Optimization of Sparse Matrix-Vector Multiplication Using Reordering Techniques on GPUs

Juan C. Pichel[a,*], Francisco F. Rivera[a], Marcos Fernández[b], Aurelio Rodríguez[b]

[a] *Electronics and Computer Science Dpt. Universidade de Santiago de Compostela, Spain*
[b] *Galicia Supercomputing Center (CESGA). Santiago de Compostela, Spain*

Abstract

It is well-known that reordering techniques applied to sparse matrices are common strategies to improve the performance of sparse matrix operations, and particularly, the sparse matrix vector multiplication (SpMV) on CPUs.

In this paper, we have evaluated some of the most successful reordering techniques on two different GPUs. In addition, in our study a number of sparse matrix storage formats were considered. Executions for both single and double precision arithmetics were also performed.

We have found that SpMV is very sensitive to the application of reordering techniques on GPUs. In particular, several characteristics of the reordered matrices that have a big impact on the SpMV performance have been detected. In most of the cases, reordered matrices outperform the original ones, showing noticeable speedups up to 2.6×. We have also observed that there is no one storage format preferred over the others.

Keywords: sparse matrix, optimization, GPUs, reordering, performance

1. Introduction

Sparse matrix-vector product (SpMV) is one of the most important computational kernels in scientific and engineering applications. It is notorious for sustaining low fractions of peak performance (typically, about 10%) on modern CPUs. The most important factors affecting the SpMV performance are the memory bandwidth limitation and the high cache-miss rate caused by the irregular and indirect memory access patterns.

On the other hand, since the arrival of the general-purpose graphics processing units (GPUs) to the HPC world many researchers are paying attention to these throughput-oriented manycore processors. GPUs are well-known because of their peak performance and high memory bandwidth. These facts make these systems good candidates for executing the SpMV efficiently. However, the irregular access patterns performed by SpMV are still a challenge for optimization.

Many strategies have been proposed to deal with the SpMV optimization. One of the most successful solutions are the reordering techniques. Note that in some cases the main goal of these

*Corresponding author

Email addresses: juancarlos.pichel@usc.es (Juan C. Pichel), ff.rivera@usc.es (Francisco F. Rivera), mfernandez@cesga.es (Marcos Fernández), aurelio@cesga.es (Aurelio Rodríguez)

techniques was different than the optimization of the sparse algebra operation. Reordering techniques evaluate the sparsity pattern of the matrix to find an appropriate permutation of rows and columns that improves the SpMV performance. To the best of our knowledge, these techniques have not been tested and discussed on GPUs.

In this paper, we have evaluated on two different GPUs (Tesla C1060 and M2050) some of the most successful reordering techniques. Four different sparse matrix storage formats (CSR, ELL, HYB and BELLPACK) have been considered using both single and double precision floating-point arithmetic. We have found that SpMV is very sensitive to the application of reordering techniques on GPUs. In particular, several characteristics of the reordered matrices that have a big impact on the SpMV performance have been detected. In most of the cases, reordered matrices outperform the original ones, showing speedups up to 2.6×. Moreover, we have detected that there is no one storage format prevailing over the others, which does not agree with the observations by Bell and Garland [5]. They found that HYB format is generally the fastest for a broad set of unstructured matrices, and from our experiments we can not state that.

The paper is structured as follows: Section 2 discusses previous work on SpMV optimization. Section 3 presents an overview of the reordering techniques and storage formats considered in this work. An explanation of the main characteristics of the SpMV kernels for GPUs is also provided. This section ends with a description of the hardware platform and matrix testbed used in the tests. Section 4 shows the performance evaluation results on both GPUs. The paper finishes with the main conclusions extracted from the work.

2. Related Work

Many works dealing with the optimization of the sparse matrix-vector product can be found in the literature. Most of these previous works try to optimize the performance of this computational kernel when executing on CPUs, and even some of them study the performance of SpMV when applying reordering techniques like those we are dealing with here. There also exist some works that try to optimize SpMV computations on GPUs, but it has not come to our notice any study about the benefits of applying these reordering techniques when executing SpMV on GPUs.

Techniques for increasing the SpMV performance can be mainly divided into two groups: data reordering and code restructuring techniques. Standard reordering techniques are considered classical methods for the SpMV optimization. The most used techniques are the bandwidth reduction algorithms, which derive from the Cuthill-McKee algorithm [10]. It has also been demonstrated the benefits of using minimum degree-based heuristics (such as the approximate minimum degree algorithm [1]) on multicore processors [19]. Oliker et al. [17] show the benefits that are offered by the application of some of these reordering algorithms to sparse codes when executed on different multiprocessor architectures. Coutinho et al. [9] perform a comparison of different data reordering algorithms for the SpMV in edge-based unstructured grid computations. However, they only focus on serial executions.

Techniques based on restructuring the code, like blocking or tiling, have been successfully applied to different irregular codes such as the product of a sparse matrix by a dense matrix [12, 16] and stationary iterative methods [22]. Im *et al.* [13] propose register and cache blocking as optimization techniques for the SpMV. In [7], a performance model for the blocked SpMV is presented, which allows to pick in nearly all cases the actual optimal blocksize. Vuduc *et al.* [24] extend the notion of blocking in order to exploit variable block shapes by decomposing the original matrix to a proper sum of submatrices storing each submatrix in a variation of the blocked CSR format. In a recent work [14], a comparative study of different blocking storage

techniques for sparse matrices on several multicore platforms is performed. Finally, Belgin *et al.* [3] introduce a representation for sparse matrices based on the observation that many matrices can be divided into blocks that share a small number of different patterns. The goal is to reduce the SpMV memory bandwidth requirements by reducing the index overhead.

Though in this work we have not considered code restructuring techniques, some authors have demonstrated that both groups of techniques are complementary. In particular, Toledo [23] evaluates different standard reordering techniques and combines them with blocking, showing that SpMV performance increases significantly depending on the size and sparseness of the considered matrix. Pinar and Heath [20] introduce a reordering technique that favors the creation of dense blocks on the pattern of the sparse matrix, and in this way the efficiency of the blocking technique proposed by Toledo is increased. Moreover, a comparison between their reordering technique and some standard reordering techniques is carried out. In another work [18] a reordering of the sparse matrix in combination with blocking techniques was successfully applied to the SpMV. This technique was evaluated on different uniprocessors and on distributed memory multiprocessors.

All of this previous works are related to the execution of SpMV on CPUs, but there are also some works focused on optimizing the performance on GPUs. One of the first papers about this issue was published by Bolz *et al.* obtaining promising results [6]. Sengupta *et al.* [21] developed more generic approaches using parallel scan primitives but that implementation was not as efficient as CPU codes at that time.

More recently, researchers have demonstrated that GPUs can execute this operation more efficiently than CPUs. In contrast with dense matrix operations (often limited by floating point throughput), sparse matrix operations typically have a much less regular memory access pattern and consequently are generally bandwidth limited. In [4], Bell and Garland demonstrate that despite the irregularity of the SpMV computation, it can be mapped successfully onto the fine-grained parallel architecture employed by the GPU. In that work they are able to harness a large fraction of the available memory bandwidth. They compare GPU SpMV results with the performance on a variety of multicore architectures obtained by Williams et al. [25], showing that GPUs offer better performance than multicores. Baskaran and Bordawekar [2] presented the key architectural optimizations that have to be addressed in GPUs for efficient executions: exploiting synchronization-free parallelism, optimizing thread mapping, aligning global memory accesses and exploiting data reuse. Evaluation of their optimizations shows that they are in par with NVIDIA's SpMV library.

As we stated above, reordering techniques could also be combined with blocking or tiling techniques to obtain better performance. This approach was used by Choi *et al.* [8] on GPUs. Authors propose a blocked ELLPACK (BELLPACK) implementation which achieves results among the best published thus far. However, a meticulous choice of data structure tuning parameters is required to make blocking techniques practical. In the same paper, authors present a performance autotuning model that tries to simplify this task.

3. Experimental Conditions

In this section the experimental conditions to evaluate the benefits of applying the reordering techniques for optimizing the SpMV are established.

3.1. Reordering Techniques

Reordering techniques have been a successful approach to improve the performance of the sparse matrix-vector multiplication (see Section 2). These techniques evaluate the sparsity pattern of the matrix to find an appropriate permutation of rows and columns of the original matrix. In some cases the main goal of these techniques was different than the optimization of the sparse algebra operation. For example, the main goal of the Cuthill-McKee ordering is to modify the sparsity pattern of the considered matrix with the aim of reducing its bandwidth [10]. Later works demonstrate that applying this reordering technique also increases the performance of the sparse matrix-vector product [20, 23]. This kind of techniques has obtained good results on different architectures, from monoprocessors to the newest multicore systems. However, until now, reordering techniques have not been evaluated on GPUs.

In this work some of the most successful reordering techniques for improving the performance of the SpMV have been considered. A brief description of them is provided next:

- Approximate Minimum Degree (AMD) [1]: The objective of this algorithm is to find a permutation of the original matrix that reduces the fill-in when a Cholesky factorization is performed.

- Distance function [19]: The goal of this technique is to increase the grouping of nonzero elements in the sparse matrix pattern that characterizes the irregular accesses and, as a consequence, increasing the locality in the execution of the SpMV code. This algorithm allows to permute individual rows/columns of the original matrix or sets of consecutive rows/columns. Depending on the choice we denote the technique as D or D_{set} respectively.

- Reverse Cuthill-McKee (RCM) [10]: As we have indicated previously, the goal of this technique is to reduce the bandwidth of the original matrix. Reverse Cuthill-McKee algorithm is the same algorithm as the original one but with the resulting index numbers reversed.

- METIS library [15]: It is considered a standard in terms of graph partitioning and reordering. The reordering technique included in the library computes fill-reducing orderings using a particular implementation of the nested dissection algorithm. In particular, we have used the `onmetis` program. This technique can only be applied to matrices with symmetric pattern.

Figure 1 shows, just for illustrative purposes, the appearance of one matrix from our testbed after applying the reordering techniques enumerated above.

3.2. Sparse Matrix Formats

For a sparse matrix, substantial memory requirement reductions can be obtained by storing only the nonzero entries. There exist many different storage formats, being ones more appropriate than others for a particular sparse matrix depending on the number and distribution of its nonzeros. These formats differ in terms of the amount of storage required, the accessing methods, and their adaptability to different applications or parallel architectures such as GPUs. Some of these formats are only well suited for matrices with a particular sparsity pattern like the diagonal format (DIA), other ones support efficient modification but not efficient matrix operations like for example the coordinate format (COO), and so on. In this work, we will mainly focus on those formats that are suitable for matrices with arbitrary structure and, at the same time, efficient for matrix operations. More precisely, we have considered the compressed row storage (CSR), ELLPACK (ELL), hybrid (HYB) and BELLPACK formats [4, 8]:

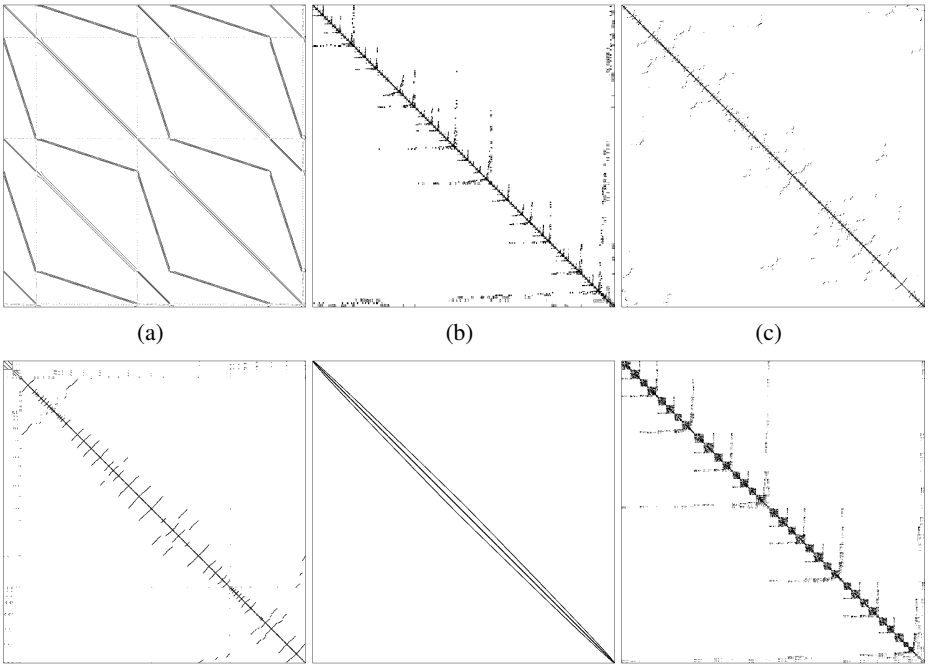

Figure 1: Example of reordered matrices using different techniques: (a) original matrix (`garon2`), and matrices after applying (b) AMD, (c) D, (d) D_{set}, (e) RCM and (f) METIS.

- Compressed Sparse Row (CSR): It is a general-purpose sparse matrix format. No assumptions are needed about the sparsity structure of the matrix. CSR allocates subsequent nonzeros in each row in contiguous memory positions and stores column indices and nonzero entries in two arrays, indices and values respectively. Besides, it needs another array of pointers that indicates the offset for each row. This format is efficient for arithmetic operations, row slicing, and matrix-vector products. Figure 2 illustrates an example of the CSR representation.

- ELLPACK (ELL): This storage scheme compresses the original sparse $n \times m$ matrix in a dense $n \times k$ matrix, where k is the maximum number of nonzeros per row of the original matrix. It also needs another $n \times k$ array of indices which stores the position (column) of each nonzero in the original matrix. This format can not be considered a general-purpose matrix format because it needs that the number of nonzeros in each row do not vary greatly through all the rows. In other case, a lot of storage space will be wasted and also the computational efficiency will decrease. However, it is suitable for a reasonable variety of matrices and the performance results it produces are generally good, so we decided to include it here. ELL representation of an example matrix is shown in Figure 2.

- Hybrid (HYB): This is a combination of two storage formats: COO and ELL. It tries to combine the computation efficiency of ELL with the simplicity and generality of COO (that stores row and column indices explicitly). The majority of the matrix entries are stored in ELL format, and those rows with a substantially different number of nonzeros

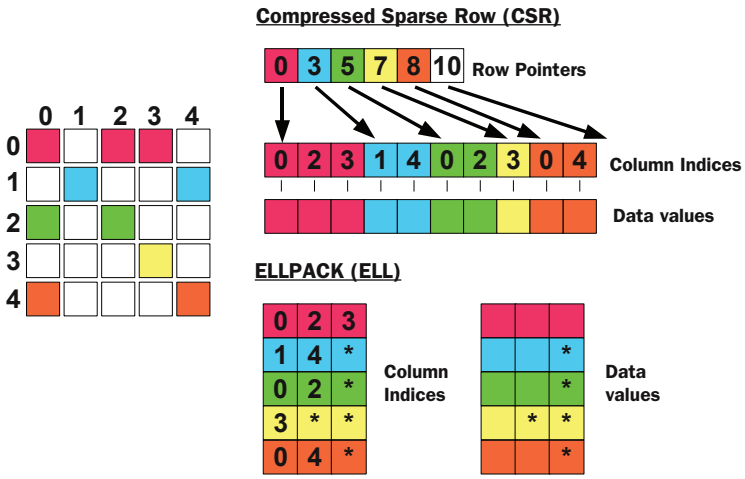

Figure 2: CSR and ELL sparse matrix storage formats.

are stored in COO format. As we will see, this scheme yields good performance results for the majority of sparse matrices we have tested.

- Blocked ELLPACK (BELLPACK): This format tries to adapt classical ELL to store $r \times c$ dense subblocks in order to save storage and flops, wasted in ELL because of the excessive zero-padding. A BELLPACK matrix is constructed from a $n \times m$ input matrix A. This matrix is firstly reorganized into a new matrix, A', stored using $r \times c$ dense subblocks. Then the block-rows are sorted in descending order of number of blocks per row, resulting in another matrix A''. Finally, the rows of A'' are partitioned into n/R non-overlapping submatrices, each of size $R \times m/c$. Each submatrix is then stored in ELL or BELLPACK format. Complete details about BELLPACK implementation can be found in [8].

3.3. Sparse Matrix-Vector Multiplication (SpMV)

Sparse matrix-vector multiplication (SpMV) is one the most important operations in scientific computing, mainly because iterative procedures for solving large linear systems (`y=A×x`) usually requires hundreds of iterations involving matrix-vector products to reach convergence. Due to this fact, there has been quite a lot of works trying to optimize this operation both in the realm of CPUs and also in the parallel world of GPUs. One of the latest and more productive efforts in this direction has been a library for SpMV operation that NVIDIA released recently and it is described in [4]. In that paper the authors comment the implementation details of the kernels used for different storage formats, and they also do the corresponding performance tests. This library supports DIA, ELL, CSR, COO, HYB and packet formats. We have used this library in order to evaluate the reordering techniques. Note that not all the supported kernels were considered in the results section. We have focused on those having a good compromise between generality and performance (that is, ELL, CSR and HYB formats). Moreover, we mentioned in the previous section another storage format, BELLPACK. This format is not included in the

GPU	NVIDIA Tesla C1060	NVIDIA Tesla M2050
CPU	2 × Intel Xeon Quad-core E5520	
Memory	6 × 2 GB DDR3-1333	
OS	Ubuntu 9.10	Ubuntu 10.04 LTS
CUDA version	3.0	3.2
Host Compiler	`gcc` 4.3.4	`gcc` 4.4.3

Table 1: Test platforms specifications.

NVIDIA library. It was developed by Choi et al. [8], and we have used their implementation of the SpMV in our tests.

The ELL kernel uses one thread per matrix row to parallelize the computation and benefits from full coalescing when accessing the sparse matrix. It is usually a good performer when the matrix fulfills the requirements commented in the previous section. CSR kernel is implemented in two different ways, one called CSR_scalar and other named CSR_vector. As the ELL kernel, CSR_scalar uses one thread per row but its memory access pattern prevents it from taking advantage of the GPU coalescing feature, which usually leads to a decrease in performance. CSR_vector kernel differs from the other two kernels, and assigns one warp per matrix row. This permits the kernel to access the CSR structure contiguously but usually not in an aligned way, which implies partial coalescing. Even so, performance of the CSR_vector kernel in our tests is always superior to that of the CSR_scalar one, so from now on when we mention the CSR kernel we will refer to the CSR_vector kernel. The HYB kernel is a combination of ELL and COO kernels (one thread per row and full coalescing in both cases). Because most nonzeros usually belong to the ELL portion, performance of the HYB kernel will often be similar to that of the ELL kernel. Further details about the implementation of each kernel can be found in [4]. Finally, BELLPACK, as the ELL kernel, benefits from full coalescing, and though its improvements over ELL apply only to matrices that have small dense block sub-structures, it has proved to be a good performer in most of the cases, and even the best for some of them. However, it is necessary to say that the optimal parameters for the blocking process (r, c) and the optimal number of threads per block had to be found by exhaustive search. Further details about the implementation of this kernel can be found in [8].

Besides, as Bell and Garland state in their work, most of the SpMV kernels can benefit from using the texture cache present on all CUDA-capable devices. In this way, using this cache to access the `x` vector often improves performance considerably. In our tests, using the texture cache always led to a better result than the correspondent non-cache kernel, so the results shown in this paper are always related to the utilization of this texture cache.

3.4. Hardware Platform and Matrix Testbed

Experiments were performed on two similar platforms, but with different GPUs. Both platforms were equipped with 12GB RAM and two Intel Xeon E5520 (Nehalem). Besides, one of the platforms was equipped with a NVIDIA Tesla C1060 and the other one with a NVIDIA Tesla M2050 (Fermi architecture). Table 1 summarizes the main characteristics of the test platforms. SpMV codes were compiled with NVIDIA CUDA compiler (`nvcc`) and the options indicated in the library documentation: `-arch=sm_13` (for the GT200 architecture) and `-arch=sm_20` (for the Fermi architecture), with the optimization flag `-O3`.

	Matrix	rows (n)	nnz		Matrix	rows (n)	nnz
1	av41092*	41092	1683902	16	nd3k	9000	3279690
2	bcsstm36	23052	320606	17	net25	9520	401200
3	crystk03	24696	1751178	18	nmos3	18588	386594
4	e40r0100*	17281	553562	19	pct20stif	52329	2698463
5	F2	71505	5294285	20	psmigr_1*	3140	543162
6	fp*	7548	848553	21	rajat15	37261	443573
7	garon2	13535	390607	22	ship_001	34920	4644230
8	gupta2	62064	4248286	23	sme3Da*	12504	874887
9	gyro_k	17361	1021159	24	sme3Dc*	42930	3148656
10	lhr10*	10672	232633	25	sparsine	50000	1548988
11	li	22695	1350309	26	syn12000a*	12000	1436806
12	msc10848	10848	1229778	27	tandem_vtx	18454	253350
13	Na5	5832	305630	28	thread	29736	4470048
14	nc5	19652	1499816	29	TSOPF_FS_b300	56814	8767466
15	ncvxbqp1	50000	349968	30	tsyl201	20685	2454957

Table 2: Matrix benchmark suite. Matrices with an asterisk have a non-symmetric sparsity pattern.

As matrix test set we have selected thirty square sparse matrices from different real applications (structural problems, circuit and n-body simulations, linear programming, ...) that represent a variety of nonzero patterns. For example, there are banded matrices, non-banded matrices with regular structure, non-symmetric matrices, etc. All these matrices are from the University of Florida Sparse Matrix Collection (UFL) [11]. Table 2 summarizes the features of the matrices. nnz is the number of nonzeros.

4. Experimental Evaluation

Performance results of the SpMV are shown and discussed next. We have compared the performance obtained by the original matrices (without reordering) with respect to the reordered ones using the four storage formats considered (CSR, ELL, HYB and BELLPACK). Note that METIS reordering technique can only be applied to matrices with a symmetric pattern (see Table 2).

Results for a particular matrix are displayed in the figures only if the original performance differs from the obtained by any of its reorderings in more than 1%. We report performance in terms of GFLOPS. Note that the cost of transferring the matrix between the host memory and the device memory is not included. Tests using single (32 bits) and double precision (64 bits) floating-point arithmetic have been carried out.

Performance results obtained on the Tesla C1060 are discussed in Section 4.1, while the ones obtained with the Fermi GPU (Tesla M2050) are shown in Section 4.2. It is worth to mention that the results with Tesla M2050 were obtained without ECC and with a 48KB L1 cache configuration. Finally, a comparison between both GPUs is shown in Section 4.3.

4.1. Tesla C1060

4.1.1. Single Precision Case

Figure 3 reports the SpMV performance results considering CSR storage format. The fact that results for 20 matrices (66% of the total) are displayed in the figure points out that performance using this format is very sensitive to the application of reordering techniques. In most

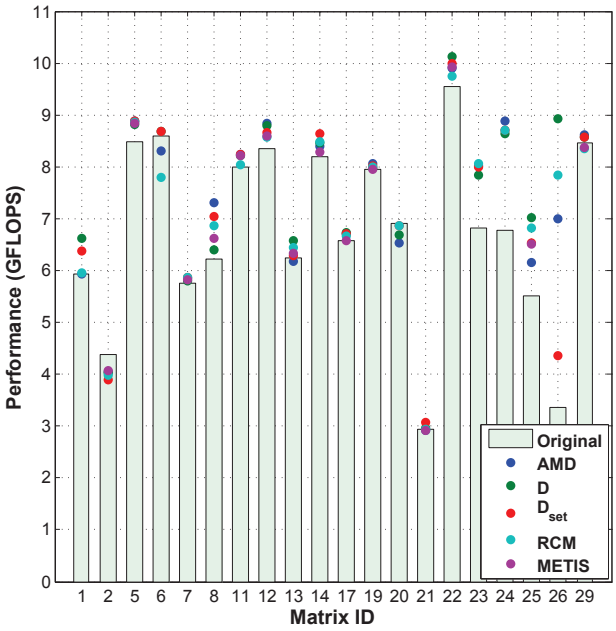

Figure 3: Performance of the reordered matrices using CSR format (single precision) on the Tesla C1060.

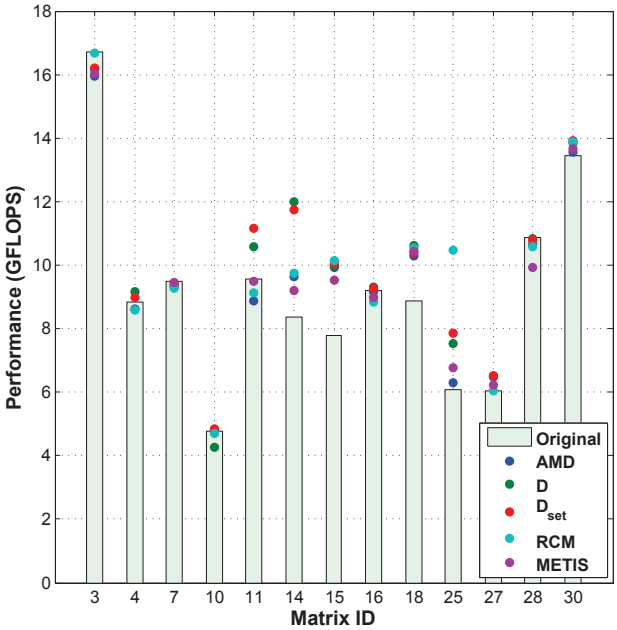

Figure 4: Performance of the reordered matrices using ELL format (single precision) on the Tesla C1060.

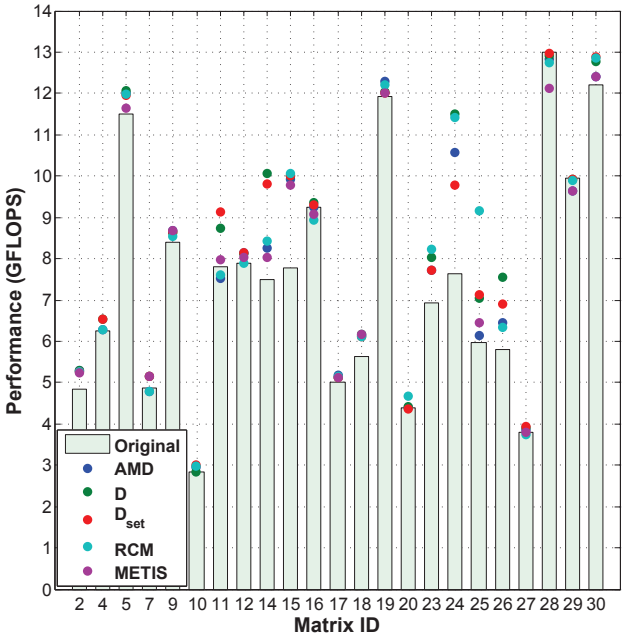

Figure 5: Performance of the reordered matrices using HYB format (single precision) on the Tesla C1060.

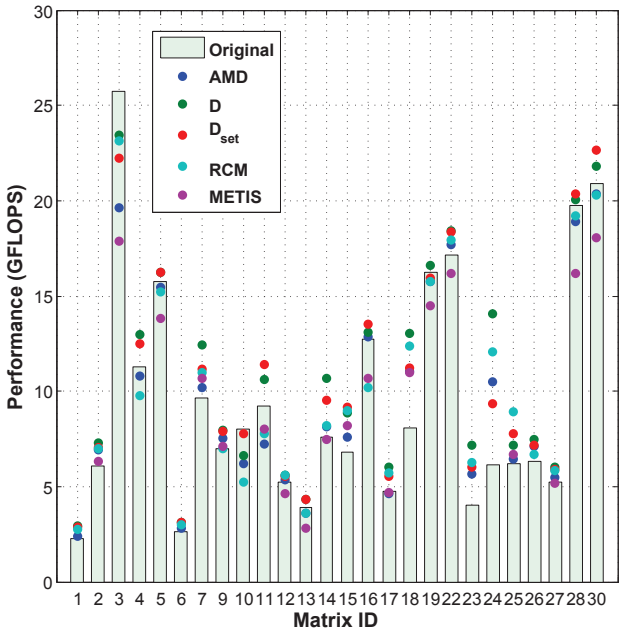

Figure 6: Performance of the reordered matrices using BELLPACK format (single precision) on the Tesla C1060.

Matrix ID	Original	AMD	D	D_{set}	RCM	METIS
1	6.40 $_H$	6.37 $_H$	**6.62** $_C$	6.37 $_C$	6.34 $_H$	-
2	6.10 $_H$	**6.95** $_B$	**7.28** $_B$	**7.04** $_B$	**6.97** $_B$	**6.31** $_B$
3	**25.72** $_B$	19.61 $_B$	23.43 $_B$	22.22 $_B$	23.14 $_B$	17.87 $_B$
4	11.31 $_B$	10.78 $_B$	**12.96** $_B$	**12.47** $_B$	9.77 $_B$	-
5	15.76 $_B$	15.43 $_B$	**16.26** $_B$	**16.26** $_B$	15.21 $_B$	13.81 $_B$
6	8.60 $_C$	8.31 $_C$	**8.69** $_C$	**8.69** $_C$	7.80 $_C$	-
7	9.62 $_B$	**10.19** $_B$	**12.45** $_B$	**11.17** $_B$	**11.00** $_B$	**10.67** $_B$
8	8.50 $_H$	**8.61** $_H$	8.46 $_H$	8.36 $_H$	**8.61** $_H$	8.50 $_H$
9	8.41 $_H$	**8.68** $_H$	**8.61** $_H$	**8.65** $_H$	**8.55** $_H$	**8.69** $_H$
10	**7.99** $_B$	6.22 $_B$	6.60 $_B$	7.79 $_B$	5.25 $_B$	-
11	9.55 $_E$	8.88 $_E$	**10.63** $_B$	**11.39** $_B$	9.13 $_E$	9.48 $_E$
12	8.34 $_C$	**8.83** $_C$	**8.80** $_C$	**8.66** $_C$	**8.58** $_C$	**8.59** $_C$
13	6.25 $_C$	6.17 $_C$	**6.57** $_C$	**6.29** $_C$	**6.45** $_C$	**6.34** $_C$
14	8.37 $_E$	**9.62** $_E$	**11.99** $_E$	**11.73** $_E$	**9.75** $_E$	**9.21** $_E$
15	7.79 $_E$	**9.93** $_E$	**10.02** $_H$	**10.03** $_E$	**10.14** $_E$	**9.78** $_H$
16	12.74 $_B$	**12.84** $_B$	**13.07** $_B$	**13.49** $_B$	10.34 $_B$	10.68 $_B$
17	6.57 $_C$	**6.71** $_C$	**6.73** $_C$	**6.71** $_C$	**6.66** $_C$	**6.58** $_C$
18	8.88 $_E$	**11.05** $_B$	**13.06** $_B$	**11.21** $_B$	**12.37** $_B$	**10.99** $_B$
19	16.23 $_B$	15.73 $_B$	**16.61** $_B$	15.95 $_B$	15.78 $_B$	14.49 $_B$
20	**6.91** $_C$	6.53 $_C$	6.69 $_C$	6.87 $_C$	6.87 $_C$	-
21	4.67 $_H$	**4.74** $_H$	**4.76** $_H$	**4.69** $_H$	4.61 $_H$	**4.78** $_H$
22	17.17 $_B$	**17.71** $_B$	**18.39** $_B$	**18.33** $_B$	**17.91** $_B$	16.17 $_B$
23	6.94 $_H$	**8.02** $_C$	**8.02** $_H$	**8.00** $_C$	**8.23** $_H$	-
24	7.63 $_H$	**10.58** $_H$	**14.05** $_B$	**9.79** $_H$	**12.06** $_B$	-
25	6.21 $_B$	**6.45** $_B$	**7.53** $_E$	**7.85** $_E$	**10.48** $_E$	**6.77** $_E$
26	6.32 $_B$	**7.08** $_B$	**8.93** $_C$	**7.18** $_B$	**7.84** $_C$	-
27	6.03 $_E$	**6.49** $_E$	**6.52** $_E$	**6.51** $_E$	**6.04** $_E$	**6.23** $_E$
28	19.73 $_B$	18.88 $_B$	**20.02** $_B$	**20.33** $_B$	19.21 $_B$	16.16 $_B$
29	**9.95** $_H$	9.64 $_H$	9.93 $_H$	9.93 $_H$	9.90 $_H$	9.64 $_H$
30	20.92 $_B$	20.36 $_B$	**21.81** $_B$	**22.68** $_B$	20.28 $_B$	18.07 $_B$

Table 3: Summary of the best single precision performance results (in GFLOPS) on the Tesla C1060. Subscripts show the corresponding storage format: C (CSR), E (ELL), H (HYB) and B (BELLPACK).

of the cases, reordered matrices outperform the original ones. Worst behavior is observed for AMD reorderings, which even then increase the performance in 60% of the cases (18 matrices). Best results are observed for the reorderings of matrices 23, 24, 25 and 26. This is caused by an increase in the clustering of the nonzeros within each row of the reordered matrices with respect to the original ones, which will lead to access to closer elements of the vector when the SpMV operation is performed, improving the spatial locality. Note that speedups up to 2.6× (matrix 26 and D reordering) are reached. The worst case corresponds to matrix 2, where some reordering degrades the original matrix performance about 9%. Average performance improvement ranges from 1.5% using METIS to 6.4% when D is considered.

Results considering ELL format are displayed in Figure 4. This format is only efficient when the maximum number of nonzeros per row does not substantially differ from the average (see Section 3.2). This is the case of the thirteen matrices in the figure. As when using CSR format, reordered techniques have an impact on the SpMV performance. We have observed for this format that a bandwidth reduction in the matrices has a big influence on the performance, which is the case of reorderings of matrices 14, 15, 18 and 25. Noticeable speedups are observed such as 1.7× for matrix 25 and RCM reordering. Best overall behavior is obtained by D_{set} technique,

improving on average the performance above 9%. Degradations are always lower than 4%.

Figure 5 shows the results obtained using the HYB kernel. Behavior is similar to that found for CSR and ELL formats in the sense that reorderings improve the performance of the original matrices in most of the cases. We must emphasize that RCM, which corresponds to the worst case, increases the performance of 19 matrices, that is, 63% of the testbed. On the other hand, HYB format is a combination of ELL and COO formats. Assuming that most nonzeros belong to the ELL portion [3], this is the reason why a bandwidth reduction has also a big impact on the performance with the HYB format (see matrices 14, 15, 18, 23, 24 and 25). Moreover, speedups higher than 1.5× were reached, which is the case of matrices 24 and 25. It is worth to mention that average improvements vary from 1.4% (METIS) to 6.8% (D).

Some observations were made when comparing the best performance obtained by each matrix and reordering considering only the formats considered by Bell and Garland [5] (CSR, ELL and HYB). There is no one format prevailing over the others. For example, considering the original matrices, HYB is the best in 12 cases (40% of the total), while ELL and CSR dominate on 9 matrices each one (30% of the total). This behavior is also found for the reorderings. It does not agree with the observations by Bell and Garland [5]. They found that HYB format is generally the fastest for a broad set of unstructured matrices, and from our experiments we can not state that. This observation was also found with double precision arithmetic, and in the tests performed on the Tesla M2050 (see Section 4.2).

Figure 6 displays the performance observed when considering the BELLPACK format. This is the storage format for which reordering techniques have the most influence. This is caused by a limitation in the BELLPACK format: it can only be successfully applied to matrices that have small dense block sub-structures [8]. Therefore, modifying the sparsity pattern of the original matrices will influence very much the efficiency of this approach. According to the results, several conclusions can be made. First, the higher performance is always obtain by FEM matrices with dense block sub-structures. For example, original matrices 3, 28 and 30 achieve 25.7, 19.7 and 20.9 GFLOPS respectively. This observation agrees with the results in [8]. Secondly, D and D_{set} obtain the best results overall. This behavior was expected because the goal of this reordering technique is to increase the grouping of nonzero elements in the sparse matrix pattern, which will favor the creation of small dense sub-blocks. In a previous work [18] the authors demonstrate this fact by means of the combination of reordering and register blocking techniques on different multiprocessors. In particular, the average improvement using D and D_{set} reorderings is 14% and 10.2% respectively, reaching speedups up to 2.2× (matrix 24). And finally, we have observed that METIS is specially inefficient with this format as it is shown by the average degradation of 8% in the performance. Results point out that the typical arrow-shaped matrices generated by METIS (see Figure 1(f)) does not contain small dense sub-blocks.

In order to summarize the observations, Table 3 contains the best performance obtained for each matrix and reordering technique considering all the storage formats. Subscripts show which storage format performs better. According to these results several conclusions can be made. First, BELLPACK is the best storage format in about 40% of the matrices. Note that we have only considered the best block size in order to apply this format.

Secondly, we have observed that for some matrices (1, 2, 11, 15, 18, 23, 24 and 26), a different storage format is preferred by some of the reorderings with respect to the original matrix. For example, AMD and D_{set} reorderings of matrix 23 obtain their best performance with CSR, while the original matrix uses the HYB format. It illustrates the difficulty to choice some storage format as the best solution.

And finally, it is noticeable that only for 4 matrices no optimizations were achieved. Note

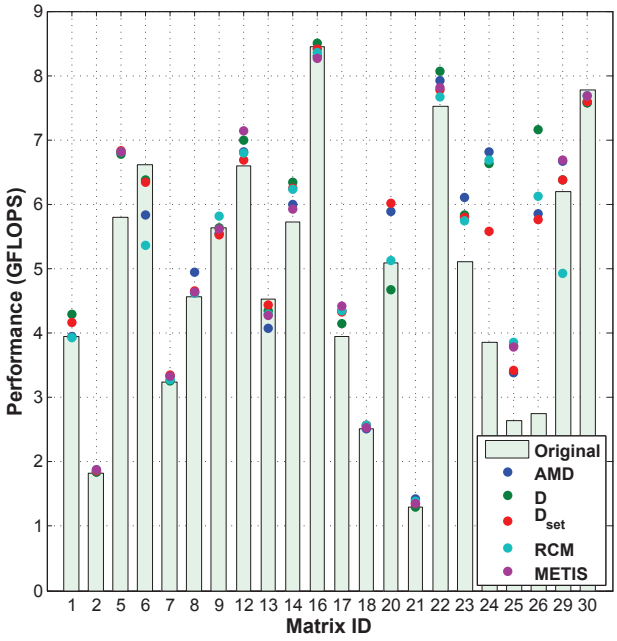

Figure 7: Performance of the reordered matrices using CSR format (double precision) on the Tesla C1060.

that, for example, D reorderings outperform 25 original matrices. This demonstrates that reordering techniques have a big impact on the SpMV performance on GPUs. Speedups reach values up to 1.8× (matrix 24) or 1.7× (matrix 25). In particular, the overall average improvement is 1%, 9.8%, 6.9%, 3.3% and -5.6% for AMD, D, D_{set}, RCM and METIS respectively. Note that METIS degrades the SpMV performance with respect to original matrices. This is caused by the bad behavior when considering the BELLPACK format.

4.1.2. Double Precision Case

Next we present the results obtained when considering double precision floating-point arithmetic. Figure 7 displays the performance achieved using the CSR format. Note that performance with double precision is always lower than the obtained with single precision arithmetic. For example, considering original matrices, average performance using single precision is 10.2 GFLOPS, while with double is 6.7 GFLOPS. This behavior was observed for all the matrices and storage formats.

In this case, as it was expected, reordering techniques have also a big influence on the SpMV performance. Note that even in the worst case, which corresponds to D, reorderings beat 19 original matrices (63% of the testbed). Best results are obtained by reorderings (matrices 24, 25 and 26) that increase the clustering of the nonzeros within each row of the matrix, improving the spatial locality in the accesses to the vector when the SpMV is performed. Speedups reach values up to 2.6× (matrix 26 and D). Average improvements range from 3.7% (METIS) to 8.2% (D), which are actually higher than those observed using single precision (see previous section).

Performance evaluation of the reordering techniques using ELL is shown in Figure 8. ELL storage format is only efficient for the thirteen matrices in the graph, in the same way as when

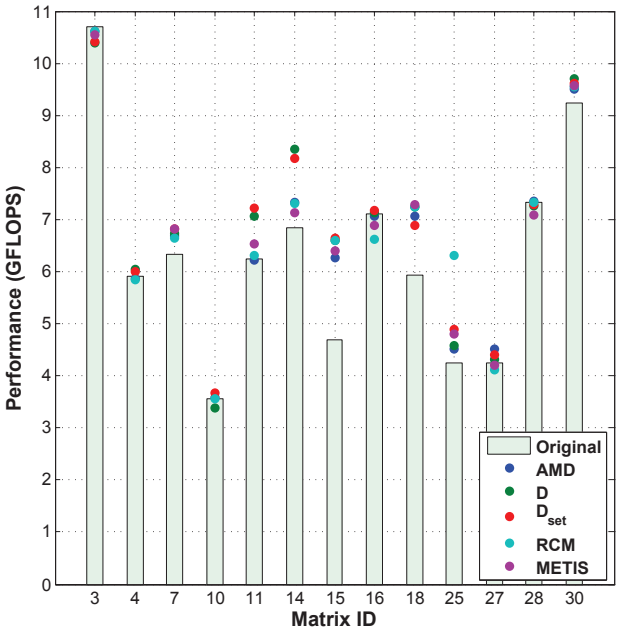

Figure 8: Performance of the reordered matrices using ELL format (double precision) on the Tesla C1060.

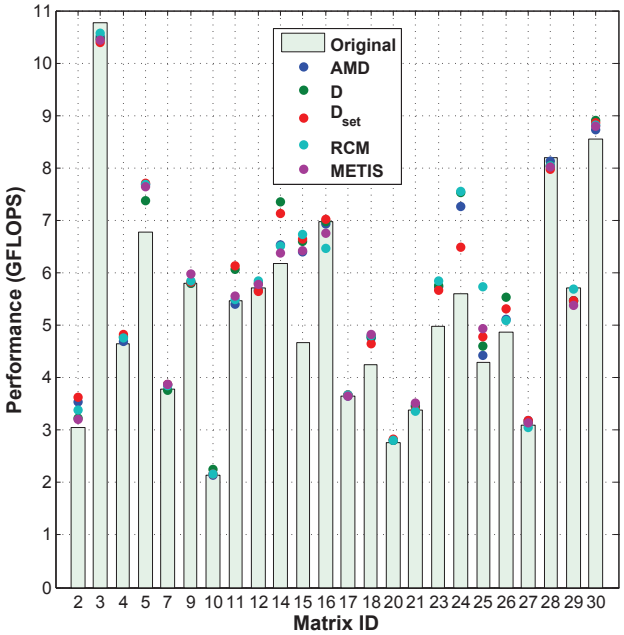

Figure 9: Performance of the reordered matrices using HYB format (double precision) on the Tesla C1060.

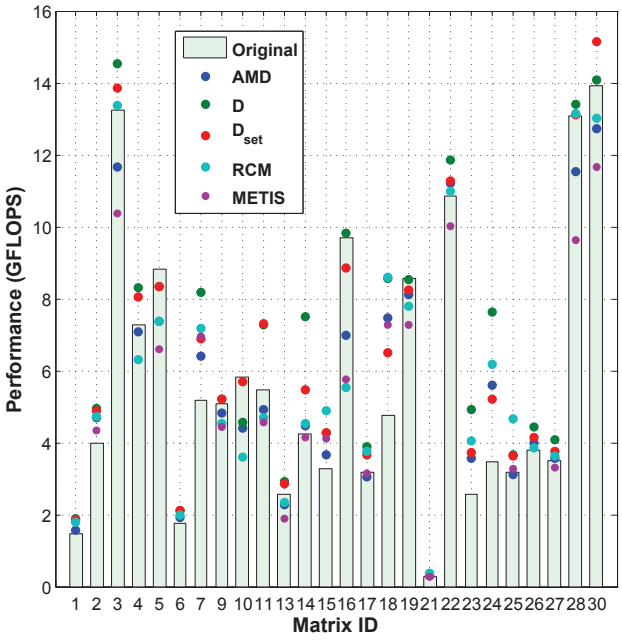

Figure 10: Performance of the reordered matrices using BELLPACK format (double precision) on the Tesla C1060.

using single precision arithmetic (Figure 4). We have also observed for double precision that a bandwidth reduction in the matrices is the most important factor that influences the performance. Noticeable improvements are observed with speedups exceeding 1.4× (matrices 15 and 25). Degradations are always lower than 4%. Average improvements vary from 5.1% (AMD) to 8.3% (D_{set}). It is noticeable that the average improvement of AMD reorderings increases from 3% in single precision to 7.4% with 64-bits arithmetic.

Figure 9 displays the results using the HYB format. In most of the cases, reordered matrices outperform the original ones. Note that the behavior of the reorderings with ELL and HYB formats (see Figure 8) is, as it was expected, very similar. This is the case, for example, of matrices 3, 16, 25 and 30. The best average improvement is obtained by D reordering technique (as in single precision), increasing performance about 6% with respect to the original matrices. Worst case corresponds to METIS, which average improvement reaches 3.1%. Some important speedups are achieved such as 1.4× with matrix 15.

The performance evaluation results when considering the BELLPACK format (Figure 10) confirm the observations obtained with single-precision arithmetic. First, the higher performance is always obtain by FEM matrices with dense block sub-structures. For example, original matrices 3, 28 and 30 achieve 13.2, 13.1 and 13.9 GFLOPS respectively. Secondly, D and D_{set} obtain the best results overall. In particular, the average improvement using D and D_{set} reorderings is 17% and 9.8% respectively, reaching speedups up to 2.2× (matrix 24). And finally, we have also observed that METIS is specially inefficient with this format as it is shown by the average degradation of about 11% in the performance.

As a summary we have included the best double precision performance results obtained for each matrix and reordering technique considering the four storage formats, CSR, ELL, HYB and

Matrix ID	Original	AMD	D	D_{set}	RCM	METIS
1	3.95 $_C$	3.95 $_C$	**4.30** $_C$	**4.17** $_C$	3.92 $_C$	-
2	3.99 $_B$	**4.68** $_B$	**4.94** $_B$	**4.88** $_B$	**4.72** $_B$	**4.33** $_B$
3	13.24 $_B$	11.66 $_B$	**14.54** $_B$	**13.86** $_B$	**13.37** $_B$	10.55 $_E$
4	7.26 $_B$	7.09 $_B$	**8.31** $_B$	**8.05** $_B$	6.30 $_B$	-
5	**8.81** $_B$	7.69 $_H$	8.33 $_B$	8.34 $_B$	7.69 $_H$	7.65 $_H$
6	**6.62** $_C$	5.84 $_C$	6.38 $_C$	6.34 $_C$	5.36 $_C$	-
7	6.33 $_E$	**6.68** $_E$	**8.18** $_B$	**6.88** $_B$	**7.19** $_B$	**6.95** $_B$
8	5.30 $_H$	**5.36** $_H$	5.26 $_H$	5.23 $_H$	**5.36** $_H$	5.26 $_H$
9	5.80 $_H$	5.79 $_H$	**5.81** $_H$	**5.83** $_H$	**5.84** $_H$	**5.97** $_H$
10	**5.82** $_B$	4.40 $_B$	4.56 $_B$	5.68 $_B$	3.59 $_B$	-
11	6.24 $_E$	6.23 $_E$	**7.28** $_B$	**7.30** $_B$	**6.30** $_E$	**6.54** $_E$
12	6.59 $_C$	**6.81** $_C$	**6.99** $_C$	**6.69** $_C$	**6.79** $_C$	**7.14** $_C$
13	**4.52** $_C$	4.07 $_C$	4.35 $_C$	4.44 $_C$	4.29 $_C$	4.27 $_C$
14	6.84 $_E$	**7.34** $_E$	**8.35** $_E$	**8.17** $_E$	**7.30** $_E$	**7.12** $_E$
15	4.69 $_E$	**6.40** $_H$	**6.60** $_E$	**6.65** $_E$	**6.73** $_H$	**6.41** $_H$
16	9.68 $_B$	8.31 $_C$	**9.82** $_B$	8.84 $_B$	8.36 $_C$	8.27 $_C$
17	3.95 $_C$	**4.35** $_C$	**4.15** $_C$	**4.32** $_C$	**4.35** $_C$	**4.41** $_C$
18	5.94 $_E$	**7.46** $_B$	**8.57** $_B$	**6.89** $_E$	**8.61** $_B$	**7.29** $_B$
19	**8.56** $_B$	8.11 $_B$	8.54 $_B$	8.23 $_B$	8.09 $_H$	7.96 $_H$
20	5.09 $_C$	**5.89** $_C$	4.67 $_C$	**6.02** $_C$	**5.12** $_C$	-
21	3.38 $_H$	**3.47** $_H$	**3.44** $_H$	**3.39** $_H$	3.37 $_H$	**3.51** $_H$
22	10.85 $_B$	**11.22** $_B$	**11.84** $_B$	**11.26** $_B$	**10.99** $_B$	10.02 $_B$
23	5.10 $_C$	**6.10** $_C$	**5.83** $_C$	**5.80** $_C$	**5.84** $_H$	-
24	5.60 $_H$	**7.26** $_H$	**7.53** $_H$	**6.48** $_H$	**7.56** $_H$	-
25	4.29 $_H$	**4.52** $_E$	**4.61** $_H$	**4.90** $_E$	**6.30** $_E$	**4.93** $_H$
26	4.86 $_H$	**5.85** $_C$	**7.15** $_C$	**5.76** $_C$	**6.13** $_C$	-
27	4.25 $_E$	**4.51** $_E$	**4.31** $_E$	**4.40** $_E$	4.12 $_E$	4.21 $_E$
28	13.07 $_B$	11.52 $_B$	**13.40** $_B$	**13.10** $_B$	**13.13** $_B$	9.62 $_B$
29	6.20 $_C$	**6.67** $_C$	**6.37** $_C$	**6.37** $_C$	5.92 $_H$	**6.69** $_C$
30	13.92 $_B$	12.71 $_B$	**14.09** $_B$	**15.15** $_B$	13.02 $_B$	11.66 $_B$

Table 4: Summary of the best double precision performance results (in GFLOPS) on the Tesla C1060. Subscripts show the corresponding storage format: C (CSR), E (ELL), H (HYB) and B (BELLPACK).

BELLPACK in Table 4.

It is worth to mention that there is not one storage format preferred over the others. However, CSR and BELLPACK formats are the best choices in about 30% of the cases respectively. Also note that different storage formats are preferred by some of the reorderings with respect to the original matrix. This is the case, for example, of matrices 5, 7, 15, 25, 26 and 29.

Moreover, we have detected some matrices for which their best performance is obtained with different storage formats in case of considering single or double precision arithmetic. For example, the original matrix 1 reaches its best performance using HYB and CSR for single and double case respectively.

On the other hand, there are only 5 matrices (16.7% of the testbed) for which no improvements were observed. Important speedups has been found. For example, speedups for matrices 15, 18, 25 and 26 exceed 1.4×. Overall average improvement is 0.8%, 8.9%, 6.3%, 2.5% and -3.6% for AMD, D, D_{set}, RCM and METIS respectively. These results show a slightly worse behavior with respect to the single precision tests. Note that METIS degrades the SpMV performance with respect to original matrices caused by the results when considering the BELLPACK format.

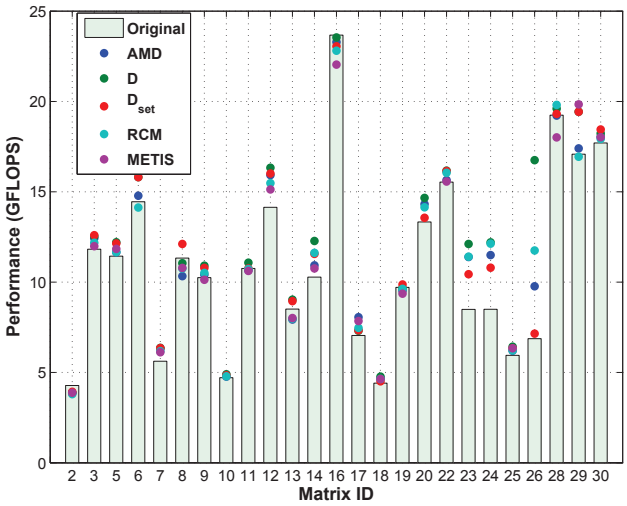

Figure 11: Performance of the reordered matrices using CSR format (single precision) on the Tesla M2050.

4.2. Tesla M2050 (Fermi architecture)

4.2.1. Single Precision Case

Figure 11 displays the SpMV performance results considering CSR storage format. In most of the cases, reordered matrices outperform the original ones. We must highlight that D and D_{set} reorderings increase the performance of 24 and 25 original matrices of the testbed. Best results are observed for those reorderings whose nonzeros are located in closer positions within each row. This grouping will lead to access to closer elements of the vector when the SpMV operation is performed, improving the spatial locality. This is the case of the reorderings of matrices 12, 23, 24 and 26. Note that speedups up to 2.4× (matrix 26 and D reordering) are reached. Average performance improvement ranges from 0.9% using METIS to 10.8% when D is considered.

Results considering ELL format are shown in Figure 12. Note that this format is only efficient when the maximum number of nonzeros per row does not substantially differ from the average (see Section 3.2). We have observed for this format that a bandwidth reduction in the matrices has a big impact on the performance. For example, reorderings of matrix 25 reach speedups up to 1.9×. Best overall behavior is obtained by D and RCM techniques, improving on average the performance about 11%.

Figure 13 shows the results obtained using the HYB kernel. Behavior is similar to that found for CSR and ELL formats in the sense that reorderings improve the performance of the original matrices in most of the cases. As we have noted previously, HYB format is a combination of ELL and COO formats. Assuming that most nonzeros belong to the ELL portion [3], this is the reason why a bandwidth reduction has also a big impact on the performance with the HYB format. Speedups higher than 1.4× were reached (matrix 24), with average improvements that vary from 1% (METIS) to 6.4% (D).

Finally, the results considering the BELLPACK format are displayed in Figure 14. These results agree with the observations obtained when using the Tesla C1060 as test platform. First, the higher performance is always obtain by FEM matrices with dense block sub-structures. This is the case of the original matrices 3, 28 and 30, whose SpMV performance is 40.1, 33.7 and 35.9

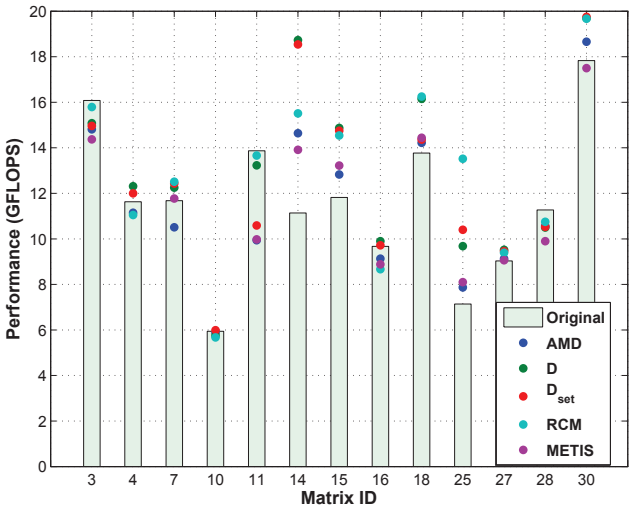

Figure 12: Performance of the reordered matrices using ELL format (single precision) on the Tesla M2050.

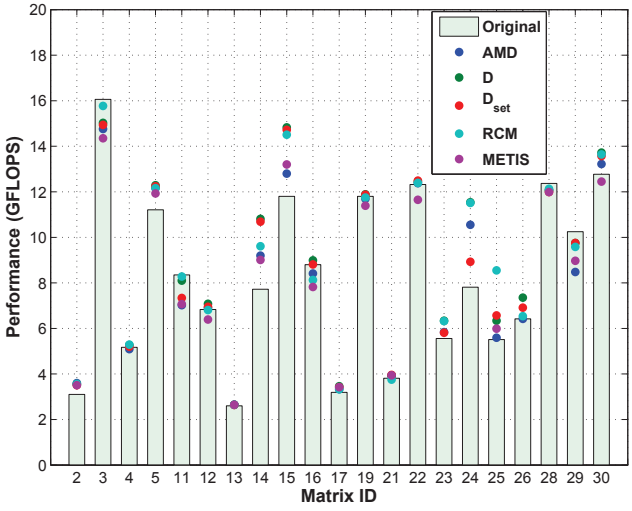

Figure 13: Performance of the reordered matrices using HYB format (single precision) on the Tesla M2050.

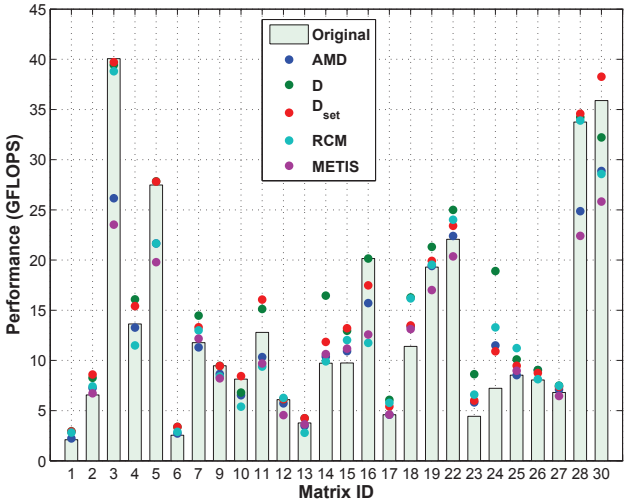

Figure 14: Performance of the reordered matrices using BELLPACK format (single precision) on the Tesla M2050.

GFLOPS respectively. Secondly, D and D_{set} obtain the best results overall. We must take into account that the goal of this reordering technique is to increase the grouping of nonzero elements in the sparse matrix pattern that characterizes the accesses of the SpMV operation [19]. In particular, the average improvement using D and D_{set} reorderings is 13.4% and 8.4% respectively, reaching speedups up to 2.6× (matrix 24). And finally, we have observed that AMD and METIS are specially inefficient with this format as it is shown by the average degradation of about 10% and 19% in the performance respectively. We have detected that AMD and METIS reorderings break the small dense sub-blocks of the original FEM matrices. This is the case of matrices 3, 5, 11, 28 and 30.

Table 5 contains the best performance obtained for each matrix and reordering technique considering all the storage formats. Subscripts show which storage format performs better. We must highlight that there are only 2 matrices for which reorderings do not improve their performance. It is specially relevant the results obtained by D and D_{set}, improving the performance of more than 80% of the testbed. In most of the cases the highest performance is achieved considering CSR or BELLPACK, which coincide with the best formats for D and D_{set} reorderings. Noticeable speedups have been observed. We must emphasize the cases of matrices 24 and 26, with speedups up to 2.2× and 2.1× respectively.

Overall average improvement is -5%, 12.3%, 8.7%, 2.7% and -12.5% for AMD, D, D_{set}, RCM and METIS respectively. Note that AMD and METIS reorderings reduce the average performance with respect to the original matrices. In both cases the problem was the performance using BELLPACK format. As we have commented above, we have detected that AMD and METIS reorderings break the small dense sub-blocks of the original FEM matrices. This is the case of matrices 3, 5, 11, 28 and 30 (see Figure 14 and Table 5).

4.2.2. Double Precision Case

Next the performance evalaution results when considering double precision floating-point arithmetic are introduced. We must highlight that the behavior observed for all the storage for-

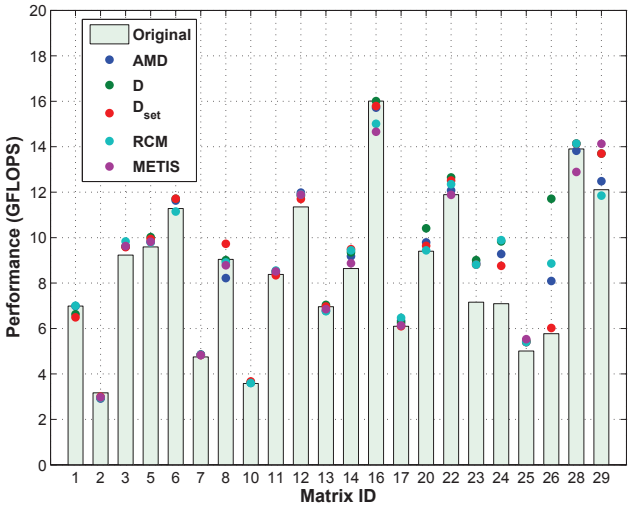

Figure 15: Performance of the reordered matrices using CSR format (double precision) on the Tesla M2050.

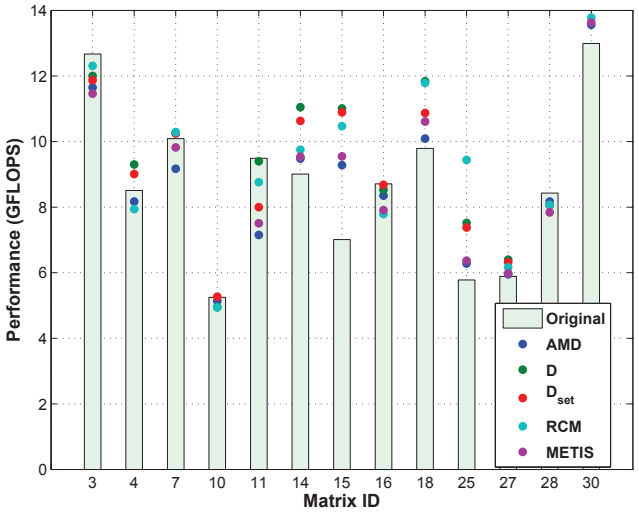

Figure 16: Performance of the reordered matrices using ELL format (double precision) on the Tesla M2050.

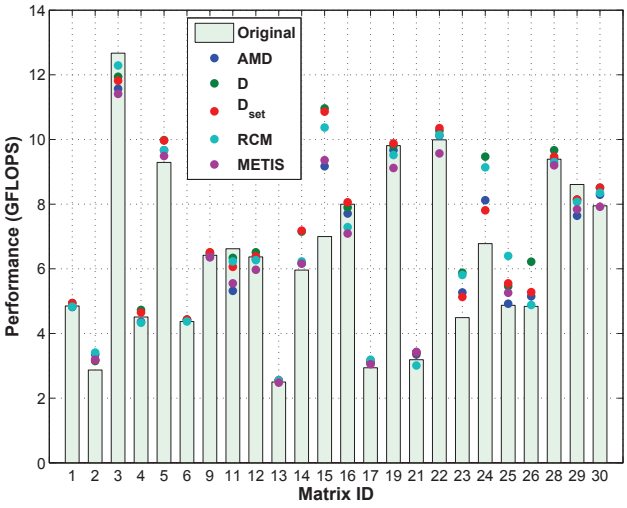

Figure 17: Performance of the reordered matrices using HYB format (double precision) on the Tesla M2050.

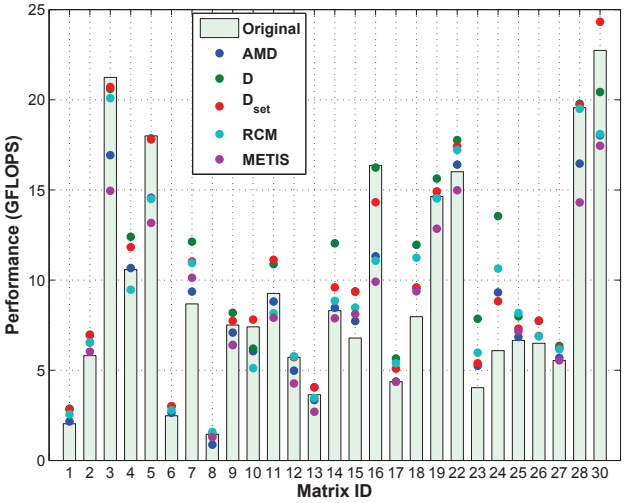

Figure 18: Performance of the reordered matrices using BELLPACK format (double precision) on the Tesla M2050.

Matrix ID	Original	AMD	D	D_{set}	RCM	METIS
1	8.17 $_C$	**8.20** $_C$	**8.18** $_C$	8.11 $_C$	**8.22** $_C$	-
2	6.56 $_B$	**7.25** $_B$	**8.27** $_B$	**8.60** $_B$	**7.42** $_B$	**6.72** $_B$
3	**40.06** $_B$	26.16 $_B$	39.44 $_B$	39.71 $_B$	38.81 $_B$	23.53 $_B$
4	13.63 $_B$	13.27 $_B$	**16.08** $_B$	**15.41** $_B$	11.50 $_B$	-
5	27.48 $_B$	21.65 $_B$	**27.84** $_B$	**27.82** $_B$	21.65 $_B$	19.79 $_B$
6	14.45 $_C$	**14.78** $_C$	**15.81** $_C$	**15.81** $_C$	14.13 $_C$	-
7	11.77 $_B$	11.30 $_B$	**14.47** $_B$	**13.31** $_B$	**13.00** $_B$	**12.18** $_B$
8	11.33 $_C$	10.33 $_C$	11.05 $_C$	**12.11** $_C$	10.75 $_C$	10.79 $_C$
9	10.25 $_C$	**10.41** $_C$	**10.91** $_C$	**10.79** $_C$	**10.51** $_C$	10.13 $_C$
10	8.14 $_B$	6.53 $_B$	6.80 $_B$	**8.44** $_B$	5.67 $_E$	-
11	13.87 $_B$	10.65 $_B$	**15.13** $_B$	**16.06** $_B$	13.66 $_E$	10.62 $_C$
12	14.14 $_C$	**15.92** $_C$	**16.33** $_C$	**16.01** $_C$	**15.48** $_C$	**15.13** $_C$
13	8.51 $_C$	7.93 $_C$	**9.03** $_C$	**8.95** $_C$	7.97 $_C$	8.01 $_C$
14	11.14 $_E$	**14.64** $_E$	**18.73** $_E$	**18.54** $_E$	**15.51** $_E$	**13.91** $_E$
15	11.82 $_B$	**12.83** $_B$	**14.87** $_E$	**14.75** $_E$	**14.54** $_E$	**13.22** $_E$
16	**23.67** $_C$	23.29 $_C$	23.54 $_C$	23.05 $_C$	22.81 $_C$	22.04 $_C$
17	7.05 $_C$	**8.06** $_C$	**7.31** $_C$	**7.34** $_C$	**7.47** $_C$	**7.85** $_C$
18	13.77 $_E$	**14.22** $_B$	**16.28** $_B$	**14.36** $_E$	**16.25** $_B$	**14.44** $_B$
19	19.30 $_B$	**19.41** $_B$	**21.31** $_B$	**19.92** $_B$	**19.53** $_B$	**17.01** $_B$
20	13.33 $_C$	**14.32** $_C$	**14.66** $_C$	**13.56** $_C$	**14.14** $_C$	-
21	3.81 $_H$	**3.92** $_H$	**3.94** $_H$	**3.96** $_H$	3.75 $_H$	**3.94** $_H$
22	22.07 $_B$	**22.40** $_B$	**25.00** $_B$	**23.41** $_B$	**24.02** $_B$	20.36 $_B$
23	8.49 $_C$	**11.39** $_C$	**12.11** $_C$	**10.44** $_C$	**11.41** $_C$	-
24	8.50 $_C$	**11.50** $_C$	**18.91** $_B$	**10.91** $_B$	**13.29** $_B$	-
25	8.53 $_B$	8.53 $_B$	**10.10** $_B$	**10.40** $_B$	**13.52** $_B$	**8.97** $_B$
26	8.06 $_B$	**9.77** $_B$	**16.75** $_C$	**8.75** $_B$	**11.75** $_C$	-
27	9.03 $_E$	**9.13** $_E$	**9.52** $_E$	**9.46** $_E$	**9.39** $_B$	**9.06** $_E$
28	33.75 $_B$	24.87 $_B$	**34.23** $_B$	**34.58** $_B$	**33.90** $_B$	22.41 $_B$
29	17.08 $_C$	**17.40** $_C$	**19.43** $_C$	**19.40** $_C$	16.94 $_C$	**19.84** $_C$
30	35.89 $_B$	28.87 $_B$	32.22 $_B$	**38.27** $_B$	28.58 $_B$	25.83 $_B$

Table 5: Summary of the best single precision performance results (in GFLOPS) on the Tesla M2050. Subscripts show the corresponding storage format: C (CSR), E (ELL), H (HYB) and B (BELLPACK).

mats agrees with the conclusions and observations explained above for single precision.

Figure 15 displays the performance achieved using the CSR format. In most of the cases, reordered matrices outperform the original ones. For example, D and D_{set} reorderings increase the performance of 20 matrices of the testbed. Best results are again observed for those reorderings whose nonzeros are located in closer positions within each row (matrices 12, 23, 24 and 26). Speedups up to 2× (matrix 26 and D reordering) are reached. Average performance improvement ranges from 0.5% using METIS to 7.2% when D is considered.

ELL format results are shown in Figure 16. A bandwidth reduction in the matrices when using this format has a big influence on the SpMV performance. For example, reorderings of matrix 25 reach speedups up to 1.6×. As we expect, according to the previous results, the best overall behavior is obtained by D and RCM techniques, improving on average the performance 9.5% and 7% respectively.

Figure 17 shows the results obtained using the HYB kernel. Because HYB format is a combination of ELL and COO formats, a bandwidth reduction has also a big impact on the performance with this format. Speedups higher than 1.4× were reached (matrix 24), with average improvements that vary from 0.9% (METIS) to 7.2% (D).

Matrix ID	Original	AMD	D	D_{set}	RCM	METIS
1	6.99 $_C$	6.99 $_C$	6.63 $_C$	6.49 $_C$	**7.00** $_C$	-
2	5.81 $_B$	**6.54** $_B$	**6.98** $_B$	**6.95** $_B$	**6.55** $_B$	**6.03** $_B$
3	**21.24** $_B$	16.93 $_B$	20.60 $_B$	20.71 $_B$	20.09 $_B$	14.95 $_B$
4	10.58 $_B$	10.67 $_B$	**12.40** $_B$	**11.83** $_B$	9.47 $_B$	-
5	**18.00** $_B$	14.57 $_B$	17.86 $_B$	17.80 $_B$	14.50 $_B$	13.17 $_B$
6	11.28 $_C$	**11.63** $_C$	**11.71** $_C$	**11.70** $_C$	11.15 $_C$	-
7	10.09 $_E$	9.36 $_B$	**12.13** $_B$	**11.03** $_B$	**10.95** $_B$	**10.13** $_B$
8	9.04 $_C$	8.22 $_C$	9.01 $_C$	**9.73** $_C$	8.91 $_C$	8.78 $_C$
9	**8.58** $_C$	8.37 $_C$	8.46 $_C$	8.41 $_C$	8.41 $_C$	8.31 $_C$
10	7.41 $_B$	6.06 $_B$	6.20 $_B$	**7.81** $_B$	5.12 $_B$	-
11	9.49 $_B$	8.81 $_B$	**10.89** $_B$	**11.13** $_B$	8.76 $_E$	8.51 $_C$
12	11.35 $_C$	**11.98** $_C$	**11.84** $_C$	**11.70** $_C$	**11.90** $_C$	**11.89** $_C$
13	6.96 $_C$	6.78 $_C$	**7.04** $_C$	**6.97** $_C$	6.76 $_C$	6.85 $_C$
14	9.01 $_E$	**9.48** $_E$	**12.04** $_B$	**10.63** $_E$	**9.75** $_E$	**9.54** $_E$
15	7.01 $_B$	**9.28** $_B$	**11.01** $_E$	**10.89** $_E$	**10.47** $_E$	**9.55** $_E$
16	**16.36** $_B$	15.72 $_C$	16.24 $_B$	15.79 $_C$	15.01 $_C$	14.66 $_C$
17	6.10 $_C$	**6.34** $_C$	**6.25** $_C$	6.10 $_C$	**6.47** $_C$	**6.13** $_C$
18	9.79 $_E$	**10.09** $_B$	**11.96** $_B$	**10.87** $_E$	**11.79** $_B$	**10.61** $_B$
19	14.64 $_B$	14.59 $_B$	**15.63** $_B$	**14.92** $_B$	14.53 $_B$	12.85 $_B$
20	9.40 $_C$	**9.79** $_C$	**10.41** $_C$	**9.64** $_C$	**9.44** $_C$	-
21	3.19 $_H$	**3.35** $_H$	**3.41** $_H$	**3.42** $_H$	3.01 $_H$	**3.43** $_H$
22	16.01 $_B$	**16.40** $_B$	**17.77** $_B$	**17.43** $_B$	**17.21** $_B$	14.98 $_B$
23	7.16 $_C$	**8.88** $_C$	**9.01** $_C$	**8.81** $_C$	**8.81** $_C$	-
24	7.09 $_C$	**9.32** $_C$	**13.55** $_B$	**8.83** $_B$	**10.64** $_B$	-
25	6.66 $_B$	**6.84** $_B$	**7.99** $_B$	**7.38** $_E$	**9.44** $_E$	**7.16** $_B$
26	6.50 $_B$	**8.09** $_C$	**11.71** $_C$	**7.75** $_B$	**8.86** $_C$	-
27	5.89 $_E$	**5.95** $_E$	**6.40** $_E$	**6.32** $_E$	**6.16** $_E$	**5.98** $_E$
28	19.56 $_B$	16.46 $_B$	**19.78** $_B$	**19.70** $_B$	19.50 $_B$	14.31 $_B$
29	12.11 $_C$	**12.48** $_C$	**13.70** $_C$	**13.67** $_C$	11.85 $_C$	**14.13** $_C$
30	22.74 $_B$	18.03 $_B$	20.43 $_B$	**24.32** $_B$	18.10 $_B$	17.45 $_B$

Table 6: Summary of the best double precision performance results (in GFLOPS) on the Tesla M2050. Subscripts show the corresponding storage format: C (CSR), E (ELL), H (HYB) and B (BELLPACK).

Finally, the results considering the BELLPACK format are displayed in Figure 18. These results agree with the observations obtained previously. That is, first, the higher performance is always obtain by FEM matrices with dense block sub-structures (original matrices 3, 28 and 30). Secondly, D and D_{set} obtain the best results overall. In particular, the average improvement using D and D_{set} is 14.1% and 9% respectively, reaching speedups up to 2.2× (matrix 24). And finally, we have also observed that AMD and METIS are specially inefficient with this format as it is shown by the average degradation of about 6% and 15% in the performance respectively. We have detected that AMD and METIS reorderings break the small dense sub-blocks of the original FEM matrices (matrices 3, 5, 11, 28 and 30).

In order to summarize the results, Table 6 shows the best performance obtained for each matrix and reordering technique considering all the storage formats. Results are very similar to those obtained with single precision arithmetic.

There are only 4 matrices for which reorderings do not improve their performance. In particular, we must emphasize the results obtained by D and D_{set}, whose reorderings outperform 22 and 24 original matrices of the testbed respectively. Note that in most of the cases the highest performance is achieved considering CSR or BELLPACK, which coincide with the best formats

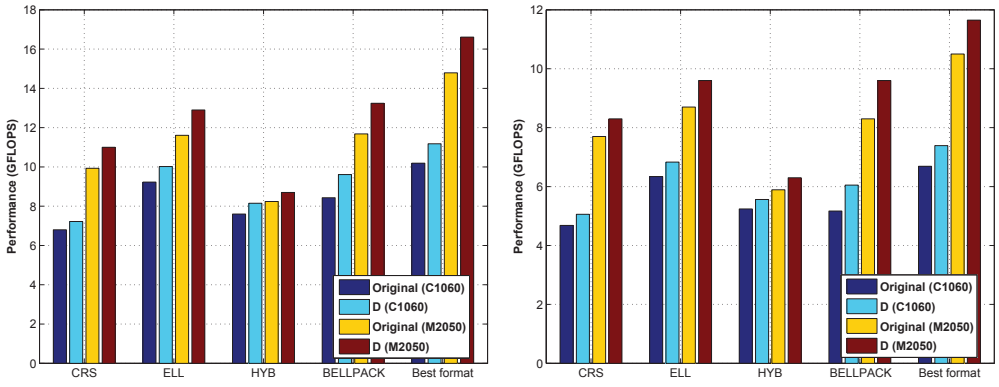

Figure 19: Performance comparison between GPUs (in GFLOPS): Tesla C1060 and Tesla M2050 (Fermi architecture). On the left single-precision performance, on the right double-precision results.

for D and D_{set}. Speedups up to 1.9× have been reached (matrix 24).

Overall average improvement is -2.5%, 10.4%, 7.2%, 1.5% and -8.1% for AMD, D, D_{set}, RCM and METIS respectively. AMD and METIS reorderings reduce the average performance with respect to the original matrices. In both cases the problem was that these reorderings break the small dense sub-blocks of the original FEM matrices, degrading the performance when using the BELLPACK format (see results of matrices 3, 5, 11, 28 and 30).

4.3. Comparison between GPUs

Finally, a comparison between the performance results obtained using both GPUs (Tesla C1060 and M2050) is presented. For illustrative purposes Figure 19 only shows the average performance obtained by the original matrices and the D reorderings. Focusing on single precision results, we have detected that speedups always higher than 1.4× are reached using the Tesla M2050 with respect to the C1060. However, there are small differences in the performance when the HYB storage format is considered. We think that this is caused by the performance of the COO parts of the matrices that use this format. Note that HYB is a combination of ELL and COO formats (see Section 3.2).

The same trend is observed when using double precision arithmetic. In this case, the differences in the performance of both GPUs increase. For example, the performance of the original matrices with CSR on the Tesla M2050 is 1.6× the one measured on the C1060.

5. Conclusions

In this paper we have explored the performance optimization of the SpMV on two different GPUs using reordering techniques. With this objective some of the most successful reordering techniques have been considered. Until now these techniques have been only tested on CPUs. Moreover, four different sparse matrix storage formats have been analyzed using both single and double precision floating-point arithmetic.

According to the performance evaluation results several conclusions can be made. First, we have detected that reordering techniques have a big impact on the SpMV performance using

GPUs. Considering CSR format, we have observed that the best results are always obtained by reorderings whose patterns show a higher level of clustering of the nonzeros within each row with respect to the original matrices. This fact will lead to access to closer elements of the vector when the SpMV operation is performed, improving the spatial locality. With ELL and HYB formats the most important factor that influences the SpMV performance is a bandwidth reduction in the matrices. Note that ELL format is only efficient when the maximum number of nonzeros per row does not substantially differ from the average. BELLPACK is the storage format for which reordering techniques have the most influence. This is caused by a limitation in the BELLPACK format: it can only be successfully applied to matrices that have small dense block sub-structures. D and D_{set} reorderings are the best choices when considering this format. This behavior was expected because the goal of this reordering technique is to increase the grouping of nonzero elements in the sparse matrix pattern, which will favor the creation of small dense sub-blocks. On the other hand, we have observed that METIS is specially inefficient with this format. Results point out that the typical arrow-shaped matrices generated by METIS break the small dense sub-blocks of the original FEM matrices.

Secondly, considering only the formats studied by Bell and Garland [5] (CSR, ELL and HYB), we have detected that there is no one storage format prevailing over the others. However, they found that HYB format is generally the fastest format for a broad set of unstructured matrices, and from our experiments on both GPUs we can not state that. If we consider the best performance obtained for each matrix and reordering technique considering all the storage formats (including BELLPACK), in most of the cases the highest performance is achieved considering CSR or BELLPACK.

Thirdly, there is no a reordering technique that in most of the cases performs better than the others. However, D and D_{set} show clearly the best overall behavior. For example, considering single-precision arithmetic, D reorderings improve on average the SpMV performance of the original matrices a 9.8% and 12.3% on the Tesla systems C1060 and M2050 respectively. Noticeable speedups up to 2.6× are reached. We must emphasize that reordered matrices outperform the original ones in most of the cases.

And finally, focusing on the average SpMV performance, we have detected that speedups always higher than 1.4× are reached using the Tesla M2050 with respect to the C1060. However, there are small differences in the performance when the HYB storage format is considered. We think that this is caused by the performance of the COO parts of the matrices that use this format.

As final remark we can state that reordering techniques are advisable in order to improve the performance of the SpMV operation on GPUs.

Acknowledgements

This work was supported by Xunta de Galicia (Spain) through the project 09TIC002CT. Authors wish to thank J. W. Choi and R. W. Vuduc for their support and help with the BELLPACK codes.

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[2] M. M. Baskaran and R. Bordawekar. Optimizing sparse matrix-vector multiplication on GPUs. Technical report, IBM Research Report RC24704 (W0812-047), 2008.

[3] M. Belgin, G. Back, and C. J. Ribbens. Pattern-based sparse matrix representation for memory-efficient SMVM kernels. In *Proc. of the Int. Conf. on Supercomputing*, pages 100–109, 2009.

[4] N. Bell and M. Garland. Efficient sparse matrix-vector multiplication on CUDA. Technical report, NVIDIA, 2008.

[5] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proc. of ACM/IEEE Conf. Supercomputing (SC)*, 2009.

[6] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In *Proc. of ACM SIGGRAPH*, pages 171–178, 2005.

[7] A. Buttari, V. Eijkhout, J. Langou, and S. Filippone. Performance optimization and modeling of blocked sparse kernels. *Int. Journal of High Performance Computing Applications*, 21(4):467–484, 2007.

[8] J. W. Choi, A. Singh, and R. Vuduc. Model-driven autotuning of sparse matrix-vector multiply on gpus. In *Proc. of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 115–126, 2010.

[9] A. L. G. A. Coutinho, M. A. D. Martins, R. M. Sydenstricker, and R. N. Elias. Performance comparison of data-reordering algorithms for sparse matrix-vector multiplication in edge-based unstructured grid computations. *Int. Journal for Numerical Methods in Engineering*, 66(3):431–460, 2006.

[10] E. Cuthill and J. McKee. *Several strategies for reducing the bandwidth of matrices*. Rose and Willoughby, 1972.

[11] T. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, 97(23), June 1997. http://www.cise.ufl.edu/research/sparse/matrices.

[12] B. B. Fraguela, R. Doallo, and E. L. Zapata. Memory hierarchy performance prediction for blocked sparse algorithms. *Parallel Processing Letters*, 9(3):347–360, 1999.

[13] E. J. Im, K. A. Yelick, and R. Vuduc. SPARSITY: Framework for optimizing sparse matrix-vector multiply. *Int. Journal of High Performance Computing Applications*, 18(1):135–158, 2004.

[14] V. Karakasis, G. Goumas, and N. Koziris. A comparative study of blocking storage methods for sparse matrices on multicore architectures. In *Proc. of IEEE Int. Conf. on Computational Science and Engineering*, pages 247–256, 2009.

[15] G. Karypis and V. Kumar. *METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*, 1997.

[16] J. J. Navarro, E. García, J. L. Larriba-Pey, and T. Juan. Block algorithms for sparse matrix computations on high performance workstations. In *Proc. IEEE Int'l. Conf. on Supercomputing*, pages 301–309, 1996.

[17] L. Oliker, X. Li, P. Husbands, and R. Biswas. Effects of ordering strategies and programming paradigms on sparse matrix computations. *SIAM Review*, 44(3):373–393, 2002.

[18] J. C. Pichel, D. B. Heras, J. C. Cabaleiro, and F. F. Rivera. Performance optimization of irregular codes based on the combination of reordering and blocking techniques. *Parallel Computing*, 31(8–9):858–876, 2005.

[19] J. C. Pichel, D. E. Singh, and J. Carretero. Reordering algorithms for increasing locality on multicore processors. In *Proc. of the IEEE Int. Conf. on High Performance Computing and Communications*, pages 123–130, 2008.

[20] A. Pinar and M. Heath. Improving performance of sparse matrix-vector multiplication. In *Proc. of Supercomputing*, 1999.

[21] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens. Scan primitives for GPU computing. In *Proc. of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, pages 97–106, 2007.

[22] M. M. Strout, L. Carter, J. Ferrante, and B. Kreaseck. Sparse tiling for stationary iterative methods. *Int. Journal of High Performance Computing Applications*, 18(1):95–113, 2004.

[23] S. Toledo. Improving memory–system performance of sparse matrix–vector multiplication. In *Proc. of the 8th SIAM Conf. on parallel processing for scientific computing*, March 1997.

[24] R. Vuduc and H. Moon. Fast sparse matrix-vector multiplication by exploiting variable block structure. In *High Performance Computing and Communications, volume 3726 of Lecture Notes in Computer Science*, pages 807–816, 2005.

[25] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix-vector multiply on emerging multicore platforms. In *Proc. of the ACM/IEEE Conf. on Supercomputing*, 2007.