warp execution efficiency), five matrix sparsity features (*nnz*, *anpr*, *npr variance*, *maxnpr*, and *distavg*), and 17 sparse matrices from 10 application domains (chemical simulations, computational fluid dynamics (CFD), electromagnetics, linear programming, economics, etc.). Subsequently, based on the deeper insights gained through the detailed performance analysis, we propose a technique called the heterogeneous CPU–GPU Hybrid (HCGHYB) scheme. It utilizes both the CPU and GPU in parallel and provides better performance over the HYB format by an average speedup of 1.7x. Heterogeneous computing is an important direction for SpMV and other application areas. Moreover, to the best of our knowledge, this is the first work where the SpMV performance on GPUs has been discussed in such depth. We believe that this work on SpMV performance analysis and the heterogeneous scheme will open up many new directions and improvements for the SpMV computing field in the future.

Keywords: sparse matrix-vector multiplication (SpMV); high performance computing (HPC); sparse matrix storage; graphics processing units (GPUs); CSR; ELL; HYB; CSR5; parallelization; heterogeneous computing

1. Introduction

Sparse matrix-vector multiplication (SpMV) is fundamental to many scientific, engineering, and other applications [1–9]. These include web ranking [10,11], communication and networked systems [12], finding steady-state and transient solutions of Markov chains [13,14], and many others [1,2,15].

Another application area of SpMV computations that has become of colossal importance these days is machine and deep learning [16–18]. The SuiteSparse collection, the University of Florida repository of sparse matrices [2], comprises thousands of matrices that are collected from tens of application domains. The significance of SpMV computations is also evident from the fact that sparse linear algebra operations is included by the Berkeley scientists in their set of motifs, called the seven dwarfs [1].

An SpMV operation can be written mathematically as $y = Ax$, where A is a sparse matrix, x is a dense vector, and y is their product that needs to be computed. Sparse matrices arising from these applications are very large and sparse. They contain a relatively small number of nonzero elements, which arise due to various properties of the underlying system. Moreover, the sparsity structure of these matrices varies greatly. Therefore, parallelizing and loadbalancing SpMV computations on multiple cores or compute nodes, and the associated memory access challenges limit SpMV performance. Naturally, a great variety of specialized data structures and algorithms have been devised over the years for SpMV computations on the mainstream processor architectures including CPUs [19–29], field programmable gate arrays (FPGAs) [30–37], and manycore integrate architectures (MICs) [38–44]. Graphics processing units (GPUs) that provide massive computing capability through a large number of cores and high memory bandwidth have accelerated the performance of many applications [45–53]. SpMV is not an exception and has benefited from GPU computing [18,42,54–59]. However, due to the extreme variations in the sparsity structure of matrices, no single scheme provides consistent and sufficiently high performance on any processor architecture [60–65]. We have done an extensive review of SpMV techniques on GPUs and believe that the performance of SpMV techniques on GPUs has not been studied in sufficient detail; see for instance the most recent review of SpMV computations on GPUs [55]. The proposed schemes are mainly being evaluated and compared in terms of the SpMV throughput in floating point operations per second (FLOPS), which alone does not provide a deep insight into the SpMV storage and computations. This line of research broadly aims to analyze and improve the performance of SpMV computations. Towards this aim, we have developed a range of techniques over the years to propose novel storage schemes and algorithms on CPU [66–72], MIC [73–75], and GPU architectures [76,77]. Our research has also focused on the applications of SpMV computations [78–85].

In this paper, we provide a detailed performance analysis of SpMV performance on GPUs. For the purpose, we selected four notable sparse matrix storage schemes—compressed sparse row (CSR), ELLAPCK (ELL), hybrid ELL/COO (HYB), and compressed sparse row 5 (CSR5)—and evaluate their performance using five performance metrics. The five performance metrics are execution time, giga FLOPS (GFLOPS), achieved occupancy, instructions per warp, and warp execution efficiency. We selected 17 sparse matrices that have been widely used by SpMV researchers for performance analysis [39,77,86,87]. These matrices are from 10 application domains including chemical simulations, computational fluid dynamics (CFD), electromagnetics, linear programming, economics, and others. We provide a detailed discussion of the SpMV performance in terms of the five performance metrics mentioned above against five matrix sparsity features, the number of nonzero elements in the matrices (*nnz*), the average number of nonzero elements per row (*anpr*), the *variance* in the number of nonzero elements per row (*npr variance*), the maximum number of nonzero elements per row (*maxnpr*), and the mean index distance between the first and the last nonzero elements in a row (*distavg*).

Subsequently, based on the deeper insights gained through the detailed performance analysis, we propose a technique for improving the HYB scheme. The scheme is called the heterogeneous CPU–GPU Hybrid (HCGHYB) scheme. It utilizes both the CPU and GPU in parallel to improve the performance. Specifically, we multiplied matrix and vector elements on GPU, while the *reduce operation* in SpMV computation (that should be carried out *atomically*) was performed on CPU. Since the reduce operation is a compute-intensive atomic operation, our scheme HCGHYB provides a better performance over the standard HYB format by an average speedup of 1.7x.

To the best of our knowledge, this is the first work where the SpMV performance on GPUs has been discussed in such depth. The detailed performance analysis provided in this paper has

helped us to understand the performance bottlenecks of the HYB scheme and we were able to devise a scheme to improve its performance. The HYB scheme is a popular choice in many SpMV and iterative solvers for sparse linear equation systems and therefore our proposed scheme is expected to generate high impact. Moreover, heterogeneous computing involving multiple processor and computing architectures is an important direction for SpMV and other application areas. We believe that this work on SpMV performance analysis and the heterogeneous scheme will open up many new directions and improvements for the SpMV computing field in the future.

The rest of the paper is organized as follows. Section 2 provides information about the matrix dataset including sparsity features of the sparse matrices in the dataset. The next four sections, Section 3 to Section 6, provide a detailed performance analysis of the SpMV computations using the four selected sparse storage schemes, CSR, ELL, HYB, and CSR5. Section 7 provides comparative performance of the four sparse schemes. Section 8 introduces our proposed scheme and provides a comparative analysis of its performance. Section 9 concludes and provides directions for future work.

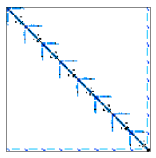
2. Dataset, Sparsity Features, and Performance Metrics

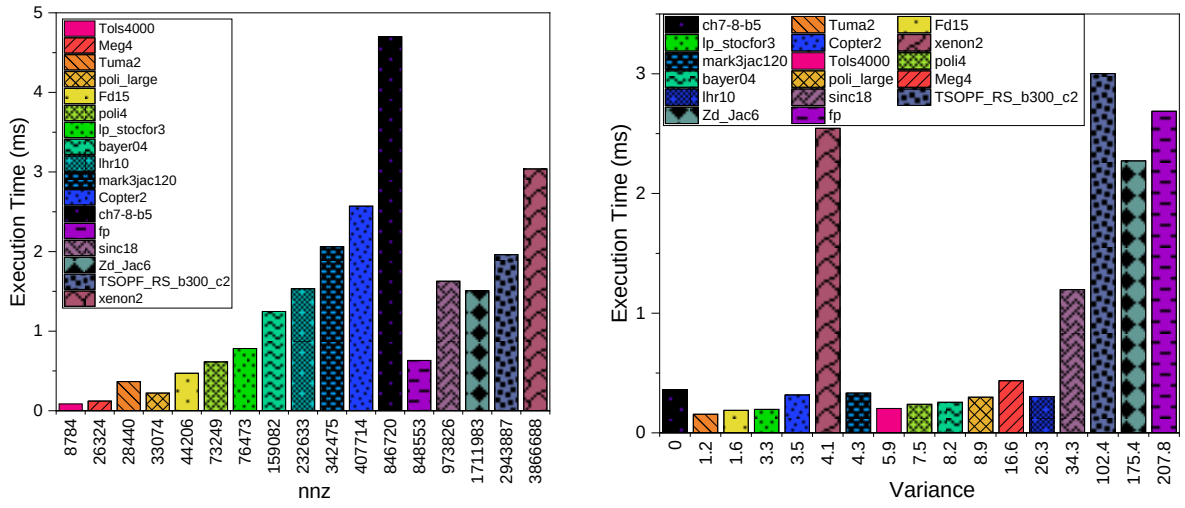
2.1. Dataset and Sparsity Features

Table 1 provides details of the dataset used in this paper for the analysis of SpMV performance on GPUs. The first column depicts the sparsity structure of the matrices. Column 2 gives the name of the matrix followed by the number of rows and columns in the matrix. The next five columns give the five matrix features, the number of nonzero elements in the matrices (*nnz*), the *variance* in the number of nonzero elements per row (*npr variance*), the average number of nonzero elements per row (*anpr*), the maximum number of nonzero elements per row (*maxnpr*), and the mean index distance between the first and the last nonzero elements in a row (*distavg*). The last column gives the application domain of the matrices.

2.2. Performance Metrics

We used five performance metrics for investigating the SpMV performance. The execution time is defined as the time taken to execute SpMV in a unit of time. GFLOPS is the number of floating point operations per second in billions achieved during the SpMV execution and hence is the measure of computational throughput achieved for the computation on a GPU. Achieved occupancy (AO) is defined as the ratio of the average active warps per cycle to the maximum number of warps supported on an SM. This metric provides an indication on the utilization of the GPU. Instructions per warp (IPW) is an indicator of the existence of thread divergence. A low IPW value indicates lower divergence and therefore better performance [88]. In addition to the IPW metric to detect divergence, we have the warp execution efficiency (WE) metric, which is defined as the ratio of the average number of active threads per warp to the maximum number of threads per warp supported on a multiprocessor expressed as a percentage [89].





(**a**) (**b**)

Figure 1. Compressed sparse row (CSR) execution time against: (**a**) nonzero elements in the matrices (*nnz*) and (**b**) nonzero elements per row (*npr variance*).

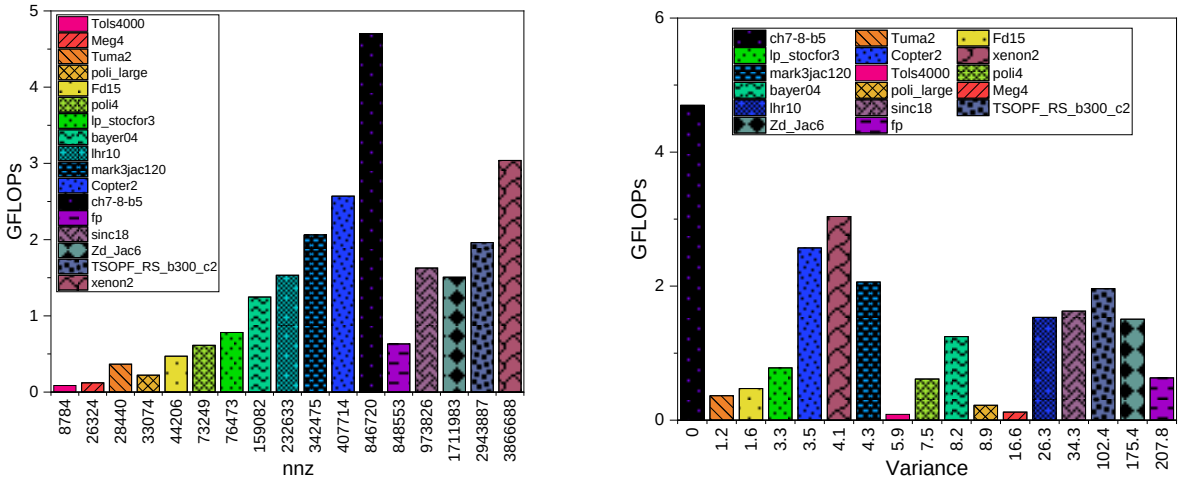
Generally, the sparse matrices can be classified as structured or unstructured based on the sparsity pattern. However, within both these sparsity patterns, hierarchically there exists more complex structural patterns that affects the performance. For example, if we consider *poli_large* and *fd15* matrices, *fd15* requires lower execution time than *poli_large* even though *fd15* has a higher *nnz* value. This is because *fd15* has a lower *variance* value than *poli_large*. Another example involves *sinc18* and *fp* matrices, both of which are unstructured, but with different sparsity patterns. Although *sinc18* has a higher *nnz* value than *fp*, it requires a shorter execution time than *fp* due to its sparsity structure. Moreover, *sinc18* has a lower *variance* value than *fp*. From the matrix visualization, it is clear that *sinc18* has a sparsity structure simpler than that of *fp*, and *sinc18* has a better *nnz* distribution than *fp*, where they are spread out among the matrix dimensions.

Figure 1b compares the execution time of CSR with respect to variance. We see that, with some exceptions, a consistent increase in execution time as *variance* increases. A higher *variance* value is defined as a higher variation in the *nnz* values of each row in a matrix. This leads to load imbalance among the threads when a single thread per row is utilized as seen in CSR. The load imbalance issue is one of the main drawbacks of SpMV computations on the GPU. However, other factors can definitely cause the existing exceptions. For example, in the case of the *mark3jac120* and *tols4000* matrices, even though the latter has a larger *variance* value than *mark3jac120*, it has a better execution time due to the considerable difference in their *nnz* values, which are 8784 and 34,2475, respectively. The same is the case for *xenon2* and *mark3jac120* even though both have structured patterns and similar variances. In this case, *mark3jac120* exceeds *xenon2* because *xenon2* is much larger than *mark3jac120* in terms of *nnz* values. In addition to the *nnz* value's impact, the sparsity pattern can also cause exceptions. For example, *lhr10* exceeds *meg4* despite its larger *variance* value because *meg4*'s sparsity pattern is more complex than *lhr10*'s in terms of *nnz* distribution. This is clear if we examine the differences between the *anpr* and *max_npr* features, which are larger in *meg4* compared to *lhr10*.

3.2. GPU Throughput

Figure 2a,b illustrate the GPU throughput of the CSR scheme with a comparison of each matrix *nnz variance* values. Figure 2a shows a regular increase in the first twelve matrices, followed by a break at the *fp* matrix and a subsequent return to increasing GFLOPS. The rest of the matrices after the peak GFLOPS values (i.e., *ch7-8-b5* and *copter2*) are *fp*, *sinc18*, *Zd_Jac6*, *TSOPF_RS_b300_c2*, and *lp_stocfor3*. Although *xenon2*, is the larger matrix in terms of *nnz* and has a structured pattern, the unstructured

matrices *ch7-8-b5* and *copter2* exceed it and have higher GFLOPS values. We noted that the sparsity pattern plays an important role in this variation.



(**a**) (**b**)

Figure 2. CSR giga floating point operations per second (GFLOPS) against: (**a**) *nnz* and (**b**) *npr variance*.

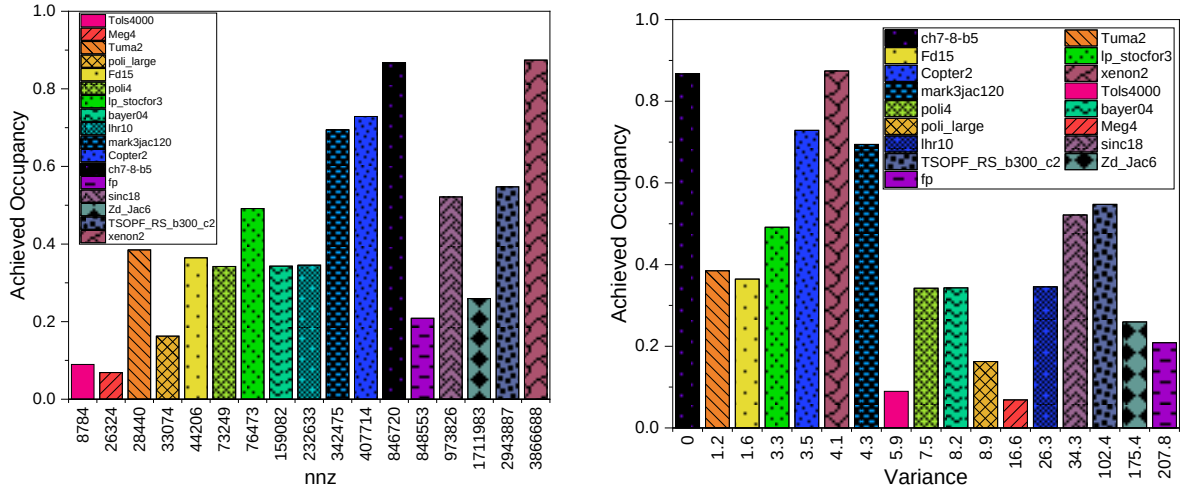
More precisely, two main factors affect sparsity. One is the distance between *nnz*s within a row in terms of granularity (in the case of CSR, we have one thread per row for processing). Therefore, the distance between the *nnz*s in a row affects the overall throughput because it affects the memory access pattern. In SpMV computations, we usually deal with multiple arrays to read the data, and the more gaps we have between nnzs in a row, the slower the access, specifically for the array x in the equation $Ax = b$. The second factor is the number of *nnz*s on each row. In our study, this is measured using *variance*. To prove this, we examined the *fp* matrix at the changing point in the figure, and if we accurately measure the distances between the *nnz*s on each row, we find large gaps in the rows. In addition, there is a variation in the number of *nnz*s within the rows. Compared to the best performance at *ch7-8-b5* and *copter2*, which have *variance* values of 0 and 3.55, respectively, *fp* has a value of 207.83. This proves that the second factor of sparsity impact, the distances between *nnz*s in both matrices, are better than in *fp*, as the visualization indicates. However, we have fewer GFLOPS for *tols4000* and *meg4* than for *fp*, but this is due to their small *nnz* value, whereas we compared *fp* with similar matrices.

There is no direct relation between matrix *variance* and GPU GFLOPS as observed in Figure 2b. However, we can generally observe, lower *variance* values have better GFLOPS when there exists a good sparsity pattern and a large number of *nnz*. In other words, for *tols4000* the *variance* of 5.92 is relatively small but it has lower GFLOPS compared to matrices with larger *variance* values due to the big difference between the number of *nnz* (e.g., *tols4000* and *sinc18*), which have a *variance* of 34.32. The same occurs with *fd15* with a *variance* of 1.65 and *TSOPF_RS_b300_c2* with a *variance* of 102.4. *Ch7-8-b5* has 0 variance, which indicates that all rows have the same number of *nnz* in addition to relatively close distances between *nnz* per row and a max-nnz. This is evident in our findings for other matrices such as, *copter2*, *xenon2*, and *mark3jac120* with a *variance* of 3.55, 4.11, and 4.36 respectively.

3.3. GPU Utilization

Figure 3a,b report the results of the achieved occupancy of the CSR scheme. AO affects the performance since it indicates the rate of GPU SM utilization. More exploitation of GPU parallelization capabilities improves the overall performance. Generally, from Figure 3a, a higher *nnz* leads to better achieved occupancy, but then we observe a few matrices with higher *nnz* and lower achieved occupancy. Consider *fp* and *zd_Jac6*, both which have a large number of *nnz* but attained low achieved occupancy. This is also the reason for the higher execution time for these two matrices. We reason that

the features of the sparsity structures include *nnz*, *npr variance* and the existence of gaps per row on the matrices. The last two conditions are in existence since we have a big *variance* for both matrices and there exists gaps within a row. The first three matrices have lower achieved occupancy since they are considered the smallest matrices in terms of *nnz*. The highest occupancy is attained by *ch7-8-b5* and *xenon2*, where all the conditions are satisfied (i.e., large number of nnz, low variance, and small gaps in the rows).

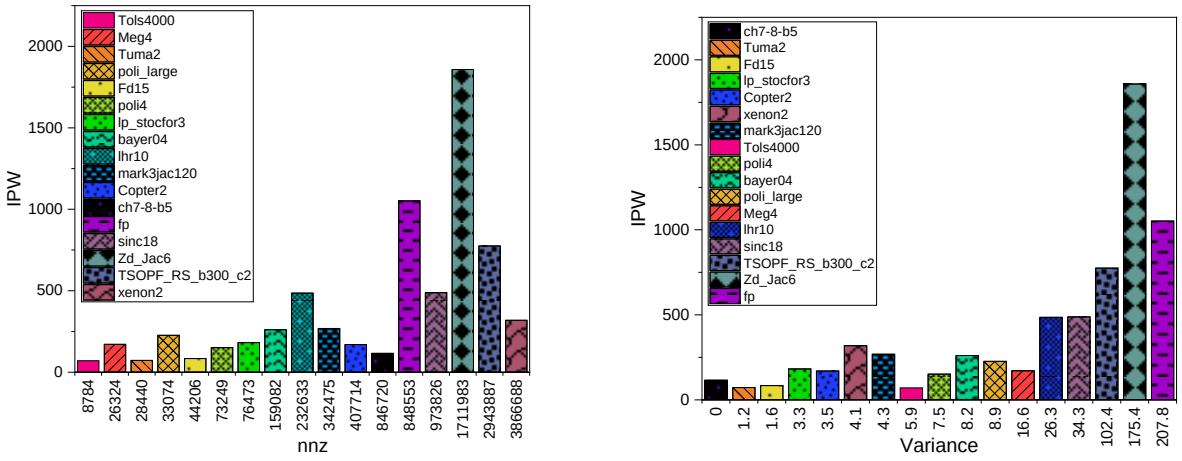


(**a**) (**b**)

Figure 3. CSR achieved occupancy against: (**a**) *nnz* and (**b**) *npr variance*.

In Figure 3b, we study the effect of matrix *variance* on the achieved occupancy. We find that low *variance* results in higher achieved occupancy while higher *variance* results in lower achieved occupancy. However, we cannot ignore the impact of *nnz*, which reverses this trend in some cases as in *tols4000*, *meg4*, and *poli_large*.

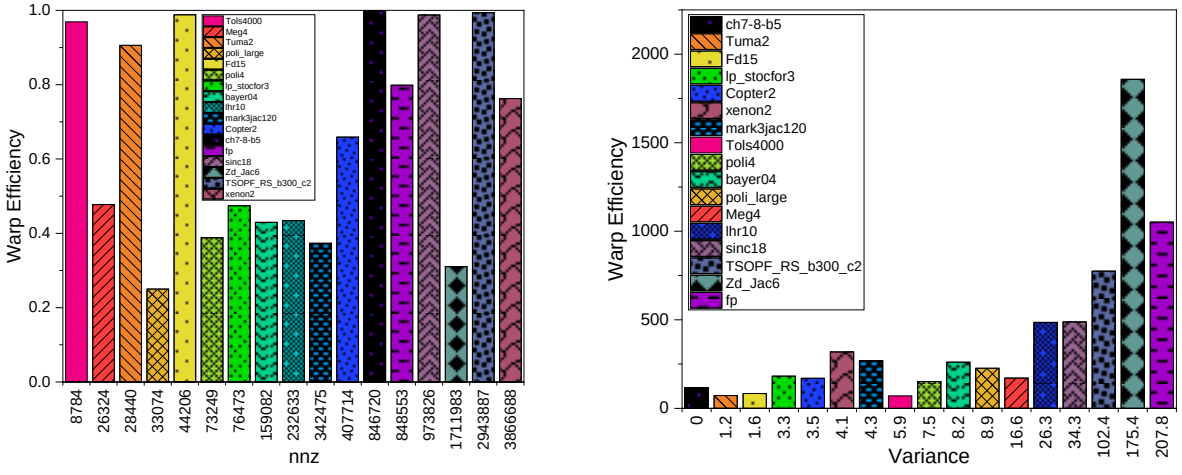
Figure 4a,b shows the instructions per warp (IPW) for the CSR scheme. Figure 4b illustrates that with higher *variance* values (which means a large variation of number of *nnz* per rows) we have a higher IPW, which indicates a higher divergence. However, in some cases such as *Zd_Jac6* and *fp*, although the *variance* of *fp* is larger than *Zd_Jac6*, *Zd_Jac6* has higher IPW due to larger *nnz* than *fp*. The same happens for *mark3jac120* and *xenon2*, where the latter is a little higher than the former. Moreover, *tols4000* has fewer IPW than *mark3jac120* and *xenon2*, although its *variance* is higher than *mark3jac120* and *xenon2*, but this is due to its small nnz. So, we can conclude that with more *nnz* variation per row in a matrix we have more divergence and therefore low performance while taking into consideration the matrix size in terms of nnz. In addition to the IPW metric to detect the existing divergence, we have the warp execution efficiency metric. In case of warp divergence some threads will be permanently switched off for various reasons such as, the presence of conditional control statements (which cause MIMD) or uncoalesced memory access (which causes waiting time to bring the data in different memory transactions). These factors affect the active threads within a warp because they result in work variation within a warp and this sometimes serializes the work as GPU is an SIMD device. Thus, warp efficiency metric provides a deep understanding of GPU utilization and indicates the level of parallelism exploited. High warp efficiency percentage indicates a high level of thread exploitation, and low warp efficiency indicates the existence of divergence.



(**a**) (**b**)

Figure 4. CSR instructions per warp (IPW) against: (**a**) *nnz* and (**b**) *npr variance*.

Figure 5a shows the impact of *nnz* on warp efficiency. There is no real impact or consistent increase of warp efficiency as *nnz* increase. Figure 5b shows a comparison between warp efficiency and matrix variance. In CSR, when the granularity is one thread per row, we need a conditional control statement. In cases of high variation of *nnz* per row, there is a high possibility of warp divergence as the execution times per thread differ, which results in the shorter threads waiting for the longer threads and hence idle threads (within a warp). Subsequently, the overall performance is impacted. Figure 5b provides evidence for our findings. We observe that matrices with low *variance* have high warp efficiency and matrices with high *variance* have lower warp efficiency with some exceptions. These exceptions exist because we definitely have other factors that affect the performance. The main factor that causes these exceptions is the number of *nnz* per row. When the average *nnz* per row (regardless of the variance) is higher, it will give us better warp efficiency. For example, *fp* and *Zd_Jac6* both have big *variance* but *fp*, which has the highest variance, has achieved better warp efficiency than *Zd_Jac6* because the average number of *nnz* per row (*anpr*) on *fp* is 112 and on *Zd_Jac6* it is 74. So, having more *nnz* on rows results in better warp efficiency. This happens with *mark3jac120* and *xenon2*, where *anpr* was 6 and 24, respectively.



(**a**) (**b**)

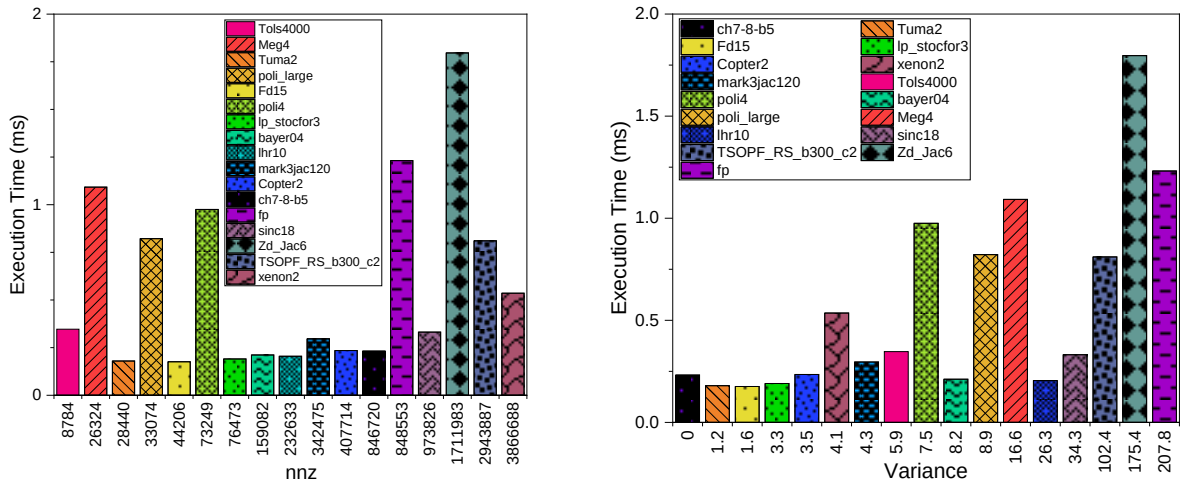
Figure 5. CSR warp efficiency against: (**a**) *nnz* and (**b**) *npr variance*.

4. ELLPACK (ELL)

The ELLPACK (ELL) [91] format uses two 2D arrays *Val* and *Col* to store a sparse matrix. The *Val* array contains the nonzero values while the 2D array *Col* contains the column indices of the nonzero elements. The correct row index is preserved in the *Col* array.

4.1. Execution Time

Figure 6a,b illustrates the ELL execution time. Figure 6a compares the execution time of each matrix against its *nnz* value. There is a variation in the attained execution time as the number of *nnz* increases. Thus, *nnz* does not have a heavy impact on the execution time. The peak times were achieved by the matrices, *Zd_Jac6*, *fp*, *poli4*, *meg4*, and *poli_large*. All these matrices have different *nnz* that varies from quite high to very low values. This implies that the affect of *nnz* on execution time for ELL is negligible. Besides, all the matrices that have slow execution time are the unstructured matrices. This is not surprising as ELL is mainly suitable for structured matrices.



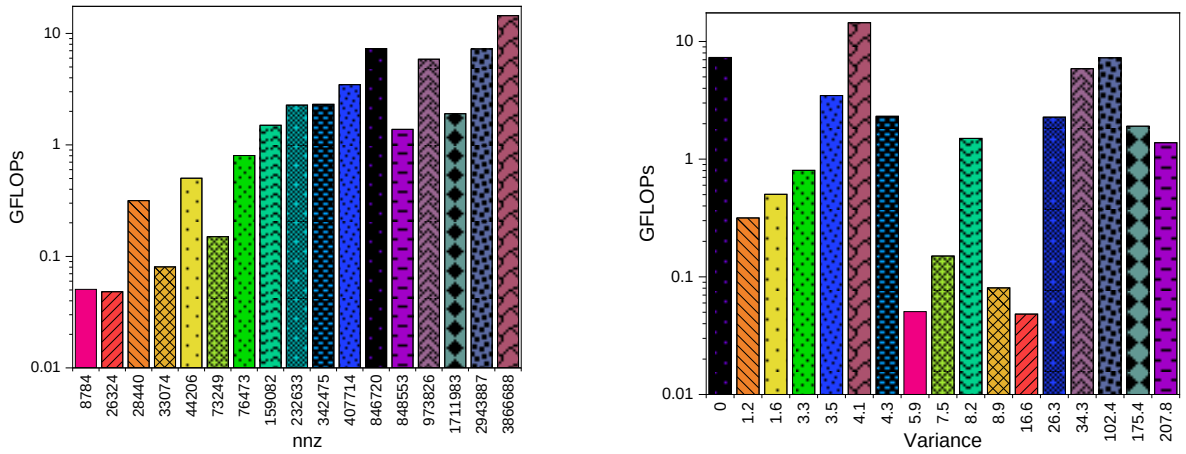
(**a**) (**b**)

Figure 6. ELLPACK (ELL) execution time against: (**a**) *nnz* and (**b**) *npr variance*.

In Figure 6b, we study the execution time for ELL against the *nnz* variance. We observe the matrices with larger *nnz* and *variance* have achieved the highest execution time as compared to the matrices with lower variances. However, even with a low variance, some matrices attained high execution times (e.g., *poli4* and *meg4*). This is because there are other factors that affects the execution time. In the ELL storage format, the row size is equal to *maxnpr* and the remaining rows are padded with zeros. If we take this into consideration in *poli4* (as an example), it indeed has relatively small *variance* but the maximum *nnz* among its rows (max_anpr) is 304 while the *anpr* is 2. This is a big difference and this implies that large number padded zeros are required to store these matrices in ELL. Therefore during the SpMV computations, we exclude these zeros using a conditional control statement, which as we have illustrated before causes a thread divergence, and hence leads to higher execution time. The same happens with *poli_large* where the *anpr* and *max_anpr* are 2 and 491, respectively. Similarly, in case of *meg4*, the *anpr* and *max_anpr* are 4 and 1193, respectively.

4.2. GPU Throughput

Figure 7a illustrates the throughput achieved by CSR with varying nnz. We observe that there is an increase in GFLOPS as the *nnz* increase with some exceptions. These exceptions are *poli4*, *fp*, and *Zd_Jac6*, which are unstructured matrices. In Figure 7b, we study the throughput relation to the matrix variance. There is no consistent trend that can be observed. Thus, the achieved GFLOPS does not depend on the matrix *variance* but rather it depends more on the *nnz* as shown in Figure 7a, considering the discussed factors.

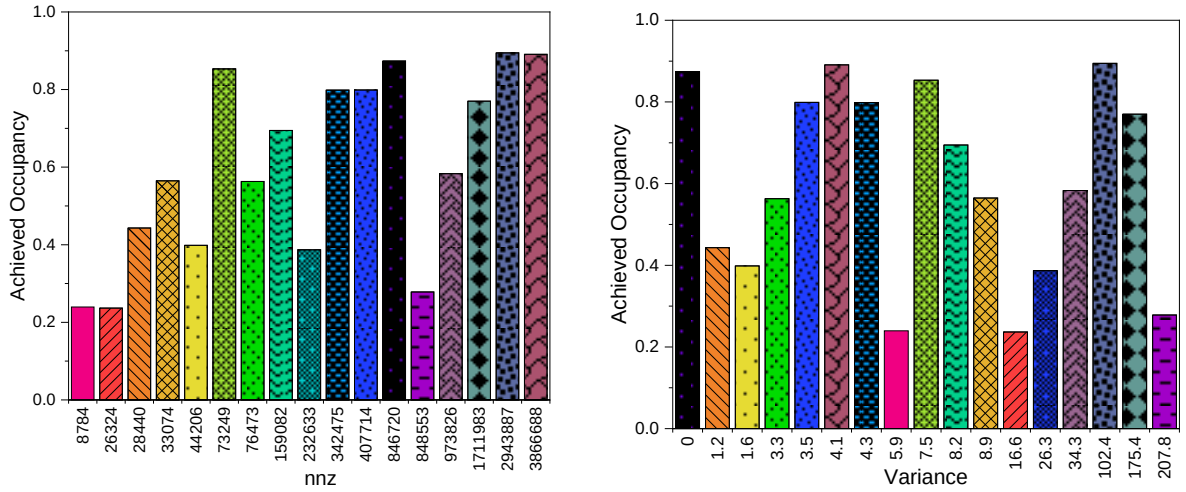


(**a**) (**b**)

Figure 7. ELL GFLOPs against: (**a**) *nnz* and (**b**) *npr variance*.

4.3. GPU Utilization

Figures 8–10 provide a detailed description of GPU utilization on ELL scheme. Figure 8a,b discuss the achieved occupancy. Higher exploitation of GPU parallelization capabilities results in better performance. Generally, as observed in Figure 8a, a larger *nnz* leads to better achieved occupancy with some exceptions, such as in *lhr10* compared to *bayer04* and *fp* compared to *ch7-8-b5*. This is explained well by considering other matrix features in Figure 8b.



(**a**) (**b**)

Figure 8. ELL achieved occupancy against: (**a**) *nnz* and (**b**) *npr variance*.

In Figure 8b, we see the *variance* of *lhr10* (26.37) is bigger than *bayer04* (8.2), which is the reason for *lhr10*'s low achieved occupancy. Furthermore, *fp* and *ch7-8-b5* are similar, where variances are 207.83 and 0, respectively. However, there are other cases where we have a higher achieved occupancy even with higher *variance* (e.g., *TSOPF_RS_b300_c2* and *Zd_Jac6* compared to *sinc18*). That is definitely caused by other factors such as nnz, which is greater on *TSOPF_RS_b300_c2* and *Zd_Jac6* than *sinc18* or meg4. Nevertheless, in some cases this is not totally true, such as in *mark3jac120* and *TSOPF_RS_b300_c2*, where *mark3jac120* is smaller than *TSOPF_RS_b300_c2* in terms of *nnz* and they have a different *variance* of 4.36 and 102.4, respectively. However, they have achieved similar achieved occupancy although both are structured. That is due to another factor, which is the gaps between *nnz* on each row among all rows, which *TSOPF_RS_b300_c2* has but *mark3jac120* does not.

Figure 9a studies the impact of *nnz* on the IPW metric, which indicates the existence of divergence. As we can see, there is a variation on the achieved IPW as the *nnz* increases, and this indicates that the sparsity structure of the matrices play a more important role than *nnz* alone, as we can see in Figure 9b showing the comparison against matrix variance.

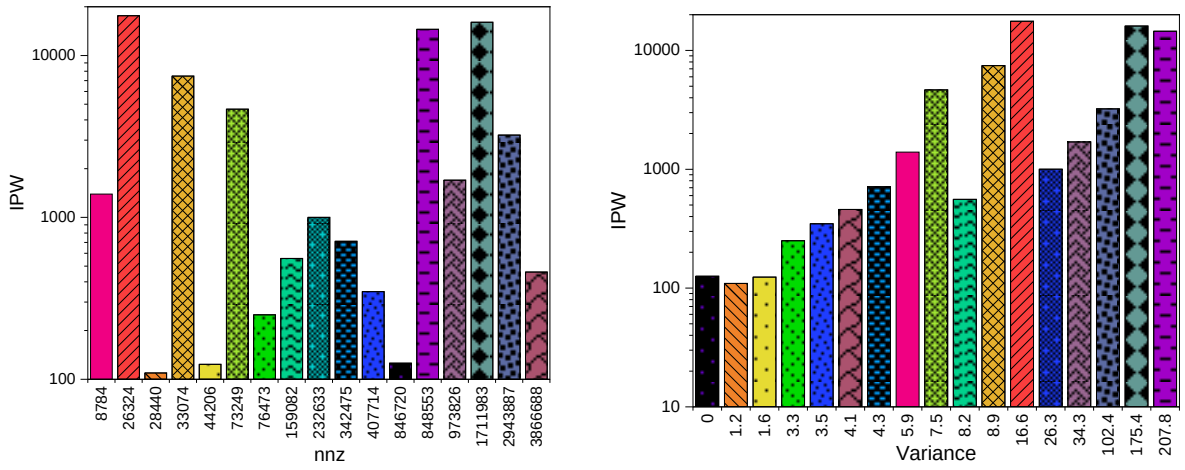


Figure 9. ELL instructions per warp (IPW) against: (**a**) *nnz* and (**b**) *npr variance*.

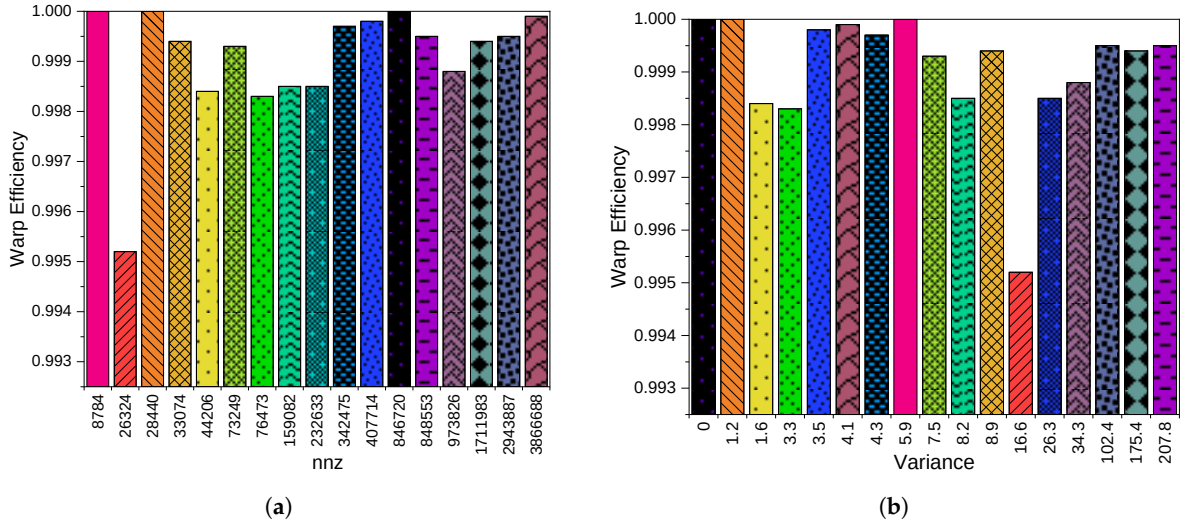


Figure 10. ELL warp efficiency against: (**a**) *nnz* and (**b**) *npr variance*.

In Figure 9b, we can see that with higher *variance* values we have a high IPW, which indicates a higher divergence with some exceptions. As an example, for *poli4* and *bayer04* (where the variances are 7.57 and 8.2, respectively), the divergence level on *bayer04* is less than *poli4* despite its bigger variance. This is due to the big difference between *anpr* and *max-npr* of 2 and 304, respectively, on *poli4*, while they are 7 and 34 on *bayer04*, respectively. This factor will cause divergence as explained earlier, and this is mainly because ELL depends on the *max-npr* to determine the size of the matrix row. The same occurs with *meg4* and *lhr19* where the *variance* is 16.66 and 26.37, respectively. *meg4* has an *anpr* and *max-npr* of 4 and 1193, respectively, while *lhr10* has 21 and 63, respectively.

Figure 10a shows the attained warp execution efficiency. All the matrices achieve a high warp execution efficiency regardless of *nnz* and are above 99% with some differences of the fractions. This indicates that there is an extremely low level of warp divergence in ELL format. Moreover, *nnz* does not affect this metric too much but it tends to have more warp efficiency with a higher nnz, with some exceptions resulting from the differences in the sparsity structures.

Figure 10b shows the impact of *variance* on the warp execution efficiency metric. We tend to have lower warp efficiency with the bigger variances. However, we have exceptions such as in *meg4*, which has the lowest warp efficiency, although there are matrices with bigger variances and that have achieved better warp efficiency, such as *sinc18* and *lhr10*. This is again due to the big difference between *max-npr* and anpr, which are 1193 and 4, respectively, in *meg4*, while this difference is too small on the comparable matrices. However, sometimes this is not the only factor that affects the warp efficiency; in some cases we also have better warp efficiency compared with matrices with lower variances, such as in *xenon2* compared to fd15 and *lp_stocfor3*. The differences between *max-npr* and *anpr* are relatively small in all of them, but the *nnz* is bigger in *xenon2*.

5. Hybrid ELL/COO (HYB)

The hybrid ELL/COO (HYB) [56] format uses the ELL format to store most of the nonzero elements of the sparse matrix. The remaining nonzero elements are stored using the COO format.

5.1. Execution Time

Figure 11a,b describes the HYB execution time. Figure 11a plots the execution time against *nnz* value for the 17 matrices. We tend to have a high execution time as *nnz* value increases. In Figure 11a we can observe *fp* and *Zd_Jac6* have the peak execution times in the figure, whereas *TSOPF_RS_b300_c2* and *xenon2* are bigger matrices but have attained short execution times. Thus, there has to be other factors that affect the result, and mostly they relate to the sparsity structure as shown in Figure 11b and is further discussed below.

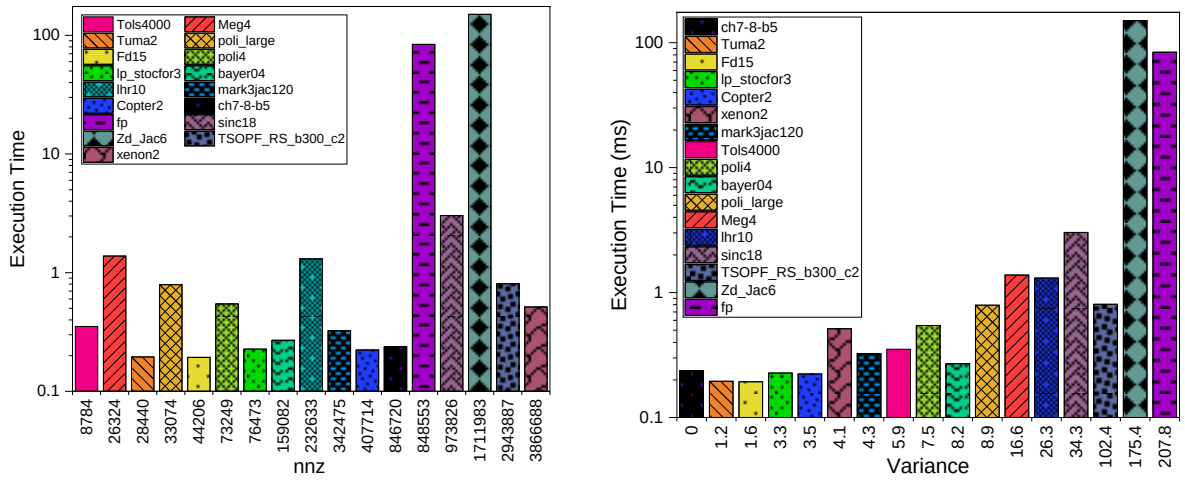


Figure 11. Hybrid ELL/COO (HYB) execution time against: (**a**) *nnz* and (**b**) *npr variance*.

In Figure 11b, we study the effect of *nnz variance* on the execution time. We find that the execution time increases as the *variance* of *nnz* increases. In addition, it is very clear that *fp* and *Zd_Jac6* have the biggest *variance* values and therefore have the longest execution times. As the *variance* value increase, the execution time also increase. However, we observe that *TSOPF_RS_b300_c2*, which has a big *variance* and a large nnz, has achieved better execution time compared to *sinc18*, which has smaller *variance* and a smaller nnz. This is because *anpr* in *TSOPF_RS_b300_c2* is bigger than in *sinc18*. The same happens with *bayer04* and *poli4* with a *variance* 8.2 and 7.57, respectively. *bayer04* achieved better execution time than *poli4* and again this was due to its bigger *nnz* and *anpr* compared to *poli4*. So, we conclude that *variance* value has a direct impact on the execution time for HYB format.

5.2. GPU Throughput

Figure 12a shows the attained GFLOPS compared to nnz. As we see, the GFLOPS increased as the *nnz* increased. Except again with *Zd_Jac6* and *fp*, which attained the smallest throughput. They both attained the longest execution times. So their structures are the worst ones for HYB format. Both have big *variance* and big difference between *anpr* and max-npr. A similar case was seen with *bayer04* and *lhr10*, where *lhr10*'s *variance* was bigger than *bayer04*'s. We further discuss this issue in Figure 12b.

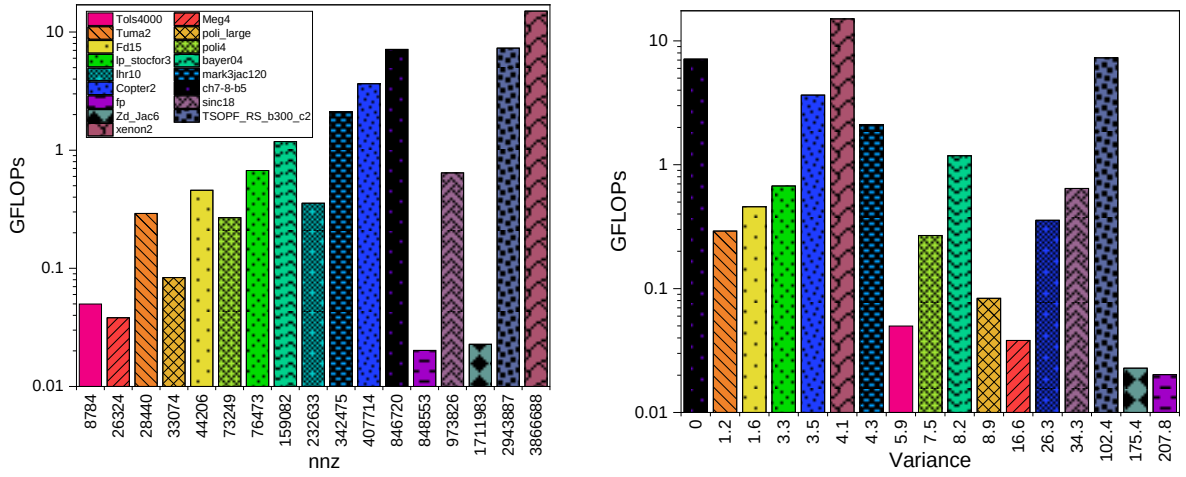


Figure 12. HYB GFLOPs against: (**a**) *nnz* and (**b**) *npr variance*.

Figure 12b shows the *variance* values for all the matrices and their achieved GFLOPS. As discussed, the matrices with the highest variances achieved the lowest GFLOPS and they are *Zd_Jac6* and *fp*. In addition, from Figure 12b it is clear that the matrices with the lowest *variance* attained the highest GFLOPS compared to the matrices with the highest *variance* values. Certainly, there are some exceptions due to other sparsity factors. One case involves *TSOPF_RS_b300_c2* and *sinc18*, where *TSOPF_RS_b300_c2* attained higher GFLOPS than *sinc18* although it has bigger variance. This is because *TSOPF_RS_b300_c2* has a bigger *nnz* than *sinc18*. The same occurs between *lp_stocfor3* and copter2 and between *poli4* and *tols4000*.

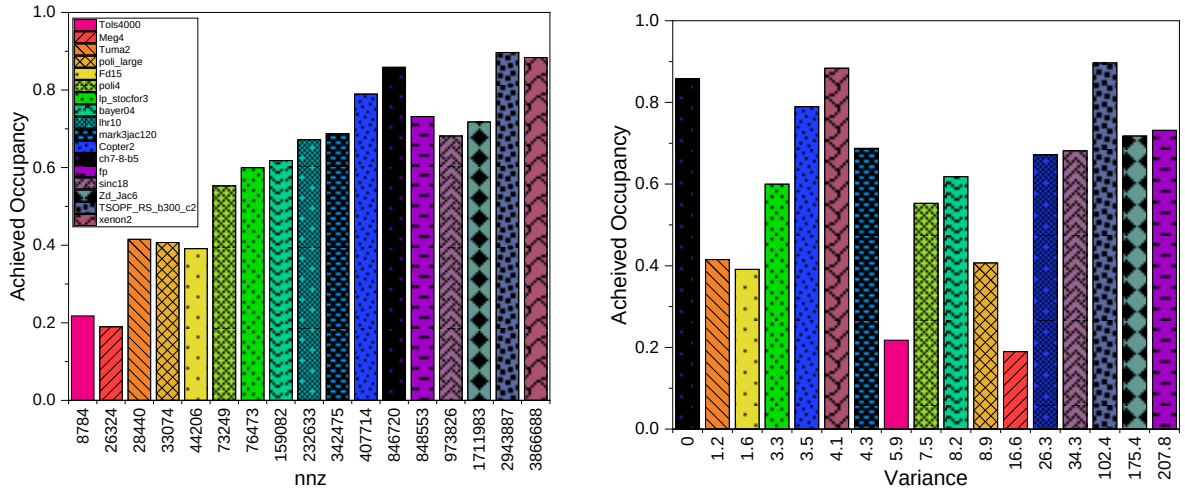
5.3. GPU Utilization

Figure 13a,b report the results of the achieved occupancy of the HYB scheme. This metric affects the performance since it indicates the rate of GPU SM utilization. More exploitation of GPU parallelization capabilities equals better performance.

From Figure 13a, a higher *nnz* leads to better achieved occupancy, but we observe a few exceptions. Let us consider *sinc18* and *ch7-8-b5*, both which have a larger number of nnz, but *sinc18* attained a lower achieved occupancy compared to *ch7-8-b5*. This may also be considered a clarification of the *sinc18* matrix's high execution time. We might deduce again that the main features of the sparsity structures are the nnz, variance, the existing gaps per row on the matrices, or the difference between *anpr* and max-npr. Again, looking to *sinc18* and *ch7-8-b5* sparsity structures, starting with *variance* we find that the *variance* of *ch7-8-b5* is lower than *sinc18*, which are 0 and 34.32, respectively. Thus, less *variance* leads to better achieved occupancy besides large nnz. This is further clarified in Figure 13b.

Figure 13b shows the impact of *variance* value on the achieved occupancy metric. We have found that there is no consistent behavior of the curve since we have high achieved occupancy with low and high variance, and this proves the impact of *nnz* on GPU utilization. We can also note the achieved occupancy of the matrix with *variance* 0 is similar to the matrix with *variance* 102.4 with the latter's superiority. This is due to many reasons. Firstly, the matrix with 102.4 *variance* (i.e., *TSOPF_RS_b300_c2*) has a larger *nnz* than the matrix with 0 *variance* (i.e., *ch7-8-b5*). Despite the large difference between *nnz*

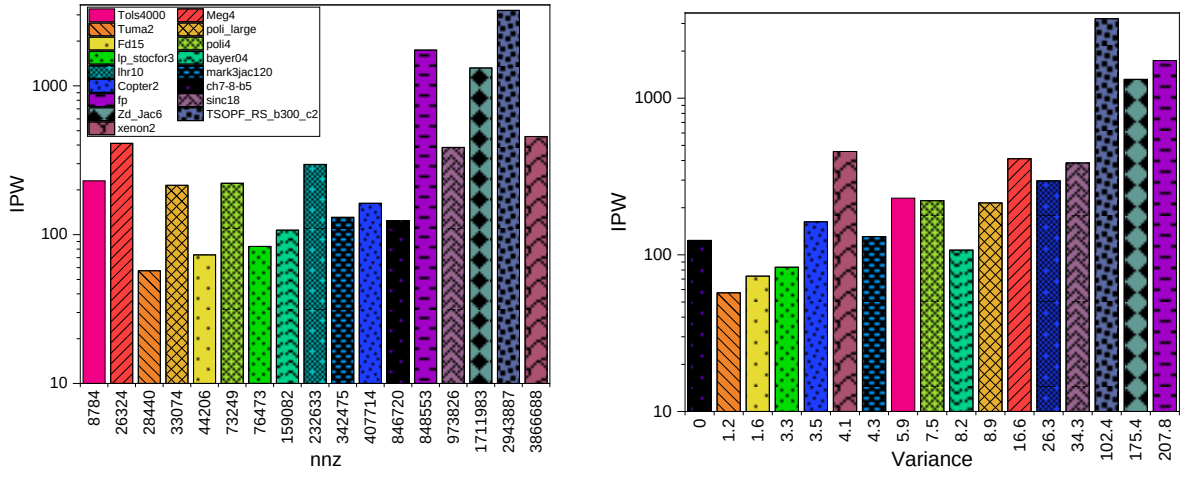
for both matrices, *ch7-8-b5* has closer achieved occupancy than *TSOPF_RS_b300_c2* and this is due to its smaller *variance* value. So, *variance* and *nnz* are complementary factors in HYB achieved occupancy. More evidence of nnz's impact is seen if we compare *TSOPF_RS_b300_c2* with *xenon2*, which has as large an *nnz* as *TSOPF_RS_b300_c2*. We see they achieved similar achieved occupancy despite the big difference between their *variance* with superiority of *TSOPF_RS_b300_c2* That is because *anpr* is larger in *TSOPF_RS_b300_c2* than *xenon2* despite *xenon2* having a larger *nnz* and lower *variance* than *TSOPF_RS_b300_c2*. The same happened between *bayer04* and *poli4*, as well as between *xenon2* and copter2. We can conclude that the first factor that affects HYB achieved occupancy is the nnz, then the *variance* value, then the *anpr* value, such that a high *nnz* with a low *variance* and big *anpr* give high achieved occupancy.



(**a**) (**b**)

Figure 13. HYB achieved occupancy against: (**a**) *nnz* and (**b**) *npr variance*.

Figure 14a,b shows the IPW results for the HYB scheme. This metric acts as an indicator of the existence of a divergence where low IPW values indicate a lower divergence and therefore better performance [88].



(**a**) (**b**)

Figure 14. HYB instructions per warp (IPW) against: (**a**) *nnz* and (**b**) *npr variance*.

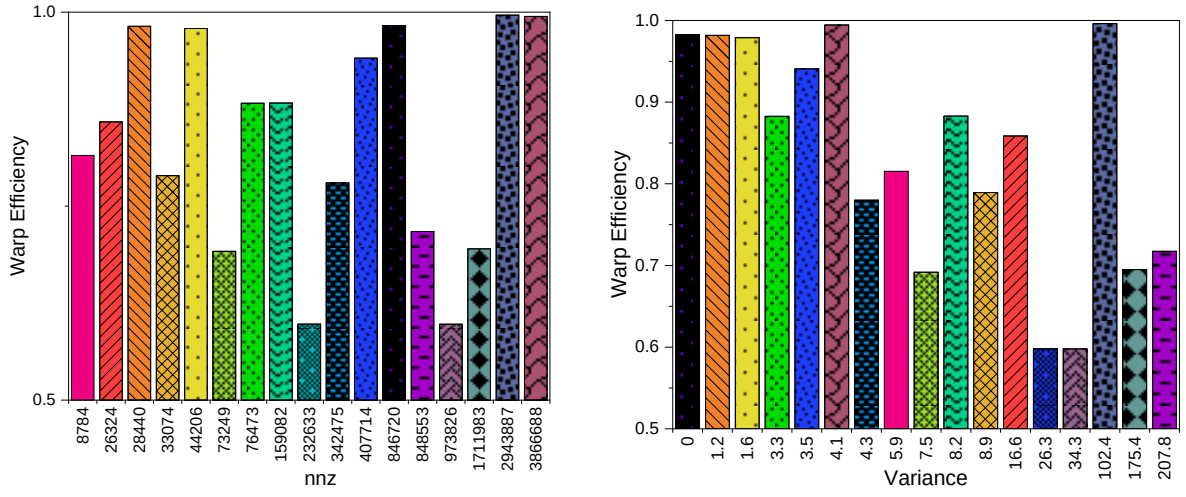
In Figure 14a, where we see the impact of *nnz* on the IPW metric, we note that there is oscillation between the lows and highs, the tendency to have a high IPW value with a higher *nnz* can be observed. Analyzing some case studies will clarify this variation. If we take *sinc18*, meg4, and *xenon2*, which have

similar IPW values and big differences in nnzs (26324, 973826, and 3866688, respectively), we conclude that the *nnz* does not play an important role in IPW value. This leads to the conclusion that the sparsity features definitely have a strong impression on the level of divergence we have on this scheme. This is further clarified in Figure 14b.

In Figure 14b we can see that with high *variance* values we tend to have a high IPW that indicates a high divergence. However, in some cases, such as *mark3jac120* and *xenon2*, although the *variance* of *mark3jac120* is larger than *xenon2*, *xenon2* has higher IPW due to its larger *nnz* than *mark3jac120*. The same happened for *bayer04* and copter2. Nevertheless, we see the opposite between *bayer04* and *poli4*, where *bayer04* has the bigger *variance* as well as the bigger *nnz* compared to *poli4*. This is because *poli4* has a big difference between *anpr* and max-npr. The same occurs between meg4 and *lhr10*. So, we can conclude that with more variation of *nnz* per row in a matrix we have more divergence and therefore low performance, taking into consideration the matrix size in terms of *nnz* value.

In addition to the IPW metric to detect the existing divergence, we have the warp execution efficiency metric. Therefore, in case of warp divergence some threads will be permanently switched off for reasons such as the presence of conditional control statements (which cause MIMD) or uncoalesced memory access (which causes waiting time to bring the data in multiple memory transactions). These factors affect the active threads within a warp because they result in work variation within a warp, and this sometimes serializes the work due to GPU is SIMD processor. Thus, this metric is another deep study of GPU utilization to clarify whether we have exploited the highest possible parallelism or not. High warp efficiency percentage indicates a high level of thread exploitation, and low warp efficiency indicates divergence existence.

Figure 15a shows a comparison between warp execution efficiency and the *nnz* value. We have a variation on the attained warp efficiency as *nnz* value increased. We can see we have high efficiency with a low *nnz* as well as with a high nnz. In addition, we have low warp efficiency with a low *nnz* as well as with a high nnz. Starting with the lowest warp efficiency in *sinc18* and *lhr10*, both with big differences in *nnz* have achieved similar warp efficiency percentage. We can first figure out that the impact of *nnz* on HYB warp efficiency is less than the impression of sparsity features. This will be clearer with the next figure, where we study the impact of *variance* feature on the attained warp efficiency.



(**a**) (**b**)

Figure 15. HYB warp efficiency against: (**a**) *nnz* and (**b**) *npr variance*.

Figure 15b shows the impact of *variance* value on warp execution efficiency. It is clear that the matrices with lower *variance* values have achieved higher warp efficiency than the matrices with higher *variance* values. Looking at the *variance* of our example from the previous figure showing *sinc18* and *lhr10*, they have closer *variance* values with high efficiency at the *variance* values of the matrices before and after them! Thus, looking to other examples will eliminate this ambiguity. Why does

TSOPF_RS_b300_c2 with 102.4 *variance* have higher efficiency than *sinc18* with 34.32 *variance* value? *TSOPF_RS_b300_c2* has a higher *nnz* than *sinc18*. The *anpr* of *TSOPF_RS_b300_c2* is higher than it is for *sinc18*. In addition, *sinc18* is an unstructured matrix while *TSOPF_RS_b300_c2* is a structured matrix. For more evidence, take *xenon2* and *TSOPF_RS_b300_c2* as another example, since both have similar warp efficiency with a big difference in *variance* value They also have similarly high nnzs and they are both structured matrices. Looking at *poli4* and *bayer04*, which have 7.57 and 8.2, respectively, *bayer04* has achieved higher warp efficiency due to its higher *nnz* compared to *poli4*. Consequently, a high *nnz* in addition to a high *anpr* with a structured pattern of the matrix is the best combination to obtain high warp efficiency in the HYB scheme.

6. Compressed Sparse Row 5 (CSR5)

The CSR5 [39] format is an improvement over the classical CSR format. See [39] for details.

6.1. Execution Time

Figure 16a,b describe CSR5 execution time. Figure 16a shows the execution time of each matrix against the nnz. We observe from Figure 16b that the execution time increase as the value of the *nnz* increase. Thus, the value of *nnz* has a direct effect on the execution time while using CSR5 scheme.

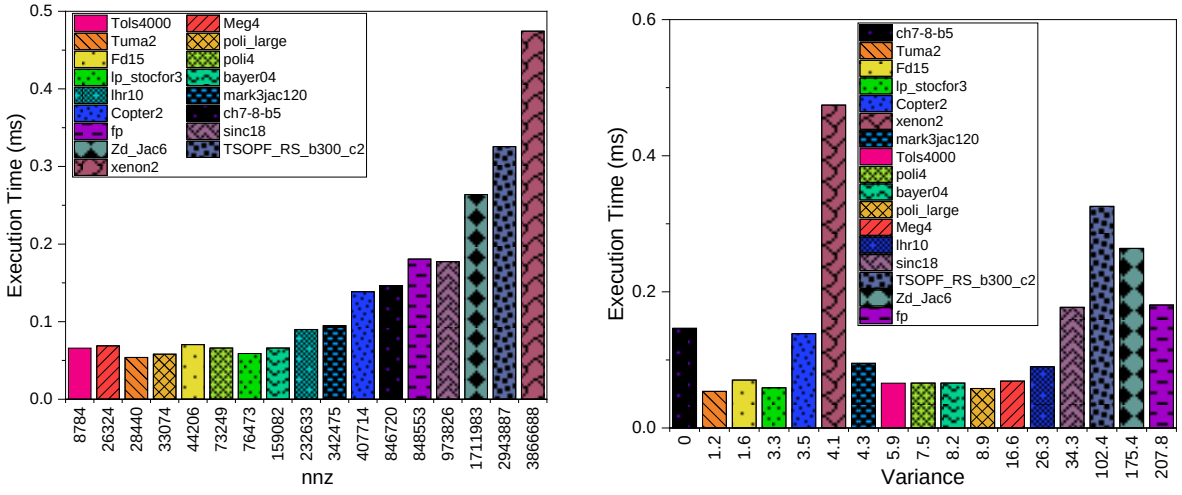


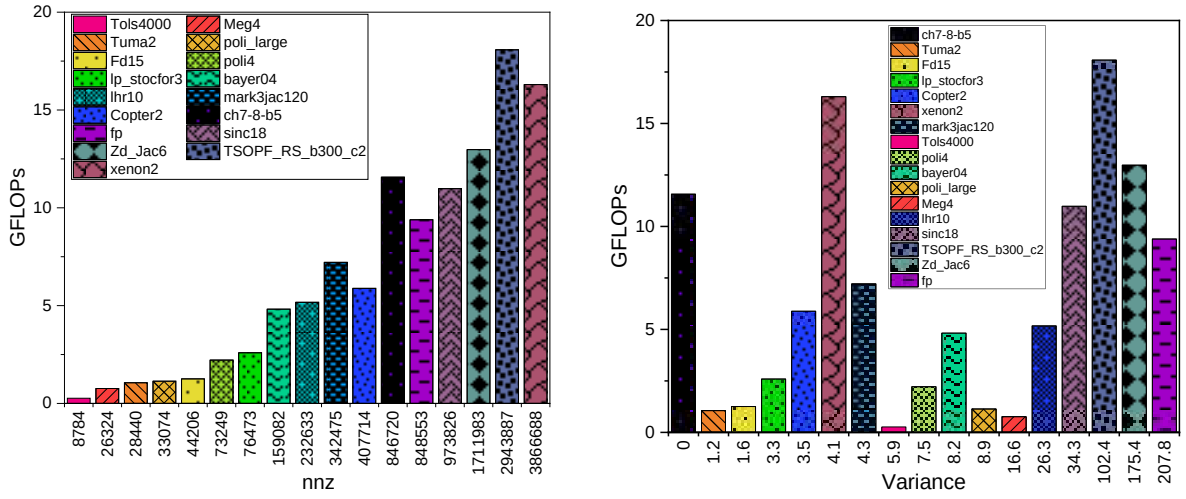
Figure 16. Compressed sparse row 5 (CSR5) execution time against: (**a**) *nnz* and (**b**) *npr variance*.

Figure 16b plots a comparison between execution time and *variance* for the CSR5 scheme. We observe a variation in the *variance* values and it tends to have longer execution times with the larger *variance* values. The matrices *mark3jac120* and *xenon2* with *variance* values 4.36 and 4.11, respectively, are exceptions. Although *mark3jac120* has larger *variance* than *xenon2*, it achieves better execution time due to its smaller *nnz* value compared to *xenon2*. The same happens for *fp* and *Zd_Jac6* because *Zd_Jac6* has a bigger *nnz* than *fp*. This again be observed again in case of *mark3jac120* and *bayer04*, with the latter's superiority due to its smaller *nnz* value. All these examples are evidence of the bigger impact *nnz* values has on the execution time than the impact *variance* of *nnz* values has on the execution time for CSR5.

6.2. GPU Throughput

Figure 17a,b illustrate GPU throughput of the CSR5 scheme with a comparison against the *nnz* value and the *variance* of each matrix. In Figure 17a, there is a kind of regular increase of attained GFLOPS as the *nnz* value increased with some exceptions. Examples of those exceptions include the *mark3jac120* and copter2 matrices. Although copter2 has a higher *nnz* value than *mark3jac120*, it has

fewer GFLOPS. This is due to *mark3jac120*'s structured pattern, while copter2 has an unstructured pattern. Another example is *fp* and *ch7-8-b5*; both are considered unstructured matrices and it is exception case of expectation. This indicates the impact of other sparsity features of the sparse matrices as shown in Figure 17b with *variance* feature.



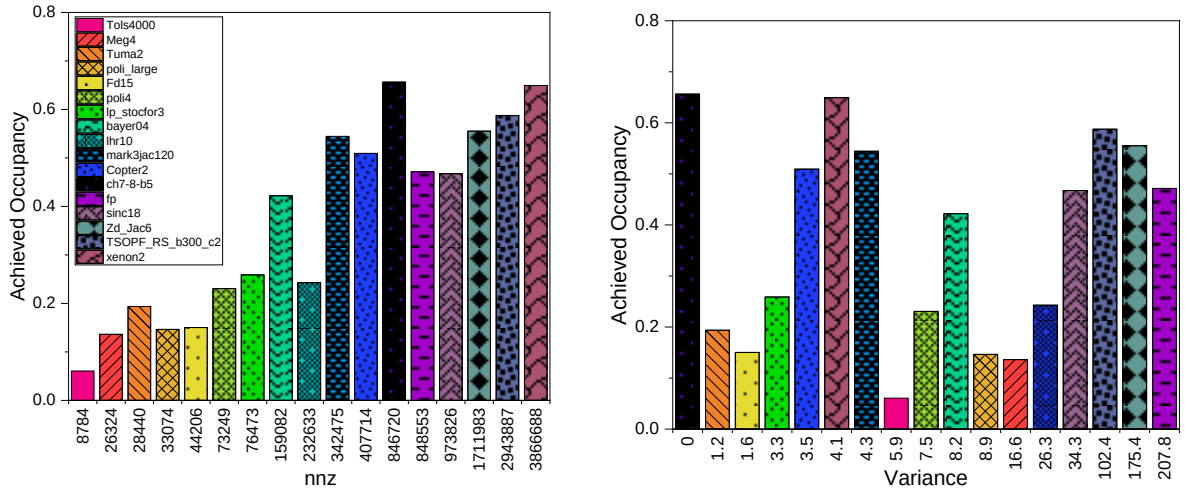
(**a**) (**b**)

Figure 17. CSR5 GFLOPs against: (**a**) *nnz* and (**b**) *npr variance*.

Regarding the impact of the matrix *variance* and GPU GFLOPS in Figure 17b, there is no direct relation between them from the diagram. However, we can generally say lower *variance* values have better GFLOPS when conditions make available a good sparsity pattern and higher *nnz* value. If we look at the same example, *fp* and *ch7-8-b5*, we see that the last has 0 *variance* where *fp* has 207.84 *variance* value, and this clarifies the reason why cha7-8-b5 exceeds *fp* on the attained GFLOPS despite having a smaller *nnz* value. In general, as Figure 17b shows, the impact of *variance* on the attained GFLOPS varies among all matrices. In addition, the attained GFLOPS in the CSR5 scheme is considered weak in general. Let us consider more examples to analyze the reasons. In the case of *TSOPF_RS_b300_c2* and *xenon2*, both have higher *nnz* values among the dataset and have big differences in their *variance* values, but they have attained approximate GFLOPS. Although the lower *nnz TSOPF_RS_b300_c2* matrix has a relatively big *variance* value compared to *xenon2*, it exceeds *xenon2* on the attained GFLOPS! That is because the *anpr* value of *TSOPF_RS_b300_c2* (i.e., 103) is bigger than *xenon2* (i.e., 24). The same happened in many cases that achieved the bigger *nnz* condition, as between *Zd_Jac6* and *ch7-8-b5* and between *Zd_Jac6* and *sinc18*. Thus, GFLOPS in the CSR5 scheme was mainly affected by *nnz* value, then by *variance* value, and then by *anpr* number, such that a large *nnz* value led to high GFLOPS. Smaller *variance* led to higher GFLOPS with some exceptions due to the existence of higher *nnz* value or higher *anpr* number where they have bigger impact than *variance* number.

6.3. GPU Utilization

Figure 18a,b report the results of the achieved occupancy of the CSR5 scheme. Generally, from Figure 18a, a higher *nnz* value leads to better achieved occupancy, but what are the reasons for opposite cases? Let us study copter2 compared to *mark3jac120*, *lhr10* compared to *bayer04*, and *fp* compared to *ch7-8-b5*. Copter2, *lhr10*, and *fp* matrices have higher *nnz* values than their compared matrices. Therefore, this indicates the impact of sparsity features on the achieved occupancy, as shown in Figure 18b.



(**a**) (**b**)

Figure 18. CSR5 achieved occupancy against: (**a**) *nnz* and (**b**) *npr variance*.

Figure 18b illustrates the effect of *variance* values on the achieved occupancy metric. We tend to have higher achieved occupancy with lower *variance* values and lower achieved occupancy with higher *variance* values, with exceptions. Let us discuss the same examples starting with copter2 compared to *mark3jac120* with 3.55 and 4.36 *variance* values, respectively. We see that copter2 has less *variance* and a higher *nnz* value than *mark3jac120* and has achieved lower achieved occupancy because *mark3jac120* is a structured matrix while copter2 not. However, in the other cases, such as in *lhr10* compared to *bayer04* and *fp* compared to *ch7-8-b5*, the matrices with lower *variance* (i.e., *bayer04* and *ch7-8-b5*) have attained higher achieved occupancy although they have smaller *nnz* values. We can conclude that higher *nnz* value besides lower *variance* value is a good combination to obtain high achieved occupancy in CSR5.

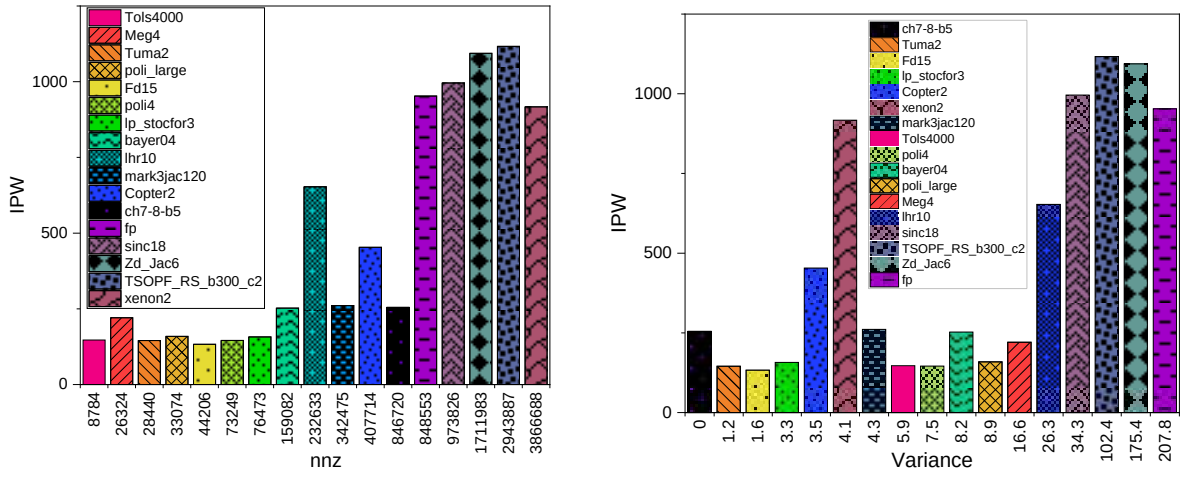
Figure 19a,b show the IPW results for the CSR5 scheme. This metric acts as an indicator of the existence of the divergence, where lower IPW values indicate lower divergence and therefore better performance [88].

Figure 19a plots the impact of *nnz* on the attained IPW. In this case, there is a tendency to have higher IPW with higher nnz. Nevertheless, there are some exceptions, which indicates the impact of sparsity structure of the matrices on the IPW result. We see that *mark3jac120*, copter2, and *ch7-8-b5* have fewer IPW than *lhr10* although they have higher nnzs! This is because they have lower *variance* values than *lhr10*. We further explain this with the next figure where we look at the *variance* effect.

In Figure 19b, we can see as *variance* value increased the IPW values increased, which indicates that with higher *variance* value the possibility of having higher divergence is high. However, in some cases, such as *Zd_Jac6* and *fp*, *Zd_Jac6* has higher IPW due to its larger *nnz* value compared to *fp*, although the *variance* of *fp* is larger than *Zd_Jac6*. The same happened for *mark3jac120* and *xenon2*, where the last was higher than the first, and *tols4000* had lower IPW than *mark3jac120* and *xenon2* despite its *variance* being higher than them, but this was due to its smaller *nnz* value. So, we can conclude that the main factor that affects IPW is the *variance* value such that with more variation of *nnz* per row in a matrix we have higher divergence and therefore lower performance, taking into consideration the matrix size in terms of *nnz* value. In other words, a low *variance* value leads to less divergence by obtaining a lower IPW value. However, the opposite case happens because the matrices with higher *nnz* values require more instructions and therefore in those cases have stronger impact than *variance* value. In addition to the IPW metric to detect the existence of divergence, we have the warp execution efficiency metric. Therefore, in the case of warp divergence some threads will be permanently off for reasons such as the presence of conditional control statements (which cause MIMD) or uncoalesced memory access (which causes waiting time to bring the data in different memory transactions). These factors affect the active threads within a warp because they result in work variation within a warp, and this serializes the work due to the GPU being an SIMD processor.

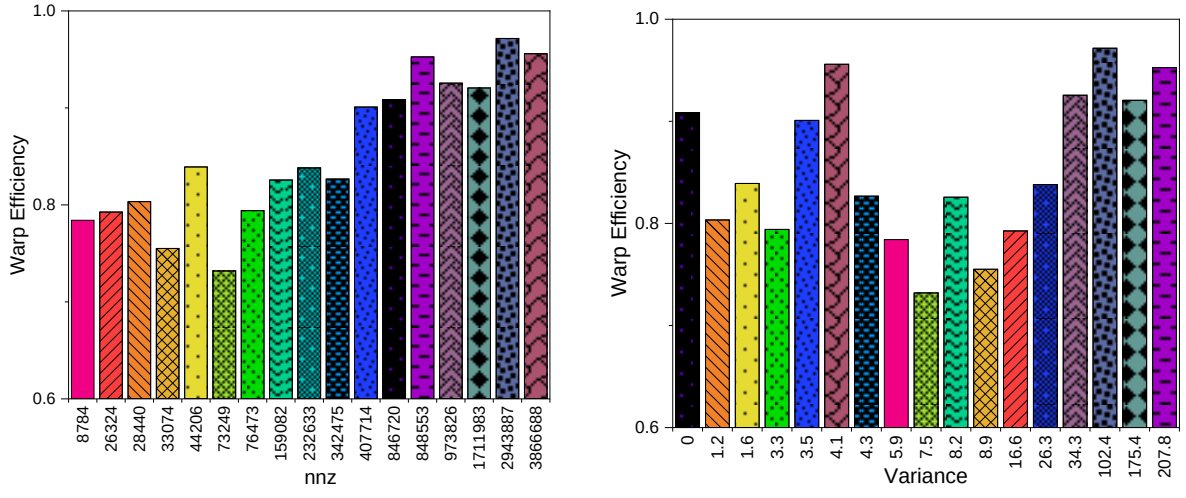
Thus, this metric is another deep study of GPU utilization to clarify whether we have exploited the highest possible parallelism or not. High warp execution efficiency percentage indicates a high level of thread exploitation, and low warp efficiency indicates divergence existence.

Figure 20a shows the impact of *nnz* on warp execution efficiency. We have in general a higher level of warp execution efficiency in this scheme, and there is a tendency to have an increased percentage of warp efficiency as the *nnz* value increases. Figure 20b shows a comparison between warp execution efficiency and matrix variance. There is oscillation between ups and downs over the *variance* axis. However, if we assume that a lower *variance* value is always the best case to obtain better warp efficiency, then it is better to study the opposite cases. Starting with *lp_stocfor3* and copter2 with 3.34 and 3.55 *variance* values, respectively, we find that copter2 has achieved higher warp efficiency than *lp_stocfor3* due to its higher *nnz* value compared to *lp_stocfor3*. The same happened with *bayer04* and *poli4*, as well as *sinc18* and *lhr10*. Thus, the main factor that affects the warp execution efficiency is *nnz* value such that a higher *nnz* gives higher warp efficiency and therefore less warp divergence occurrence with consideration of *variance* value.



(**a**) (**b**)

Figure 19. CSR5 instructions per warp (IPW) against: (**a**) *nnz* and (**b**) *npr variance*.



(**a**) (**b**)

Figure 20. CSR5 warp efficiency against: (**a**) *nnz* and (**b**) *npr variance*.

7. SpMV Performance on GPUs (Summary)

Table 2 summarizes our findings of the impact of sparsity features on the SpMV performance on GPUs for the four schemes. The performance involves the execution time, GPU utilization, and GPU throughput. Except for the *nnz* feature, it reports the effect of GPU utilization and GPU throughput and excludes the execution time since it is logically to have more execution time with a higher *nnz*. The SpMV computations performance on GPUs depends on many factors and we shall find a good configuration to obtain a high performance. Table 2 shows the various configuration possibilities of each scheme in terms of the sparsity features that would provide good performance. Some of the sparsity features have a high effect on some schemes while they have medium or low effect on others. Such as a high value of *npr variance* (the *variance* in the number of nonzero elements per row) will have a high effect on CSR and ELL; medium effect on HYB and low effect on CSR5. This is because in CSR and ELL the *variance* affects the contiguous memory access and causes thread divergence. In addition it creates surplus zero padding in ELL. The features *maxnpr* and *anpr* directly correlate with *npr variance* as the difference between the two features provides an average *variance* for all the rows. The zero paddings due to a higher *npr variance* is reduced in HYB due to a better partitioning of scheme, and CSR5 further optimizes the partition and improves the coalesced access and thread divergence.

Table 2. Performance of the four sparse storage scheme against the five matrix sparsity features.

	nnz	*npr variance*	*distavg*	*anpr*	*maxnpr*
CSR	medium	high	medium	medium	high
ELL	medium	high	low	medium	high
HYB	medium	medium	low	medium	medium
CSR5	medium	low	low	medium	low

8. The Proposed Scheme (HCGHYB)

In this section, we discuss our proposed scheme, the heterogeneous CPU–GPU hybrid (HCGHYB) scheme, which is an enhanced version of the HYB scheme. We explain the scheme in Section 8.1, followed by its detailed performance analysis in comparison with all four schemes in Section 8.2.

8.1. HCGHYB: Motivation and Description

The main idea of this scheme is to divide the work between the CPU and GPU to accelerate the computations. The motivation of this acceleration is to avoid using the atomic operation that is used on regular HYB in COO portion of the computation. The atomicAdd operation is used usually to avoid the race condition problem on the parallel environments. In COO portion multiple threads are assigned per row, whose values needs to be added to compute the partial sums in the SpMV process. Unfortunately, the regular HYB uses atomicAdd operation to accomplish this addition. However, atomic operations are kind of synchronization processes that prevent the interference between threads on multithreaded environment such as writing on the same memory location. But in this case the threads will be serialized which can be quite costly and result in slow down the performance. We avoid this issue by doing the multiplication process on the GPU in parallel while doing the addition part on the CPU. Multiplication is costly operation compared to addition so we do it on parallel on GPU. While on the other hand, the simpler operation (i.e., addition) is done on CPU. This result in better performance than HYB as shown on the following section.

Many challenges are involved in CPU–GPU heterogeneous computing [92]. They can by processing unit specific such as computational power of processing units, achieving load balance among the processing units, utilizing the memory bandwidth between CPUs and GPU, avoiding the overhead of launching the GPU kernels and extra data transfer, and architectural differences between various processing units. Other issues that need to be considered are pipelining data transfer between CPU and GPU with computation. We also need to consider the limitations of both processing units (CPU and GPU) in terms of memory, bandwidth, the number of GPU threads and CPU cores

(and threads per cores), etc. [92]. Other issues are more application-specific such as complexity and nature of algorithms, level of parallelism offered by the algorithm, best strategy for dividing the work among CPUs and GPUs and solving data dependency. The partitioning problem is one of the most challenging problems especially with regards to SpMV [93]. A combination of different SpMV kernels along with dynamic partitioning is required to achieve the best performance. The remaining challenges are based on the objective or attaining certain goals such as performance and energy saving.

8.2. HCGHYB: Performance Analysis

In this section we analyze the performance of HCGHYB compared to CSR, ELL, HYB, and CSR5 to depict the achieved level of optimization. The performance metrics considered are: execution time, GPU throughput in terms of GFLOPS, and GPU utilization (achieved occupancy, instructions per warp, and warp efficiency).

8.2.1. Execution Time

Figure 21 illustrates the achieved execution time of all notable schemes. We can see from the figure, the shortest time is attained by CSR5. This is due to improved load balancing among the threads and improved coalesced memory accesses. The next better scheme is HCGHYB and this is due to avoiding the latency of atomic operations that exists on HYB and dividing the work among the CPU and GPU processors. It is clear that our proposed scheme outperforms the HYB scheme by attaining less execution time for almost all the matrices. This is due to removing the serialization, which was caused by the atomic operation required by the HYB scheme. We achieve speedups of 375.75x and 54.53x in the best and average cases, respectively. There was a single case where HYB performed slightly better than our proposed HCGHYB scheme and resulted in HCGHYB providing a speedup of 0.99x. The average speedup is calculated using the following equation.

$$speedupavg = \frac{\sum_{i=0}^{16} HYB_i}{\sum_{i=0}^{16} HCGHYB_i} \quad (1)$$

On the other hand, the worst schemes are HYB and CSR while in some cases is ELL such as in *poli_large* and *poli4*. The relation of the performance of each scheme and the impact of the sparsity pattern is explained in detail in the previous sections. HCGHYB attains an average execution time of 0.264 seconds compared to 14.39 (HYB), 0.56 (ELL), 0.88 (CSR), and 0.14 (CSR5) seconds.

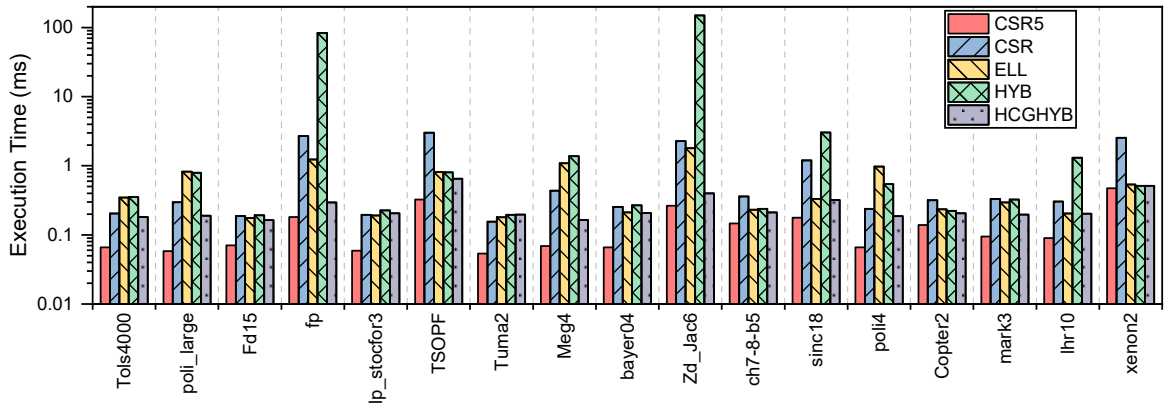


Figure 21. HCGHYB execution time compared to CSR, ELL, HYB, and CSR5.

8.2.2. GPU Throughput

Figure 22 shows the GPU throughput comparison among all the schemes. The highest single GFLOPS among all schemes for all the matrices is for CSR5. The next best scheme is HCGHYB. While the worst GFLOPS is achieved by HYB then by ELL, while CSR is in the middle between

the best and worst cases, except in some cases such as when computing *copter2* and *mark3jac120*. By removing the synchronization caused by atomic operations, we successfully increase the attained GFLOPS compared to HYB. We achieve GFLOPS gain of 375.57x and 1.7x in the best and average cases, respectively. There was a single case where HYB performed slightly better than our proposed HCGHYB scheme and resulted in HCGHYB providing a GFLOPS of 0.99x. The values below 1 indicate that HYB performed better than our scheme. We note that in *xenon2* matrix case the throughput is the same for both schemes, this is due to the fact that the whole matrix is processed as ELL (no COO part), so our optimization does not affect its performance. The average GFLOPS gain is calculated using the following equation:

$$GFLOPS\_gain_{avg} = \frac{\sum_{i=0}^{16} HCGHYB_i}{\sum_{i=0}^{16} HYB_i} \quad (2)$$

HCGHYB attains an average throughput of 3.94 GFLOPs compared to 2.31 (HYB), 2.71 (ELL), 1.38 (CSR), and 6.9 (CSR5) GFLOPS.

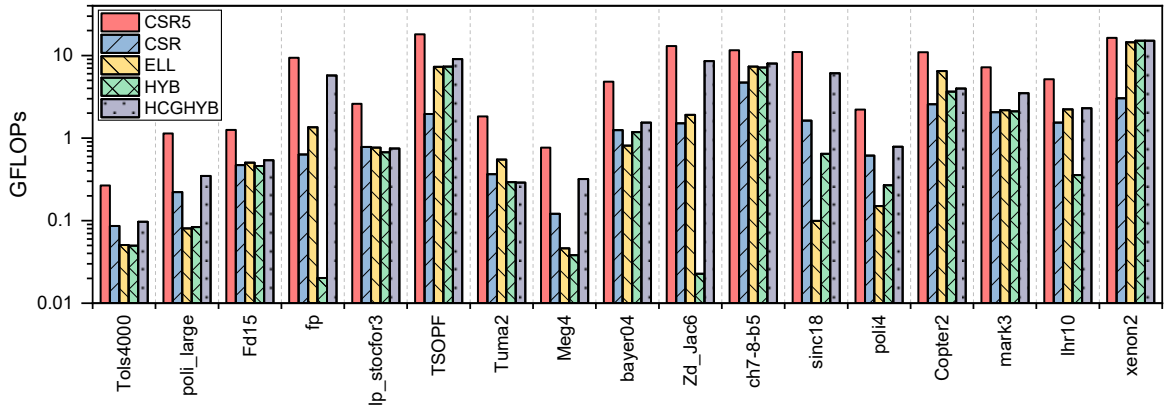


Figure 22. Heterogeneous CPU–GPU hybrid (HCGHYB) GPU throughput compared to CSR, ELL, HYB, and CSR5.

8.2.3. GPU Utilization

Figure 23 reports the achieved occupancy for all the notable schemes among the discussed matrices. Avoiding the usage of atomic operation in HCGHYB results in a higher occupancy than HYB in most cases. We attain an achieved occupancy gain of 1.3x and 1.01x in the best and average cases, respectively. There were a few cases where HYB performed slightly better than our proposed HCGHYB scheme and resulted in HCGHYB providing an achieved occupancy of 0.72x. The average achieved occupancy gain is calculated using the following equation:

$$AchivedOccup\_gain_{avg} = \frac{\sum_{i=0}^{16} HCGHYB_i}{\sum_{i=0}^{16} HYB_i} \quad (3)$$

The best achieved occupancy has been attained by HCGHYB scheme with a competition in some cases with ELL and sometimes with HYB. The worst occupancy has been achieved by CSR5 then by CSR. HCGHYB attains an average achieved occupancy of 0.61 compared to 0.606 (HYB), 0.604 (ELL), 0.42 (CSR), and 0.36 (CSR5).

Figure 24 shows the instructions per warp metric for the compared schemes. And as we explained previously a high IPW indicates presence of divergence. We can see from the figure in most cases the HCGHYB has less IPW than HYB which indicates that our proposed scheme also has less divergence level than HYB. We achieve better IPW of 21.65x and 1.6x in the best and average cases, respectively.

There were some cases where HYB performed slightly better than our proposed HCGHYB scheme and resulted in HCGHYB providing 0.97x. The average IPW gain is calculated using the following equation:

$$IPW\_gain_{avg} = \frac{\sum_{i=0}^{16} HCGHYB_i}{\sum_{i=0}^{16} HYB_i} \tag{4}$$

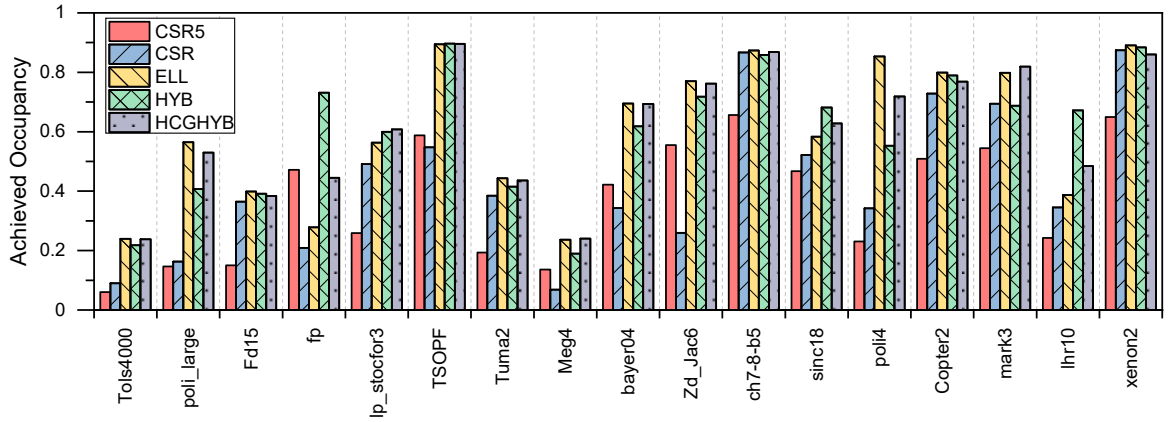


Figure 23. HCGHYB achieved occupancy compared to CSR, ELL, HYB, and CSR5.

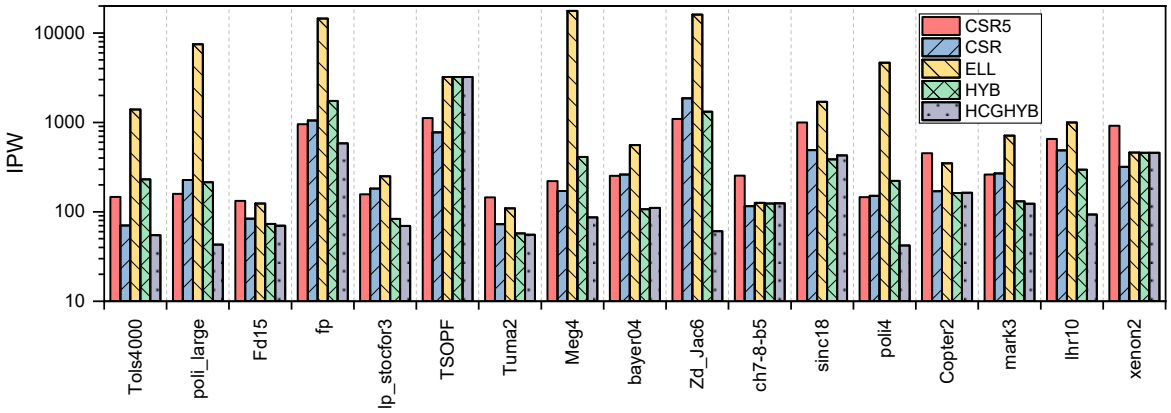


Figure 24. HCGHYB IPW compared to CSR, ELL, HYB, and CSR5.

We can see that the lowest IPW has been achieved by HCGHYB and this explains why it has the best achieved occupancy. In some cases there is a competition between HCGHYB and HYB. The worst case is given by ELL scheme since it has the highest IPW almost among all the matrices. The second worst scheme is CSR5. This indicates that lower divergence rate can be attained by HCGHYB then ELL and HYB while CSR and CSR5 have a higher level of divergence. HCGHYB attains an average IPW of 340 compared to 562 (HYB), 4140 (ELL), 397 (CSR), and 474 (CSR5).

Figure 25 shows the warp execution efficiency results for compared schemes. As shown HCGHYB has achieved better warp efficiency than HYB in most cases. We attain warp execution efficiency gain of 1.76x and 1.2x in the best and average cases, respectively. The worst case is 1.00x which indicates both schemes performed equally. The average warp execution efficiency gain is calculated using the following equation.

$$WE\_gain_{avg} = \frac{\sum_{i=0}^{16} HCGHYB_i}{\sum_{i=0}^{16} HYB_i} \tag{5}$$

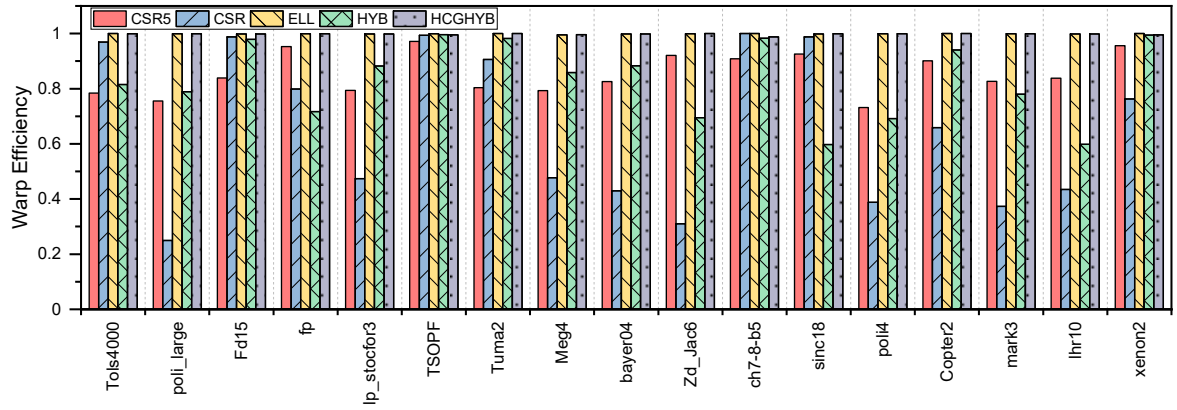


Figure 25. HCGHYB execution efficiency compared to CSR, ELL, HYB, and CSR5.

Figure 25 shows the warp execution efficiency which is a metric with high percentage indicates to high level of threads exploitation and low warp efficiency indicates to divergence existence. Consequently, we can conclude that the existence of the atomic operations areas on of having high divergence rate by having less warp efficiency and more instructions per warp. HCGHYB attains an average WE of 99.79% compared to 83.43% (HYB), 99.91% (ELL), 65.90% (CSR), and 85.45% (CSR5).

9. Conclusions and Future Work

SpMV is fundamental to many scientific, engineering, business, and social applications, such as CFD, electromagnetics, economics, and machine and deep learning. The University of Florida repository of sparse matrices comprises thousands of matrices that are collected from tens of application domains. Also, SpMV is a core operation in finding the iterative solution of sparse linear equation systems.

A great variety of specialized data structures and algorithms have been devised over the years for SpMV computations on the mainstream processor architectures. However, due to the extreme variations in the sparsity structure of matrices, parallelizing and loadbalancing SpMV computations on multiple cores or compute nodes, and the associated memory access challenges limit SpMV performance. No single scheme provides consistent and sufficiently high performance on any processor architectures. An extensive review of SpMV techniques on GPUs shows that the performance of SpMV techniques on GPUs has not been studied in sufficient detail. SpMV computations are mainly being evaluated and compared for their SpMV throughput in FLOPS, which alone does not provide a deep insight into the SpMV performance.

In this paper, we provided a detailed performance analysis of SpMV performance on GPUs using four notable sparse matrix storage schemes (CSR, ELL, HYB, and CSR5), five performance metrics (execution time, GFLOPS, achieved occupancy, instructions per warp, and warp execution efficiency), five matrix sparsity features (*nnz*, *anpr*, *npr variance*, *maxnpr*, and *distavg*), and 17 sparse matrices from 10 application domains (chemical simulations, CFD, electromagnetics, linear programming, economics, etc.). To the best of our knowledge, this is the first work where the SpMV performance on GPUs has been discussed in such depth.

Subsequently, we proposed the heterogeneous CPU–GPU Hybrid (HCGHYB) scheme that utilizes both the CPU and GPU in parallel to improve the performance. Specifically, we multiply the matrix and vector elements on GPU, while the *atomic reduce operation* in SpMV computation is performed on CPU. Since the *atomic reduce operation* is a compute-intensive operation, our scheme HCGHYB provides better performance over the standard HYB format by an average speedup of 1.7x. The HYB scheme is a popular choice in many SpMV and iterative solvers for sparse linear equation systems and therefore our proposed scheme is expected to generate high impact.

The detailed performance analysis provided in this paper has helped us to understand the performance bottlenecks of the HYB and other schemes and we were able to devise a scheme to

improve the performance of the HYB scheme. Heterogeneous computing that involves multiple processor and computing architectures is an important direction for SpMV and other application areas. We believe that this work on SpMV performance analysis and the heterogeneous scheme will open up many new directions and improvements for the SpMV computing field in the future.

The directions for future work are many. The knowledge gained from the current performance analysis shows that the compute kernels in the existing schemes such as HYB and CSR5 can be parallelized on GPUs using multiple SMs or on heterogeneous architectures. Our proposed scheme that uses both CPU and GPU can be improved by improved parallelization on the GPU and CPU cores. Moreover, in the future, we plan to improve the breadth, depth, and quality of our performance analysis work on GPU and other architectures. This would help us to further understand the performance bottlenecks of the existing SpMV schemes and the performance profiles of various processor architectures.

Author Contributions: Conceptualization, S.A., T.M., and R.M.; methodology, S.A., T.M., and R.M.; software, S.A.; validation, S.A., T.M., and R.M.; formal analysis, S.A., T.M., R.M., I.K., and A.A.; investigation, S.A., T.M., and R.M.; resources, R.M., I.K., and A.A.; data curation, S.A.; writing—original draft preparation, S.A., T.M., and R.M.; writing—review and editing, R.M.; visualization, S.A. and T.M.; supervision, R.M.; project administration, R.M., A.A., and I.K.; funding acquisition, R.M., A.A., and I.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Deanship of Scientific Research (DSR) at the King Abdulaziz University, Jeddah, under grant number RG-10-611-38.

Acknowledgments: The authors acknowledge with thanks the technical and financial support from the Deanship of Scientific Research (DSR) at the King Abdulaziz University (KAU), Jeddah, Saudi Arabia, under Grant No. RG-10-611-38. The experiments reported in this paper were performed on the Aziz supercomputer at KAU.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Asanovic, K.; Bodik, R.; Catanzaro, B.C.; Gebis, J.J.; Husbands, P.; Keutzer, K.; Patterson, D.A.; Plishker, W.L.; Shalf, J.; Williams, S.W.; et al. *The Landscape of Parallel Computing Research: A View from Berkeley*; Technical Report UCB/EECS-2006-183; EECS Department, University of California: Berkeley, CA, USA, 2006.
2. Davis, T.A.; Hu, Y. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.* **2011**, *38*, 1:1–1:25. [CrossRef]
3. Yang, W.; Li, K.; Li, K. A hybrid computing method of SpMV on CPU–GPU heterogeneous computing systems. *J. Parallel Distrib. Comput.* **2017**, *104*, 49–60. [CrossRef]
4. Huan, G.; Qian, Z. A new method of Sparse Matrix-Vector Multiplication on GPU. In Proceedings of the 2012 2nd International Conference on Computer Science and Network Technology, Changchun, China, 29–31 December 2012. [CrossRef]
5. Hassani, R.; Fazely, A.; Choudhury, R.U.A.; Luksch, P. Analysis of Sparse Matrix-Vector Multiplication Using Iterative Method in CUDA. In Proceedings of the 2013 IEEE Eighth International Conference on Networking, Architecture and Storage, Xi'an, China, 17–19 July 2013. [CrossRef]
6. Guo, P.; Wang, L. Auto-Tuning CUDA Parameters for Sparse Matrix-Vector Multiplication on GPUs. In Proceedings of the 2010 International Conference on Computational and Information Sciences, Chengdu, China, 17–19 December 2010. [CrossRef]
7. Merrill, D.; Garland, M. Merge-Based Parallel Sparse Matrix-Vector Multiplication. In Proceedings of the SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 13–18 November 2016. [CrossRef]
8. Ahamed, A.K.C.; Magoulès, F. Efficient implementation of Jacobi iterative method for large sparse linear systems on graphic processing units. *J. Supercomput.* **2016**, *73*, 3411–3432. [CrossRef]
9. Hou, K.; Feng, W.-C.; Che, S. Auto-Tuning Strategies for Parallelizing Sparse Matrix-Vector (SpMV) Multiplication on Multi- and Many-Core Processors. In Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, USA, 29 May–2 June 2017. [CrossRef]

10. Langville, A.N.; Meyer, C.D. A survey of eigenvector methods for web information retrieval. *SIAM Rev.* **2005**, *47*, 135–161. [CrossRef]
11. Kamvar, S.D.; Haveliwala, T.H.; Manning, C.D.; Golub, G.H. Extrapolation methods for accelerating PageRank computations. In *Proceedings of the 12th International Conference on World Wide Web*; ACM: New York, NY, USA, 2003; pp. 261–270.
12. Heffes, H.; Lucantoni, D. A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance. *IEEE J. Sel. Areas Commun.* **1986**, *4*, 856–868. [CrossRef]
13. Bylina, J.; Bylina, B.; Karwacki, M. An efficient representation on GPU for transition rate matrices for Markov chains. In *Parallel Processing and Applied Mathematics*; Springer: Berlin, Germany, 2013.
14. Bylina, J.; Bylina, B.; Karwacki, M. A Markovian Model of a Network of Two Wireless Devices. *Comput. Netw.* **2012**. [CrossRef]
15. Ahamed, A.K.C.; Magoules, F. Fast sparse matrix-vector multiplication on graphics processing unit for finite element analysis. In Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, Liverpool, UK, 25–27 June 2012. [CrossRef]
16. Yu, J.; Lukefahr, A.; Palframan, D.; Dasika, G.; Das, R.; Mahlke, S. Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*; Association for Computing Machinery: New York, NY, USA, 2017; pp. 548–560. [CrossRef]
17. Mohammed, T.; Joe-Wong, C.; Babbar, R.; Francesco, M.D. Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; pp. 854–863. [CrossRef]
18. Benatia, A.; Ji, W.; Wang, Y.; Shi, F. BestSF: A Sparse Meta-Format for Optimizing SpMV on GPU. *ACM Trans. Archit. Code Optim.* **2018**, *15*. [CrossRef]
19. Abdali, S.K.; Wise, D.S. Experiments with quadtree representation of matrices. In Proceedings of the Symbolic and Algebraic Computation International Symposium ISSAC '88, Rome, Italy, 4–8 July 1988; Springer: Berlin/Heidelberg, Germany, 1989; pp. 96–108. [CrossRef]
20. Langr, D.; Simecek, I.; Tvrdik, P. Storing sparse matrices to files in the adaptive-blocking hierarchical storage format. In Proceedings of the 2013 Federated Conference on Computer Science and Information Systems (FedCSIS), Krakow, Poland, 8–11 September 2013; pp. 479–486.
21. Simecek, I.; Langr, D.; Tvrdík, P. Space efficient formats for structure of sparse matrices based on tree structures. In Proceedings of the 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 23–26 September 2013; pp. 344–351.
22. Simecek, I.; Langr, D. Tree-based space efficient formats for storing the structure of sparse matrices. *Scalable Comput. Pract. Exp.* **2014**, *15*, 1–20.
23. Zhang, J.; Wan, J.; Li, F.; Mao, J.; Zhuang, L.; Yuan, J.; Liu, E.; Yu, Z. Efficient sparse matrix–vector multiplication using cache oblivious extension quadtree storage format. *Future Gener. Comput. Syst.* **2016**, *54*, 490–500.
24. Meyer, J.C.; Natvig, L.; Karakasis, V.; Siakavaras, D.; Nikas, K. Energy-efficient Sparse Matrix Auto-tuning with CSX. In Proceedings of the 27th IEEE International Parallel & Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), Cambridge, MA, USA, 20–24 May 2013.
25. Elafrou, A.; Goumas, G.I.; Koziris, N. A lightweight optimization selection method for Sparse Matrix-Vector Multiplication. *CoRR* **2015**, arXiv:1511.02494.
26. Shaikh, M.A.H.; Hasan, K.M.A. Efficient storage scheme for n-dimensional sparse array: GCRS/GCCS. In Proceedings of the 2015 International Conference on High Performance Computing Simulation (HPCS), Amsterdam, The Netherlands, 20–24 July 2015; pp. 137–142. [CrossRef]
27. Martone, M.; Filippone, S.; Tucci, S.; Paprzycki, M.; Ganzha, M. Utilizing Recursive Storage in Sparse Matrix-Vector Multiplication-Preliminary Considerations. In *CATA*; ISCA: Honolulu, HI, USA, 2010; pp. 300–305.
28. Martone, M. Efficient multithreaded untransposed, transposed or symmetric sparse matrix–vector multiplication with the recursive sparse blocks format. *Parallel Comput.* **2014**, *40*, 251–270. [CrossRef]
29. Guo, D.; Gropp, W. Applications of the streamed storage format for sparse matrix operations. *Int. J. High Perform. Comput. Appl.* **2014**, *28*, 3–12. [CrossRef]

30. Bakos, J.D.; Nagar, K.K. Exploiting Matrix Symmetry to Improve FPGA-Accelerated Conjugate Gradient. In *Proceedings of the 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*; IEEE Computer Society: Washington, DC, USA, 2009; pp. 223–226. [CrossRef]
31. Grigoras, P.; Burovskiy, P.; Hung, E.; Luk, W. Accelerating SpMV on FPGAs by Compressing Nonzero Values. In *Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*; IEEE Computer Society: Washington, DC, USA, 2015; pp. 64–67. [CrossRef]
32. Boland, D.; Constantinides, G.A. Optimizing Memory Bandwidth Use and Performance for Matrix-vector Multiplication in Iterative Methods. *ACM Trans. Reconfigurable Technol. Syst.* **2011**, *4*, 22:1–22:14. [CrossRef]
33. Kestur, S.; Davis, J.D.; Chung, E.S. Towards a Universal FPGA Matrix-Vector Multiplication Architecture. In *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*; IEEE Computer Society: Washington, DC, USA, 2012; pp. 9–16. [CrossRef]
34. deLorimier, M.; DeHon, A. Floating-point Sparse Matrix-vector Multiply for FPGAs. In *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*; ACM: New York, NY, USA, 2005; pp. 75–85. [CrossRef]
35. Dorrance, R.; Ren, F.; Marković, D. A Scalable Sparse Matrix-vector Multiplication Kernel for Energy-efficient Sparse-blas on FPGAs. In *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*; ACM: New York, NY, USA, 2014; pp. 161–170. [CrossRef]
36. Grigoraş, P.; Burovskiy, P.; Luk, W.; Sherwin, S. Optimising Sparse Matrix Vector multiplication for large scale FEM problems on FPGA. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–9. [CrossRef]
37. Kuzmanov, G.; Taouil, M. Reconfigurable sparse/dense matrix-vector multiplier. In Proceedings of the 2009 International Conference on Field-Programmable Technology, Sydney, Australia, 9–11 December 2009; pp. 483–488. [CrossRef]
38. Yan, S.; Li, C.; Zhang, Y.; Zhou, H. *yaSpMV: Yet Another SpMV Framework on GPUs*; ACM SIGPLAN Notices; ACM: New York, NY, USA, 2014; Volume 49, pp. 107–118.
39. Liu, W.; Vinter, B. CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. In *Proceedings of the 29th ACM on International Conference on Supercomputing*; ACM: New York, NY, USA, 2015; pp. 339–350. [CrossRef]
40. Liu, X.; Smelyanskiy, M.; Chow, E.; Dubey, P. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*; ACM: New York, NY, USA, 2013; pp. 273–282.
41. Saule, E.; Kaya, K.; Çatalyürek, Ü.V. Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi. In *Parallel Processing and Applied Mathematics, Proceedings of the 10th International Conference, PPAM 2013, Warsaw, Poland, 8–11 September 2013*; Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J., Eds.; Revised Selected Papers, Part I; Springer: Berlin/Heidelberg, Germany, 2014; pp. 559–570. [CrossRef]
42. Kreutzer, M.; Hager, G.; Wellein, G.; Fehske, H.; Bishop, A.R. A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. *SIAM J. Sci. Comput.* **2014**, *36*, C401–C423. [CrossRef]
43. Yzelman, A.N. Generalised Vectorisation for Sparse Matrix: Vector Multiplication. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*; ACM: New York, NY, USA, 2015; pp. 6:1–6:8. [CrossRef]
44. Tang, W.T.; Zhao, R.; Lu, M.; Liang, Y.; Huynh, H.P.; Li, X.; Goh, R.S.M. Optimizing and Auto-tuning Scale-free Sparse Matrix-vector Multiplication on Intel Xeon Phi. In *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*; IEEE Computer Society: Washington, DC, USA, 2015; pp. 136–145.
45. Cheng, J.R.; Gen, M. Accelerating genetic algorithms with GPU computing: A selective overview. *Comput. Ind. Eng.* **2019**, *128*, 514–525. [CrossRef]
46. Jeon, M.; Venkataraman, S.; Phanishayee, A.; Qian, J.; Xiao, W.; Yang, F. Analysis of large-scale multi-tenant {GPU} clusters for {DNN} training workloads. In Proceedings of the 2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19), Renton, WA, USA, 10 July 2019; pp. 947–960.
47. Aqib, M.; Mehmood, R.; Alzahrani, A.; Katib, I.; Albeshri, A.; Altowaijri, S.M. Smarter Traffic Prediction Using Big Data, In-Memory Computing, Deep Learning and GPUs. *Sensors* **2019**, *19*, 2206. [CrossRef]

48. Aqib, M.; Mehmood, R.; Alzahrani, A.; Katib, I.; Albeshri, A.; Altowaijri, S.M. Rapid Transit Systems: Smarter Urban Planning Using Big Data, In-Memory Computing, Deep Learning, and GPUs. *Sustainability* **2019**, *11*, 2736. [CrossRef]
49. Magoulès, F.; Ahamed, A.K.C. Alinea: An Advanced Linear Algebra Library for Massively Parallel Computations on Graphics Processing Units. *Int. J. High Perform. Comput. Appl.* **2015**, *29*, 284–310. [CrossRef]
50. Muhammed, T.; Mehmood, R.; Albeshri, A.; Katib, I. UbeHealth: A Personalized Ubiquitous Cloud and Edge-Enabled Networked Healthcare System for Smart Cities. *IEEE Access* **2018**, *6*, 32258–32285. [CrossRef]
51. Kirk, D.B.; Hwu, W.M.W. *Programming Massively Parallel Processors: A Hands-on Approach*, 1st ed.; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2010.
52. Owens, J.; Houston, M.; Luebke, D.; Green, S.; Stone, J.; Phillips, J. GPU Computing. *Proc. IEEE* **2008**, *96*, 879–899. [CrossRef]
53. Fevgas, A.; Daloukas, K.; Tsompanopoulou, P.; Bozanis, P. Efficient solution of large sparse linear systems in modern hardware. In Proceedings of the 2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA), Corfu, Greece, 6–8 July 2015. [CrossRef]
54. Nisa, I.; Siegel, C.; Rajam, A.S.; Vishnu, A.; Sadayappan, P. Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs. In Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, Canada, 21–25 May 2018; pp. 1056–1065. [CrossRef]
55. Filippone, S.; Cardellini, V.; Barbieri, D.; Fanfarillo, A. Sparse Matrix-Vector Multiplication on GPGPUs. *ACM Trans. Math. Softw.* **2017**, *43*, 1–49. [CrossRef]
56. Bell, N.; Garland, M. *Efficient Sparse Matrix-Vector Multiplication on CUDA*; Techreport NVR-2008-004; Nvidia Corporation: Santa Clara, CA, USA, 2008.
57. Choi, J.W.; Singh, A.; Vuduc, R.W. Model-driven Autotuning of Sparse Matrix-vector Multiply on GPUs. *SIGPLAN Not.* **2010**, *45*, 115–126. [CrossRef]
58. Flegar, G.; Anzt, H. Overcoming Load Imbalance for Irregular Sparse Matrices. In *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms*; ACM: New York, NY, USA, 2017; pp. 2:1–2:8. [CrossRef]
59. Ashari, A.; Sedaghati, N.; Eisenlohr, J.; Parthasarathy, S.; Sadayappan, P. Fast Sparse Matrix-vector Multiplication on GPUs for Graph Applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*; IEEE Press: Piscataway, NJ, USA, 2014; pp. 781–792. [CrossRef]
60. Su, B.Y.; Keutzer, K. clSpMV: A Cross-Platform OpenCL SpMV Framework on GPUs. In *Proceedings of the 26th ACM International Conference on Supercomputing*; ACM: New York, NY, USA, 2012; pp. 353–364. [CrossRef]
61. Guo, P.; Wang, L.; Chen, P. A Performance Modeling and Optimization Analysis Tool for Sparse Matrix-Vector Multiplication on GPUs. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 1112–1123. [CrossRef]
62. Li, J.; Tan, G.; Chen, M.; Sun, N. SMAT: An Input Adaptive Auto-tuner for Sparse Matrix-vector Multiplication. *SIGPLAN Not.* **2013**, *48*, 117–126. [CrossRef]
63. Sedaghati, N.; Mu, T.; Pouchet, L.N.; Parthasarathy, S.; Sadayappan, P. Automatic Selection of Sparse Matrix Representation on GPUs. In *Proceedings of the 29th ACM on International Conference on Supercomputing*; ACM: New York, NY, USA, 2015; pp. 99–108. [CrossRef]
64. Benatia, A.; Ji, W.; Wang, Y.; Shi, F. Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU. In Proceedings of the 2016 45th International Conference on Parallel Processing (ICPP), Philadelphia, PA, USA, 16–19 August 2016; pp. 496–505. [CrossRef]
65. Li, K.; Yang, W.; Li, K. Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 196–205. [CrossRef]
66. Kwiatkowska, M.; Parker, D.; Zhang, Y.; Mehmood, R. Dual-Processor Parallelisation of Symbolic Probabilistic Model Checking. In *Proceedings of the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*; IEEE Computer Society: Washington, DC, USA, 2004; pp. 123–130.

67. Mehmood, R.; Parker, D.; Kwiatkowska, M. *An Efficient BDD-Based Implementation of Gauss-Seidel for CTMC Analysis*; Technical Report CSR-03-13; School of Computer Science, University of Birmingham: Birmingham, UK, 2003.
68. Mehmood, R.; Crowcroft, J. *Parallel Iterative Solution Method for Large Sparse Linear Equation Systems*; Technical Report UCAM-CL-TR-650; University of Cambridge, Computer Laboratory: Cambridge, UK, 2005.
69. Mehmood, R.; Crowcroft, J.; Elmirghani, J.M.H. A Parallel Implicit Method for the Steady-State Solution of CTMCs. In Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation, Monterey, CA, USA, 11–14 September 2006; pp. 293–302.
70. Mehmood, R.; Lu, J.A. Computational Markovian Analysis of Large Systems. *J. Manuf. Technol. Manag.* **2011**, *22*, 804–817. [CrossRef]
71. Usman, S.; Mehmood, R.; Katib, I.; Albeshri, A.; Altowaijri, S. ZAKI: A Smart Method and Tool for Automatic Performance Optimization of Parallel SpMV Computations on Distributed Memory Machines. *Mob. Networks Appl.* **2019**. [CrossRef]
72. Usman, S.; Mehmood, R.; Katib, I.; Albeshri, A. ZAKI+: A Machine Learning Based Process Mapping Tool for SpMV Computations on Distributed Memory Architectures. *IEEE Access* **2019**, *7*, 81279–81296. [CrossRef]
73. Alyahya, H.; Mehmood, R.; Katib, I. Parallel Sparse Matrix Vector Multiplication on Intel MIC: Performance Analysis. In *Smart Societies, Infrastructure, Technologies and Applications*; Mehmood, R., Bhaduri, B., Katib, I., Chlamtac, I., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 306–322.
74. Alyahya, H.; Mehmood, R.; Katib, I. Parallel Iterative Solution of Large Sparse Linear Equation Systems on the Intel MIC Architecture. In *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*; Mehmood, R., See, S., Katib, I., Chlamtac, I., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 377–407. [CrossRef]
75. Alzahrani, S.; Ikbal, M.R.; Mehmood, R.; Fayez, M.; Katib, I. Performance Evaluation of Jacobi Iterative Solution for Sparse Linear Equation System on Multicore and Manycore Architectures. In *Smart Societies, Infrastructure, Technologies and Applications*; Mehmood, R., Bhaduri, B., Katib, I., Chlamtac, I., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 296–305.
76. AlAhmadi, S.; Muhammed, T.; Mehmood, R.; Albeshri, A. Performance Characteristics for Sparse Matrix-Vector Multiplication on GPUs. In *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*; Mehmood, R., See, S., Katib, I., Chlamtac, I., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 409–426. [CrossRef]
77. Muhammed, T.; Mehmood, R.; Albeshri, A.; Katib, I. SURAA: A Novel Method and Tool for Loadbalanced and Coalesced SpMV Computations on GPUs. *Appl. Sci.* **2019**, *9*, 947. [CrossRef]
78. El-Gorashi, T.; Pranggono, B.; Mehmood, R.; Elmirghani, J. A Mirroring Strategy for SANs in a Metro WDM Sectioned Ring Architecture under Different Traffic Scenarios. *J. Opt. Commun.* **2008**, *29*, 89–97. [CrossRef]
79. Mehmood, R.; Alturki, R.; Zeadally, S. Multimedia applications over metropolitan area networks (MANs). *J. Netw. Comput. Appl.* **2011**, *34*, 1518–1529. [CrossRef]
80. Mehmood, R.; Graham, G. Big Data Logistics: A health-care Transport Capacity Sharing Model. *Procedia Comput. Sci.* **2015**, *64*, 1107–1114. [CrossRef]
81. Mehmood, R.; Meriton, R.; Graham, G.; Hennelly, P.; Kumar, M. Exploring the Influence of Big Data on City Transport Operations: A Markovian Approach. *Int. J. Oper. Prod. Manag.* **2017**, *37*, 75–104. [CrossRef]
82. El-Gorashi, T.E.H.; Pranggono, B.; Mehmood, R.; Elmirghani, J.M.H. A data Mirroring technique for SANs in a Metro WDM sectioned ring. In Proceedings of the 2008 International Conference on Optical Network Design and Modeling, Vilanova i la Geltru, Spain, 12–14 March 2008; pp. 1–6. [CrossRef]
83. Pranggono, B.; Mehmood, R.; Elmirghani, J.M.H. Performance Evaluation of a Metro WDM Multi-channel Ring Network with Variable-length Packets. In Proceedings of the 2007 IEEE International Conference on Communications, Glasgow, UK, 24–28 June 2007; pp. 2394–2399. [CrossRef]
84. Altowaijri, S.; Mehmood, R.; Williams, J. A Quantitative Model of Grid Systems Performance in Healthcare Organisations. In Proceedings of the 2010 International Conference on Intelligent Systems, Modelling and Simulation, Liverpool, UK, 27–29 January 2010; pp. 431–436.
85. Kwiatkowska, M.; Mehmood, R.; Norman, G.; Parker, D. A Symbolic Out-of-Core Solution Method for Markov Models. *Electron. Notes Theor. Comput. Sci.* **2002**, *68*, 589–604. [CrossRef]
86. Langr, D.; Tvrdik, P. Evaluation Criteria for Sparse Matrix Storage Formats. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 428–440. [CrossRef]

87. Abu-Sufah, W.; Karim, A.A. An Effective Approach for Implementing Sparse Matrix-Vector Multiplication on Graphics Processing Units. In *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*; IEEE Computer Society: Washington, DC, USA, 2012; pp. 453–460. [CrossRef]
88. *Professional CUDA C Programming*, 1st ed.; Wrox Press Ltd.: Birmingham, UK, 2014.
89. Profiler User's Guide, Available online: https://docs.nvidia.com/cuda/profiler-users-guide/index.html (accessed on 12 October 2020).
90. Saad, Y. SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations—Version 2. 1994. Available online: https://www-users.cs.umn.edu/~saad/software/SPARSKIT/ (accessed on 12 October 2020).
91. Grimes, R.G.; Kincaid, D.R.; Young, D.M. *ITPACK 2.0 User'S Guide*; Center for Numerical Analysis, The University of Texas at Austin: Austin, TX, USA, 1979.
92. Mittal, S.; Vetter, J.S. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comput. Surv.* **2015**, *47*. [CrossRef]
93. Benatia, A.; Ji, W.; Wang, Y.; Shi, F. Sparse matrix partitioning for optimizing SpMV on CPU-GPU heterogeneous platforms. *Int. J. High Perform. Comput. Appl.* **2020**, *34*, 66–80. [CrossRef]