Parallel Sparse Matrix Vector Multiplication on Intel MIC: Performance Analysis

Hana Alyahya[1(✉)], Rashid Mehmood[2], and Iyad Katib[1]

[1] Computer Science Department, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Kingdom of Saudi Arabia
Hana.alyahya@gmail.com, iakatib@kau.edu.sa
[2] High-Performance Computing Center, King Abdulaziz University, Jeddah, Kingdom of Saudi Arabia
RMehmood@kau.edu.sa

Abstract. Numerous important scientific and engineering applications rely on and are hindered by, the intensive computational and storage requirements of sparse matrix-vector multiplication (SpMV) operation. SpMV also forms an important part of many (stationary and non-stationary) iterative methods for solving linear equation systems. Its performance is affected by factors including the storage format used to store the sparse matrix, the specific computational algorithm and its implementation. While SpMV performance has been studied extensively on conventional CPU architectures, research on its performance on emerging architectures, such as Intel Many Integrated Core (MIC) Architecture, is still in its infancy. In this paper, we provide a performance analysis of the parallel implementation of SpMV on the first-generation of Intel Xeon Phi Coprocessor, Intel MIC, named Knights Corner (KNC). We use the offload programming model to offload the SpMV computations to MIC using OpenMP. We measure the performance in terms of the execution time, offloading time and memory usage. We achieve speedups of up to 11.63x on execution times and 3.62x on offloading times using up to 240 threads compared to the sequential implementation. The memory usage varies depending on the size of the sparse matrix and the number of non-zero elements in the matrix.

Keywords: SpMV · Intel Many Integrated Core Architecture (MIC) KNC · OpenMP · CSR · Xeon Phi

1 Introduction

Numerous important scientific, engineering and smart city applications require computations of sparse matrix-vector multiplication (SpMV) [1–5]. The SpMV operation is also an important part of many iterative solvers of linear equation systems, both stationary (e.g. Jacobi method) and non-stationary (e.g., Conjugate Gradient (CG)) [6]. The performance of SpMV is affected by factors including the storage format used to store the sparse matrix, the specific computation algorithm and its implementation. SpMV is considered a bottleneck due to its intensive computational and storage needs. Sparse matrices that arise from real life problems typically are large but consist of a

R. Mehmood et al. (Eds.): SCITA 2017, LNICST 224, pp. 306–322, 2018.
https://doi.org/10.1007/978-3-319-94180-6_29

relatively small number of nonzero elements. Efficient storage formats are required to store only the nonzero elements such that the use of memory is minimized while providing flexible and fast access to the matrix nonzero elements. Many sparse storage formats have been proposed over the years, well-known of these include, among others, the Coordinate format (COO), Compressed Sparse Row (CSR) format, Modified Sparse Row (MSR), Modified MTBDD format, and the Diagonal format [7–9].

The design of the current systems brings new challenges and opportunities. Compared to systems over the last years, today's systems show that while the number of cores increases the performance get better [10]. The multicore, many-core and storage capabilities allow developers to optimize their algorithms and benefit from those technologies. Many Integrated Core (MIC) architecture is a highly parallel engine and efficient processor architecture that achieve high performance through utilization of large of number of cores like vector register and high bandwidth on package memory. Intel Knights Corner (KNC) is the name of the first generation based on MIC architecture. The second generation of Intel Xeon Phi will be based on Intel Knights Landing (KNL) [11] chip and it will be available as stand-alone processor in addition to coprocessor.

In this paper, we provide a performance analysis of parallel implementation of SpMV on the first-generation of Intel Xeon Phi Coprocessor named Knights Corner (KNC). We used the offload programming model to offload the SpMV to MIC. OpenMP directive constructs was used for parallelization. The well-known Compressed Row Storage (CSR) format was chosen to store the sparse matrix efficiently. To measure the performance, we have calculated the execution time, offloading time and memory usage. The experimental results show that the performance of the parallel implementation achieved up to 11.63x performance gain on execution time and 3.62x on offloading time compared to the sequential implementation. The memory usage varies depending on the size of the sparse matrix and the number of non-zero elements in the matrix.

The rest of the paper is organized as follows: In Sect. 2, background on the SpMV computation and Intel MIC architecture is presented. Section 3 reviews the literature related to parallel implementation of SpMV. Section 4 explains the methodology used in this paper. Section 5 discusses the results and gives performance analysis of the SpMV. Section 6 concludes the paper.

2 Background

2.1 Sparse Matrix Vector Multiplication (SpMV)

The sparse matrix vector multiplication kernel is shown in Eq. (1) where A is a square sparse matrix N $\times$ N, x and y are vectors of length N. The matrix A is multiplied by vector x and added to vector y

$$y = y + Ax \tag{1}$$

Due to the irregular pattern of the non-zero values in the sparse matrix A, the SpMV considered to be one of the most time-consuming kernel. As result, the performance of the SpMV is poor. The compressed row storage format CSR is one of the solution to efficiently store the sparse matrices and reduce the memory overhead. CSR store the sparse matrices as follow: it has three arrays, val[nnz] array of size nnz where nnz is the number of non-zero elements in matrix A. val[nnz] array is used to store the value of non-zero elements. Col_in[nnz] is an array of size nnz and it stores the column indices of non-zeros. Row_ptr[n + 1] is an array of size n + 1 and it stores non-zeros in each row [8].

2.2 Intel Many Integrated Core Architecture (MIC)

The Intel Many Integrated Core Architecture is an architecture developed by Intel company. The key feature of this architecture is that in one chip, there are many intel® processor cores. Another advantage of this architecture is that it supports many programming languages such as the standard C, Fortran, and C++. The flexibility of compiling and running the code in any of Intel® Xeon® processors is also an important feature. In addition, it supports the most widely used parallel programming models such as OpenMP and MPI [12]. The Intel products that based on this architecture are more likely used in the high-performance computing applications as well as in supercomputers [13].

Intel Xeon Phi coprocessor is based on Intel MIC architecture. It supports up to 61 small x86 cores that works together. It has 8 memory controllers and support up to 16 GDDR channels. It has a transfer speed of 5.5GT/s in theory. Intel Xeon Phi has two level of cache memory. The instruction level cache with size of 32 KB and the data cache with size of 32 KB [14]. Figure 1 shows an overview of Intel MIC architecture.

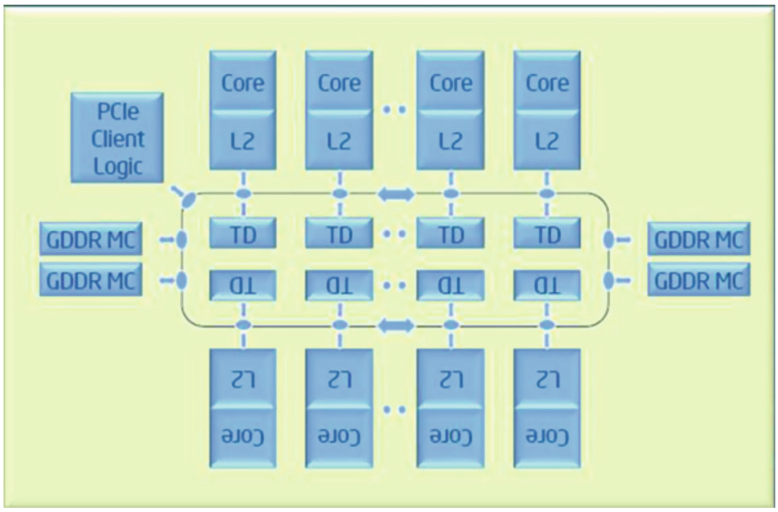

Fig. 1. Overview of Intel MIC architecture [6]

Xeon Phi has two execution modes: offload execution and native (coprocessor) execution [15]. In the offload mode, the host send part of the code to xeon phi and the output data is sent back from the coprocessor to xeon. Whereas, in the native mode, the code is run natively in the coprocessor. Figures 2 and 3 shows the two modes.

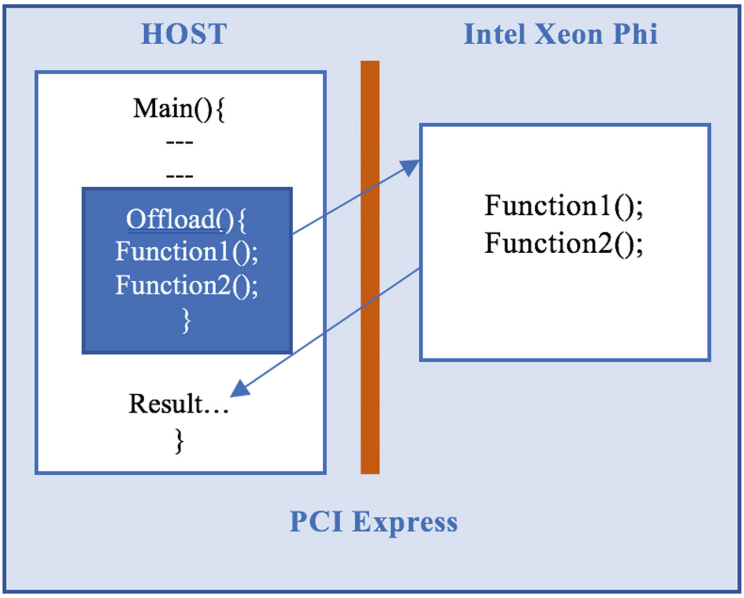

Fig. 2. Offloading mode in Xeon Phi

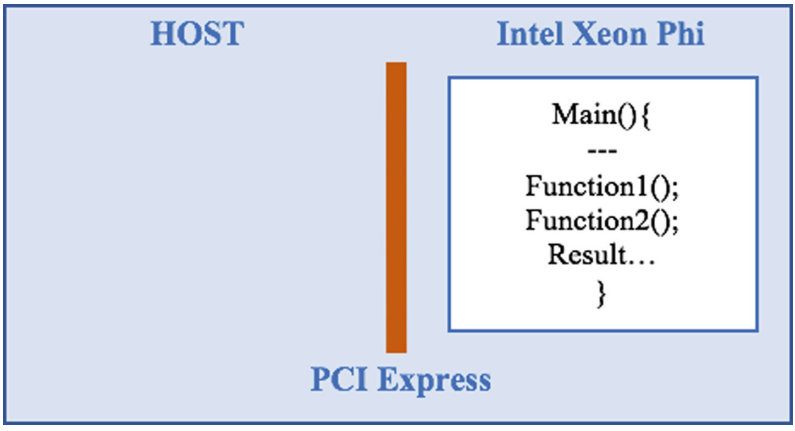

Fig. 3. Native mode in Xeon Phi

3 Related Work

Numerous studies have been done in the SpMV computation as it is used in many scientific and engineering applications. Ye et al. [9] report an implementation of SpMV computations on Intel MIC architecture using OpenMP, MPI, and hybrid MPI/OpenMP models. Their study shows that the hybrid model performs well on Intel MIC architecture. In [15], the authors implemented the SpMV on CPU, MIC, and GPU clusters and evaluate the performance of each cluster. They show that MIC outperform other accelerators using small number of MPI processes. However, the performance goes down when the number of MPI process increases due to communication overhead. Saule et al. [16] studies the performance of Intel Xeon Phi coprocessor for SpMV and focuses on the memory bandwidth. Their results show that Xeon Phi couldn't reach its peak performance due to the memory latency not the bandwidth. Xing et al. [17] presented a parallel implementation of SpMV on Intel MIC architecture using specialized ELLPACK-based storage format and three proposed load balancer. Their implementation has better performance than the best available implementation of SpMV on GPU.

In [18], the authors designed a new data structure for general sparse matrix storage to improve the performance of Sparse Matrix-Vector Multiplication (SpVM) on modern hardware. The authors implemented the SpVM using standard storage format CSR and their scheme on different architectures such as general CPUs, Intel Xeon Phi and GPGPU. The results show that their scheme outperforms the CSR on Intel Xeon Phi on most of the tested matrices. In [19], Maeda and Takahashi evaluated the performance of parallel Sparse Matrix-Vector Multiplication (SpVM) on different architectures such as CPU, Intel MIC, and GPU clusters. They used CSR storage format to store the sparse matrices. The result shows that the performance of parallel SpVM using CPU cluster in comparison to single process is increased by 42.57. In some matrices the performance is low due to load imbalance and communication overheads. The performance of parallel SpVM on accelerators is higher than on CPU cluster in the matrices that have large amount of non-zero or when using small number of MPI processes. However, when the number of MPI processes become large, the performance of parallel SpVM on MIC is low due to communication overhead. To overcome, the authors proposed to apply the Segmented Scan (SS) method to MIC cluster to improve the parallel SpVM. As a result, the performance of imbalanced matrices with 64 MPI processes is increased. In [20], the authors analyzed and evaluated the performance of Sparse Matrix Vector Multiplication (SpVM) and Krylov methods on GPUs. They considered different methods for solving sparse linear systems with symmetric and non-symmetric matrices. They applied different storage format and show their impact on the performance of the iterative solvers.

4 Methodology

We collected the matrices from the University of Florida online matrix collection [21]. We collected square matrices only and ignore other matrices. The SpMV is based on the off-diagonal matrices only because we are implementing the Jacobi iterative method for future work. The sparse matrices are form different application domains such as

optimization problem, directed graph, undirected random graph, circuit simulation problem, undirected graph, directed weighted graph, undirected multigraph, computational fluid dynamics problems, structural problem, and electromagnetics problem. Table 1 shows the application and their abbreviation. For simplicity, we will use abbreviation in the remaining parts of the paper. The details of the matrices are given in Table 2. We mention the dimension, the non-zero elements, the non-zero elements per row, the non-zero elements off diagonal, and the application domain. Figures 4, 5, 6, and 7 plots sparsity structure of some matrices from the collection.

Table 1. Applications name and their abbreviation

Application name	Abbreviation
Optimization problem	OP
Directed graph	DG
Undirected random graph	URG
Circuit simulation problem	CSP
Undirected graph	UG
Directed weighted graph	DWG
Undirected multigraph	UMG
Computational fluid dynamics problem	CFDP
Structural problem	SP
Electromagnetics problem	EMP
Model reduction problem	MRP

Table 2. Sparse matrices properties

Name	Size	nnz	nnz/row	Off diagonal nnz	Application
nlpkkt240	28.0 M	401.2 M	14.33	373.2 M	OP
arabic-2005	22.7 M	640.0 M	28.14	420.8 M	DG
rgg_n_2_24_s0	16.8 M	132.6 M	7.90	88.4 M	URG
circuit5 M	16.8 M	50.3 M	10.71	33.6 M	CSP
delaunay_n24	16.8 M	50.3 M	3.00	33.6 M	UG
nlpkkt200	16.2 M	232.2 M	14.30	216.0 M	OP
wb-edu	9.8 M	57.2 M	5.81	38.0 M	DG
nlpkkt160	8.3 M	118.9 M	14.25	110.6 M	OP
indochina-2004	7.4 M	194.1 M	26.18	127.7 M	DG
ljournal-2008	5.4 M	79.0 M	14.73	51.9 M	DG
cage15	5.2 M	99.2 M	19.24	94.0 M	DWG
soc-LiveJournal1	4.8 M	69.0 M	14.23	45.7 M	DG
channel-500x100x100-b050	4.8 M	42.7 M	8.89	28.5 M	UG
kron_g500-logn21	2.1 M	91.0 M	43.41	14.1 M	UMG
HV15R	2.0 M	283.1 M	140.33	281.1 M	CFDP
wikipedia-20051105	1.6 M	19.8 M	12.08	13.2 M	DG

(*continued*)

Table 2. (*continued*)

Name	Size	nnz	nnz/row	Off diagonal nnz	Application
G3_circuit	1.6 M	4.6 M	2.92	3.0 M	CSP
Flan_1565	1.6 M	59.5 M	38.01	57.9 M	SP
af_shell10	1.5 M	27.1 M	17.96	25.6 M	SP
cage14	1.5 M	27.1 M	18.02	25.6 M	DWG
Hook_1498	1.5 M	31.2 M	20.83	29.7 M	SP
StocF-1465	1.5 M	11.2 M	7.67	9.8 M	CFDP
Geo_1438	1.4 M	32.3 M	22.46	29.7 M	SP
Serena	1.4 M	33.0 M	23.69	31.6 M	SP
in-2004	1.4 M	16.9 M	12.23	11.0 M	DG
atmosmodd	1.3 M	8.8 M	6.94	7.5 M	CFDP
hollywood-2009	1.1 M	57.5 M	50.46	38.0 M	UG
dielFilterV3real	1.1 M	45.2 M	40.99	44.1 M	EMP
bone010	986.7 K	36.3 M	36.82	35.3 M	MRP
ldoor	952.2 K	23.7 M	24.93	22.8 M	SP
audikw_1	943.7 K	39.3 M	41.64	38.4 M	SP
RM07R	381.7 K	37.5 M	98.16	37.1 M	CFDP

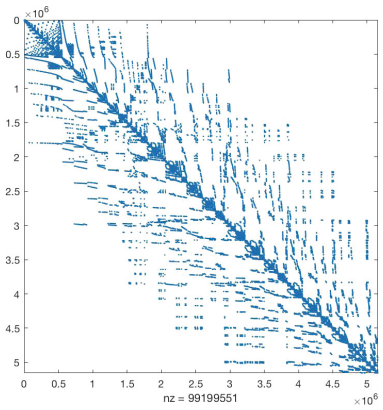

Fig. 4. Sparsity of matrix Cage15

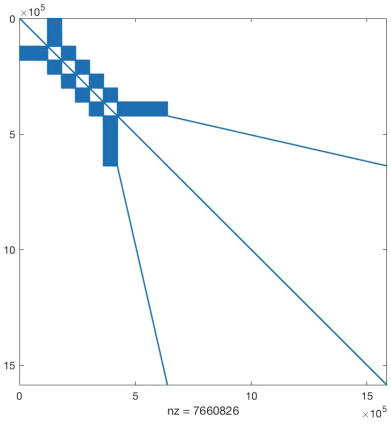

Fig. 5. Sparsity of matrix G3_circuit

We have a total of 32 sparse matrices all of them are in the Matrix Market format. We convert them to CSR and apply parallel SpMV computation. The parallelization process works as follow: the instruction is divided among the threads. Each thread will do the calculation and bring the results back. This process continue until the loop is finished. The number of threads used are 1, 4, 16, 32, 64, 128, and 240. Figure 8 shows the workflow.

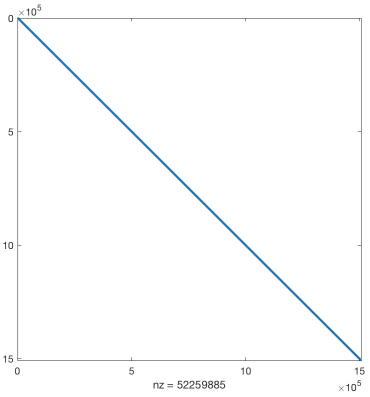

Fig. 6. Sparsity of matrix af_shell10

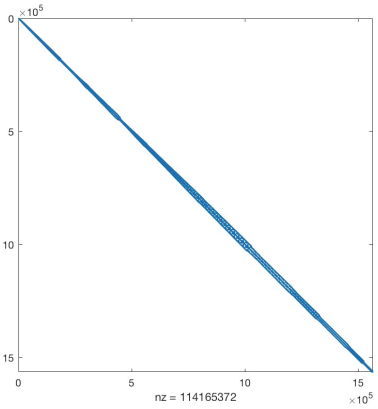

Fig. 7. Sparsity of matrix Flan_1565

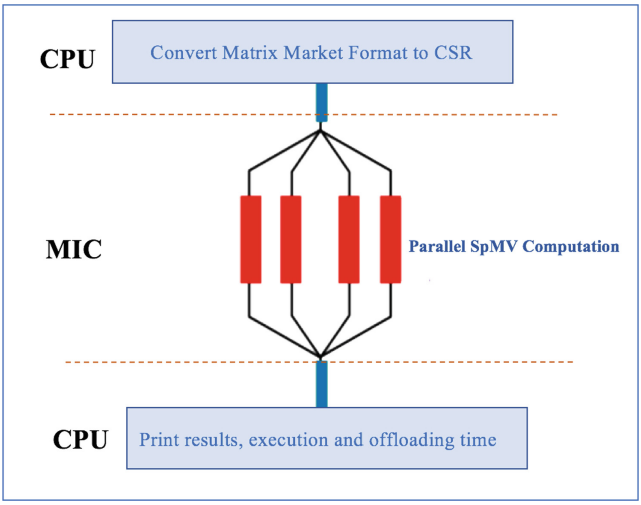

Fig. 8. Workflow

Algorithm 1 shows the pseudocode of the parallel SpMV. In line 2, an OpenMP pragma is added to the outer loop so, each thread will do the computation separately until they reach the end of the loop. The outer loop will begin with zero until it reaches the size of Matrix A which is n in this case. The inner loop will start from the first row that contains non-zero elements and end at last row that contains non-zero. Line 6 shows the main operation. Each row will be multiplied by the values of vector x and then will be added to vector y. When the outer loop finishes the result will be returned

and sent back to the CPU. Figure 9 shows the parallelization process of SpMV computation where y_1, y_2, and y_n represents the y vector, colored boxes represents the sparse matrix non-zero elements, x_1, x_2, and x_n represents the x vector, and thread $_0$, thread $_1$, and thread $_{th\_n}$ represents the thread number. As shown in the figure, each thread multiplies a row with whole vector x. At the end, the summation of y vector is performed.

Algorithm 1 Parallel Sparse Matrix Vector Multiplication With CSR, y=y+Ax

```
1: procedure SpMV-CSR(val, x, n, row_ptr, col_in)
2:     #pragma omp parallel for private(j)
3:     for i = 0 : n do
4:         y = 0.0
5:         for j = row_ptr[i] : row_ptr[i + 1] do
6:             y+ = val[j] * x[col_in[j]]
7:         end for
8:     end for
9: end procedure
```

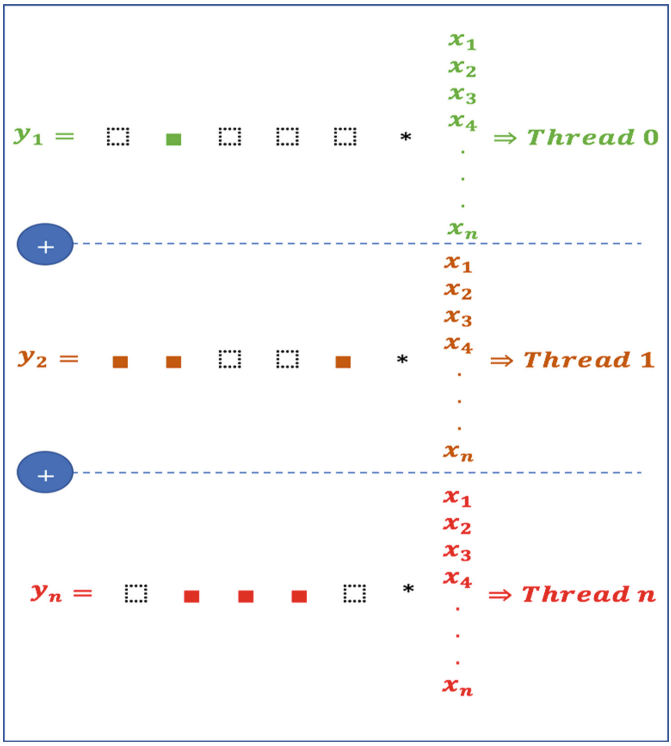

Fig. 9. Parallelization of SpMV

5 Results and Analysis

5.1 Environmental Setup

For experiments, we use Aziz supercomputer which is a high-performance computer located in King Abdul-Aziz University, Jeddah. It is one of the top 500 supercomputers in the world and one of the top 10 supercomputers in Kingdom of Saudi Arabia [22]. Table 3 shows the specification of tools used in the experiments.

Table 3. Specification of tools used

	Tools/Library	Version
OS	Linux	2.6.32-358.23.2.el6.x86_64
OpenMP	OpenMP	17.0.2
C Compiler	Intel C Compiler (icc)	17.0.2

5.2 Experimental Results

To implement the parallel SpMV efficiently on Intel MIC architecture we have followed three steps. Firstly, we read the sparse matrix in CSR format since the downloaded matrices are in the Matrix Market (MM) format. This is done using the CPU as it is having large memory compared to MIC. Secondly, when the matrix and vector is ready, we offloaded the part of the code that has the SpMV computation to MIC. After the SpMV offloaded to MIC, we use OpenMP pragmas to parallelize the "for" loops. Finally, the results are sent from the coprocessor to the host and the host will print the results and the execution time. We calculate the execution time and offloading time using different number of threads 1, 4, 8, 16, 32, 64, 128, and 240. In addition, we calculate amount of memory used by each matrix. Note that the execution time is the time taken to execute the SpMV computation and offloading time is the time taken to offload the SpMV computation to MIC and that includes the execution time. For simplicity, we divided the matrices into four groups according to their sizes, Groups 1, 2, 3, and 4, each group have eight matrices. The details of these matrices have been given earlier in Sect. 4.

Figures 10, 11, 12, and 13 show the execution time against the number of threads for Groups 1, 2, 3, and 4, respectively. It can be clearly seen in Fig. 10 that using 240 threads gives the best execution time among others. On average, the execution time of parallel implementation with 240 threads is 4.59x faster than the serial one. Group No.2 has exactly the same behavior as the first one except the last three matrices. The best execution time can be found in 16 threads for matrix named kron_g500-logn21 and 128 threads for matrices HV15R and wikipedia-20051105. Group No.3 and 4 are different, the execution time varies from one matrix to another but still the parallel execution is better than the serial one.

Figures 14, 15, 16, and 17 show the offloading time against the number of threads to group 1, 2, 3, and 4. For offloading time, all four groups have the same behavior of execution time. The best offloading time in Fig. 14 is when using 240 threads. The

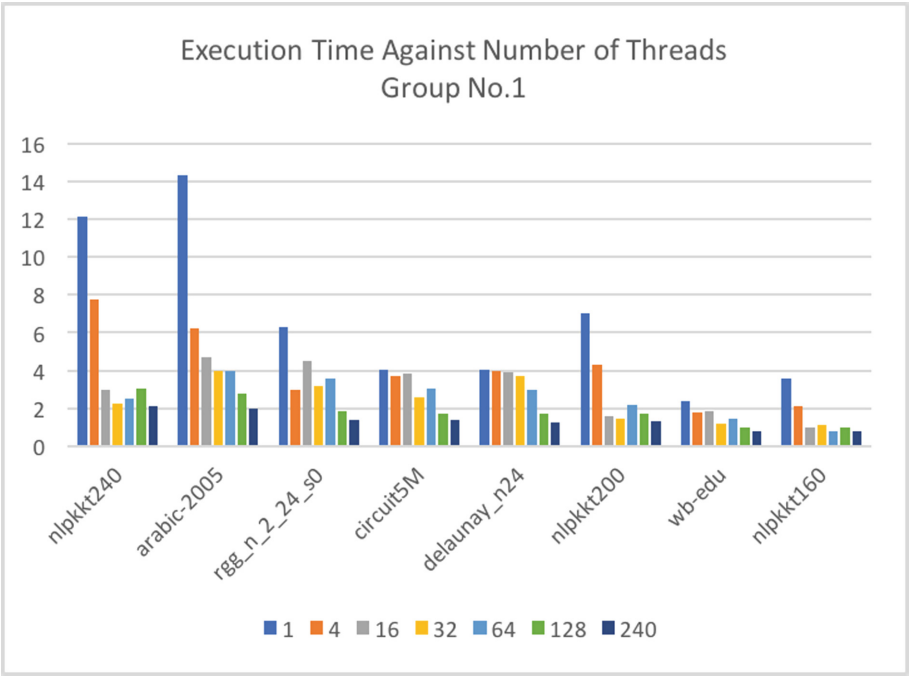

Fig. 10. Execution time against number of threads Group No. 1

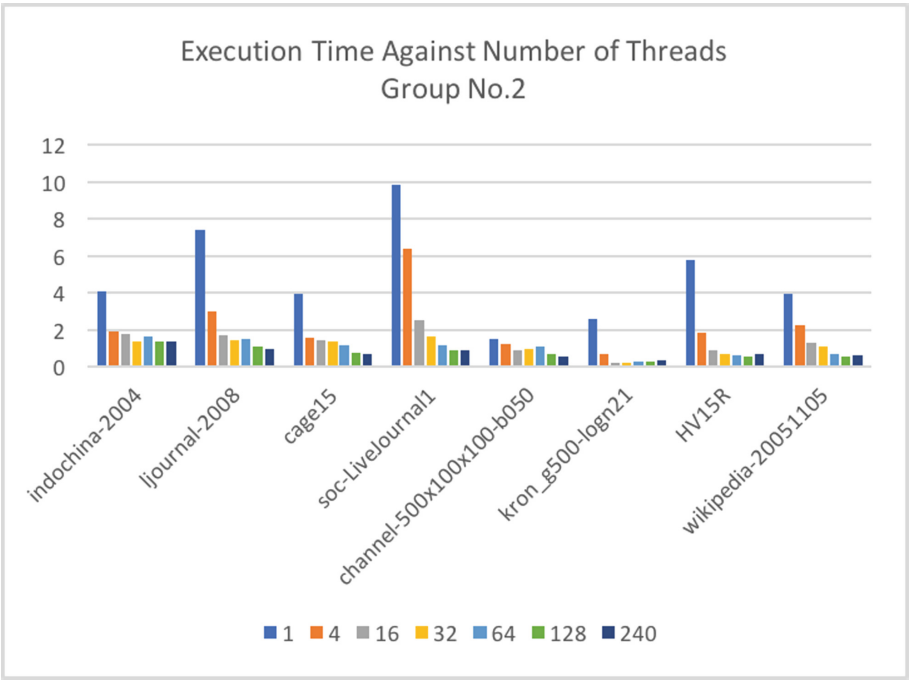

Fig. 11. Execution time against number of threads Group No. 2

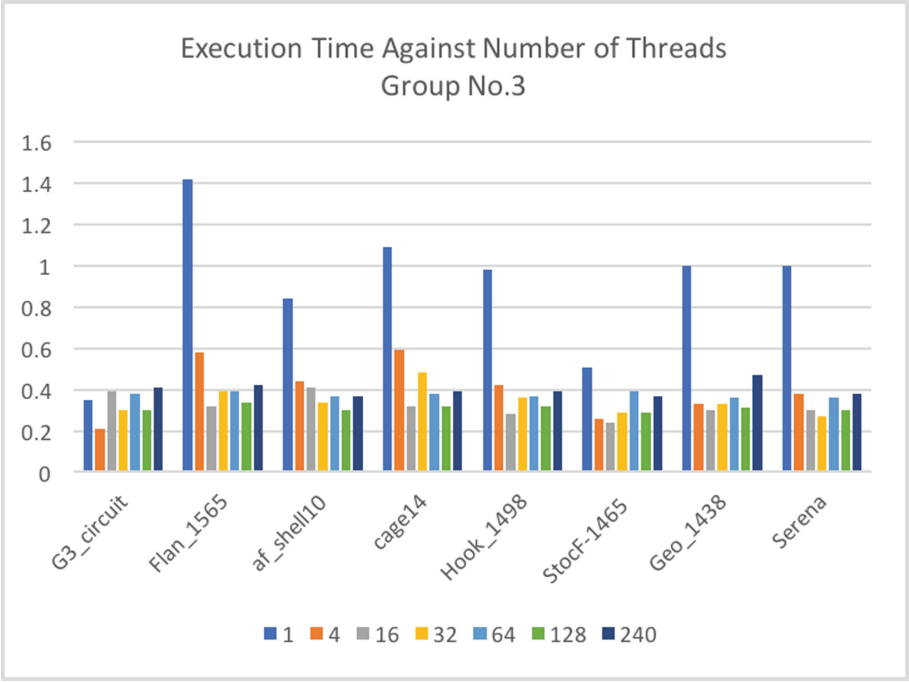

Fig. 12. Execution time against number of threads Group No. 3

second group is the same except the las three matrices which has best offloading time when using 16 and 128 threads. Although, the offloading time in Group No. 3 and 4 varies from one matrix to the another, the parallel implementation still better than serial one.

Finally, Fig. 18 shows the memory usage against the number of off diagonal non-zeros. It can be clearly seen that the off diagonal non-zero has a strong effect in the memory. The larger the off diagonal non-zeros are the larger memory is needed. However, that doesn't apply to some matrices which may be affected by other factors rather than the off diagonal non-zeros.

To summarize, the execution time of the parallel implementation is 4.89x faster than the sequential implementation. The offloading time of the parallel implementation is 1.65x faster than the sequential one. The memory usage depends on the off diagonal non-zeros and some other factors.

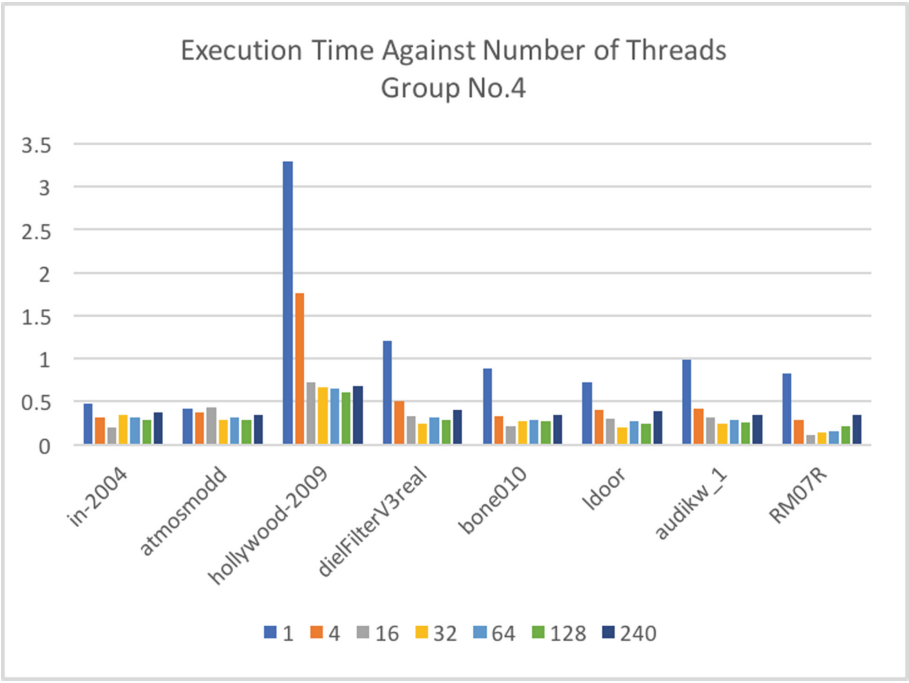

Fig. 13. Execution time against number of threads Group No. 4

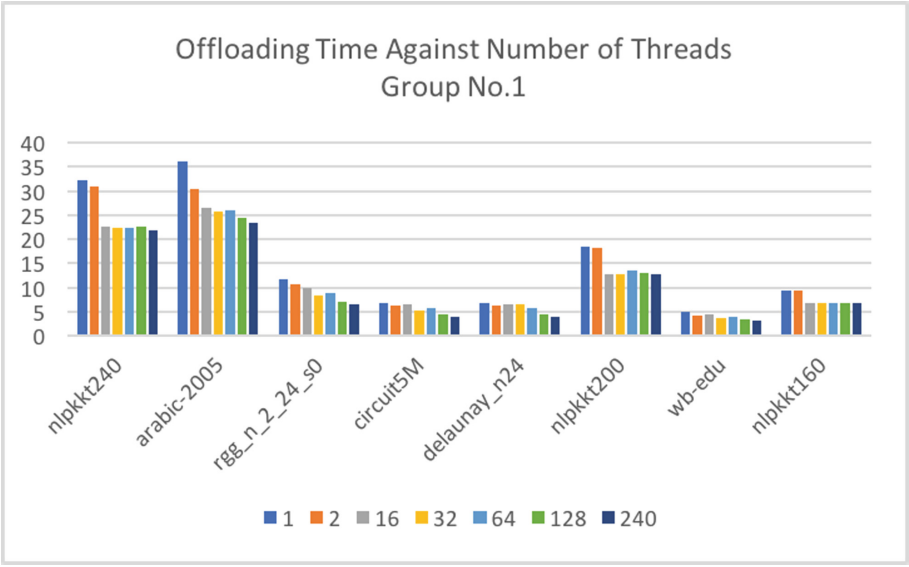

Fig. 14. Offloading time against number of threads Group No. 1

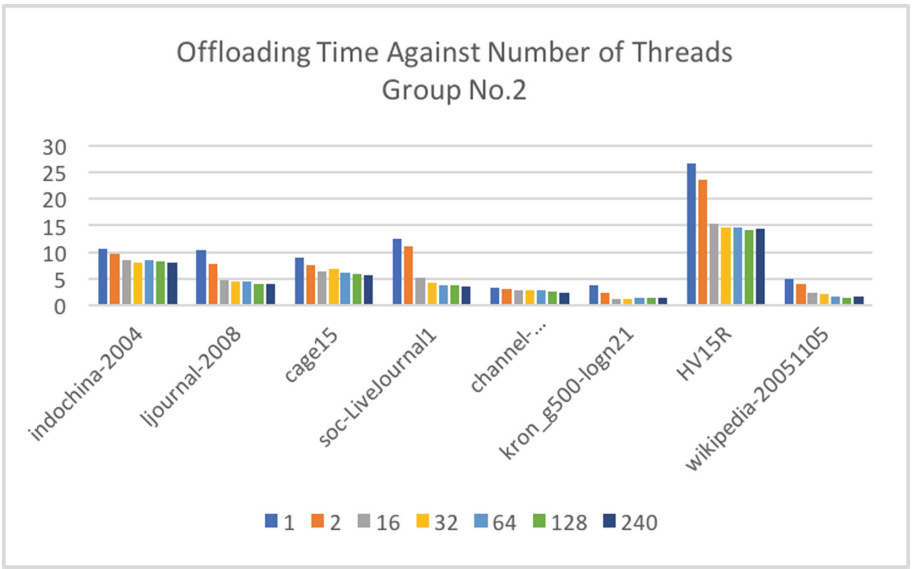

Fig. 15. Offloading time against number of threads Group No. 2

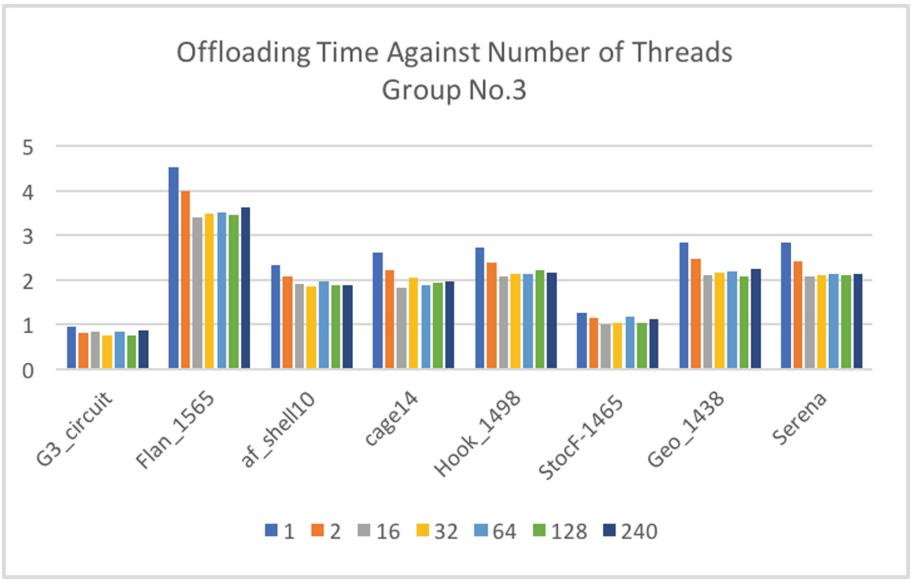

Fig. 16. Offloading time against number of threads Group No. 3

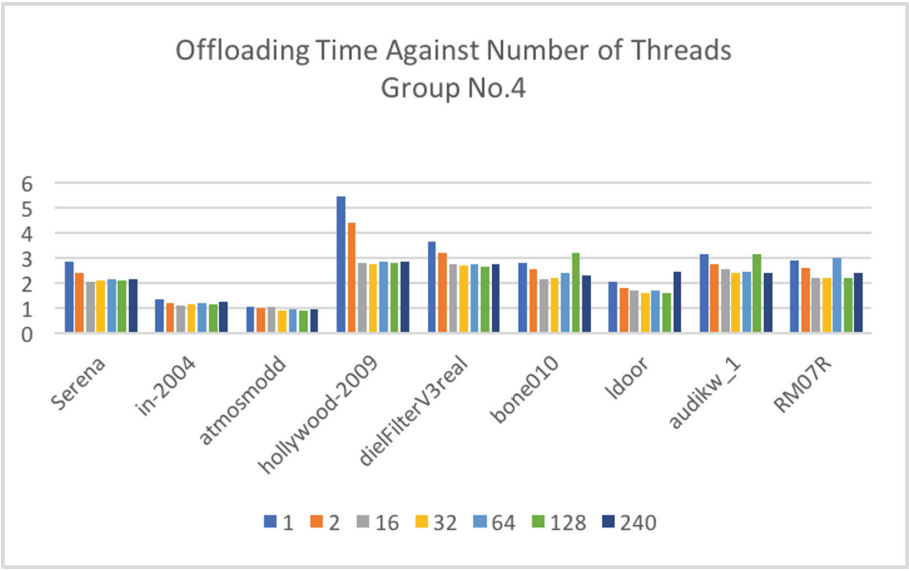

Fig. 17. Offloading time against number of threads Group No. 4

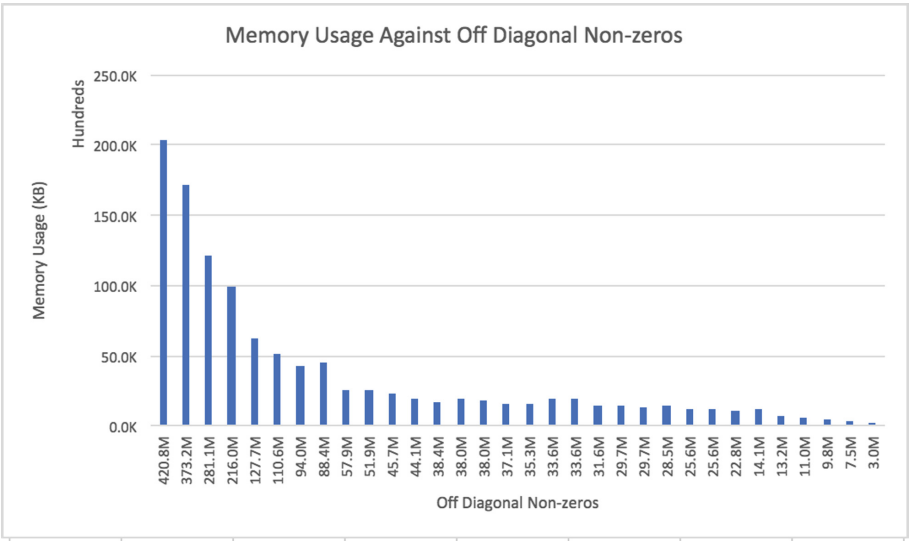

Fig. 18. Memory usage against off diagonal non-zeros

6 Conclusion

In this paper, we presented a parallel implementation of SpMV computation using Intel MIC architecture. The standard storage format CSR used to store the sparse matrices. To measure the performance, we use execution time, offloading time, and memory usage. The performance of the parallel SpMV achieved up to 11.63x on the Intel Xeon Phi coprocessor. In addition, the offloading time was improved by up to 3.62 times with parallel implementation. The memory usage varies depending on the off diagonal non-zero elements but in most cases the larger is the number of non-zeros the larger memory is needed.

Acknowledgments. The experiments reported in this paper were performed on the Aziz supercomputer at King AbdulAziz University, Jeddah, Saudi Arabia.

References

1. Mehmood, R., Lu, J.A.: Computational Markovian analysis of large systems. J. Manuf. Technol. Manag. **22**, 804–817 (2011)
2. Mehmood, R., Alturki, R., Zeadally, S.: Multimedia applications over metropolitan area networks (MANs). J. Netw. Comput. Appl. **34**, 1518–1529 (2011)
3. 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. **37**, 75–104 (2016)
4. Altowaijri, S., Mehmood, R., Williams, J.: A quantitative model of grid systems performance in healthcare organisations. In: ISMS 2010 - UKSim/AMSS 1st International Conference on Intelligent Systems, Modelling and Simulation, pp. 431–436 (2010)
5. Mehmood, R., Graham, G.: Big data logistics: a health-care transport capacity sharing model. Procedia Comput. Sci. **64**, 1107–1114 (2015)
6. Mehmood, R.: Disk-Based Techniques for Efficient Solution of Large Markov Chains (2004)
7. Banu, S.J.: Performance Analysis on Parallel Sparse Matrix Vector Multiplication Micro-Benchmark Using Dynamic Instrumentation Pintool, pp. 129–136 (2013)
8. Mehmood, R., Crowcroft, J.: Parallel iterative solution method for large sparse linear equation systems. Technical report Number UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, Cambridge, UK (2005)
9. Mehmood, R.: Serial disk-based analysis of large stochastic models. In: Baier, C., Haverkort, Boudewijn R., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) Validation of Stochastic Systems. LNCS, vol. 2925, pp. 230–255. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24611-4_7
10. Giles, M.B., Reguly, I.: Trends in high-performance computing for engineering calculations. Philos. Trans. R. Soc. A. **372**, 20130319 (2014)
11. Sodani, A., Gramunt, R., Corbal, J., Kim, H.S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., Liu, Y.C.: Knights landing: second-generation Intel Xeon Phi product. IEEE Micro **36**, 34–46 (2016)
12. Cramer, T., Schmidl, D., Klemm, M., Mey, D.: OpenMP Programming on Intel Xeon Phi Coprocessors: An Early Performance Comparison. Marc@Rwth, pp. 38–44 (2012)
13. Intel® Many Integrated Core Architecture - Advanced

14. Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., Wang, Y.: High-Performance Computing on the Intel® Xeon Phi™. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06486-4
15. Maeda, H., Takahashi, D.: Performance evaluation of sparse matrix-vector multiplication using GPU/MIC cluster. In: 2015 Third International Symposium on Computing and Networking, pp. 396–399 (2015)
16. Saule, E., Kaya, K., Atalyürek, U.V.Ç.: Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi (2013)
17. Liu, X., Smelyanskiy, M., Chow, E., Dubey, P.: Efficient sparse matrix-vector multiplication on x86-based many-core processors. In: Proceedings of 27th International ACM Conference on Supercomputing ICS 2013, p. 273 (2013)
18. Kreutzer, M., Hager, G., Wellein, G.: A unified sparse matrix data format for modern processors with wide SIMD units. SIAM J. Sci. Comput. **36**, 1–25 (2013). https://arxiv.org/abs/1307.6209v1
19. Maeda, H., Takahashi, D.: Parallel sparse matrix-vector multiplication using accelerators. In: Gervasi, O., et al. (eds.) ICCSA 2016. LNCS, vol. 9787, pp. 3–18. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42108-7_1
20. Ahamed, A.-K.C., Magoules, F.: Iterative methods for sparse linear systems on graphics processing unit. In: 2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 and IEEE 9th International Conference on Embedded Software Systems, pp. 836–842 (2012)
21. Search the University Florida Matrix Collection. http://yifanhu.net/GALLERY/GRAPHS/search.html
22. About Aziz. http://hpc.kau.edu.sa/Pages-About-Aziz-en2.aspx