



OPEN

# Implementation and optimization of SpMV algorithm based on SW26010P many-core processor and stored in BCSR format

Mengfei Ma<sup>1</sup>, Xianqing Huang<sup>1,2</sup>✉, Jiali Xu<sup>1,2</sup> & Dongning Jia<sup>3</sup>

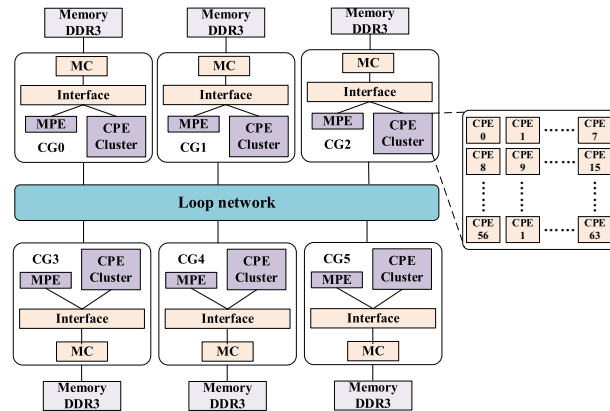
The irregular distribution of non-zero elements of large-scale sparse matrix leads to low data access efficiency caused by the unique architecture of the Sunway many-core processor, which brings great challenges to the efficient implementation of sparse matrix–vector multiplication (SpMV) computing by SW26010P many-core processor. To address this problem, a study of SpMV optimization strategies is carried out based on the SW26010P many-core processor. Firstly, we design a memorized data storage transformation strategy to transform the matrix in CSR storage format into BCSR (Block Compressed Sparse Row) storage. Secondly, the dynamic task scheduling method is introduced to the algorithm to realize the load balance between slave cores. Thirdly, the LDM memory is refined and designed, and the slave core dual cache strategy is optimized to further improve the performance. Finally, we selected a large number of representative sparse matrices from the Matrix Market for testing. The results show that the scheme has obviously speedup the processing procedure of sparse matrices with various sizes and sizes, and the master–slave speedup ratio can reach up to 38 times. The optimization method used in this paper has implications for other complex applications of the SW26010P many-core processor.

**Keywords** Sparse matrix–vector multiplication, Caching algorithm, SW26010P, Block compressed sparse row, High performance computing

High-performance computing, also known as supercomputing, is a vital frontier branch of computer science. Since the birth of electronic computers, the improvement of computing performance has been one of the core goals pursued by information industry practitioners. It is not only a critical symbol of a country's comprehensive scientific research level, but also provides the basis for national security, economic, social development and other sustainable development<sup>1,2</sup>. At present, the development trend of supercomputers is many-core and heterogeneous many-core. The computing capability of the new-generation Sunway supercomputer is provided by a SW26010P many-core processor that includes 6 core-groups (CGs), each of which includes one management processing element (MPE), and one  $8 \times 8$  CPE cluster, these main components are connected through a ring network, as shown in Fig. 1. When the program is running, the main process runs on the MPE. MPE is essentially a general-purpose processor, in which complex computing departments can specify to CPE, each CPE is a lightweight core and performs a single thread<sup>3</sup>. MPEs mainly include computation, control, communication, and I/O functions. CPEs are mainly used for computation. A CPE and MPE are called a master core and slave core, respectively. The CPEs in the CGs are interconnected with one another and with external interactions through an intra-array network. Each CPE has an independent instruction cache and 256 KB local data memory (LDM) that is consisted of SPM. The LDM of CPE is identical to a cache that can improve data access speed<sup>4,5</sup>.

Sparse matrix–vector multiplication (SpMV)  $y = Ax$  is a fundamental computational kernel in scientific and engineering computing, which has been widely used in many fields such as fluid dynamics simulation calculation, data mining, graphics, and image processing<sup>6,7</sup>. The core of scientific and engineering computation lies in solving systems of linear equations. However, solving systems of linear equations is very time-consuming, and most of the computation in solving systems of equations is related to SpMV. Hence, the computational performance of SpMV often dramatically impacts the overall application time consumption<sup>8,9</sup>. In order to adapt the underlying

<sup>1</sup>Information Research Department, Qingdao Marine Science and Technology Center, Qingdao, China. <sup>2</sup>Laoshan Laboratory, Qingdao, China. <sup>3</sup>College of Information Science and Engineering, Ocean University of China, Qingdao, China. ✉email: xqhuang0816@163.com



**Figure 1.** General architecture of the SW26010P many-core processor.

architecture of hardware accelerators, researchers focus on the reconstruction of SpMV algorithm to improve the computing performance of hardware accelerators. Such as Intel Xeon Phi<sup>10,11</sup>, general purpose Graphics Processor (GPGPU)<sup>12,13</sup>, advanced micro devices (AMD)<sup>14,15</sup>, field programmable gate array (FPGA)<sup>16,17</sup> and so on. With the advent of Sunway Taihulight supercomputer<sup>18</sup>, it is equipped with SW26010P many-core processor unique hardware architecture and has strong parallel computing capabilities. However, due to inconsistent hardware architecture, the optimization strategy of SpMV on GPU, AMD, Intel Xeon Phi and other processors cannot be applied to Sunway Supercomputer. The Sunway supercomputer is an important tool for solving major scientific problems and promoting technological innovation, mainly used for performing complex scientific calculations and simulations, such as climate models, physical experiment simulations, analysis of biomolecular structures, finite element analysis, computational fluid dynamics, circuit simulations, and so on. According to the literature<sup>19–21</sup>, SpMV is the core computational step in these applications; optimizing SpMV can significantly improve overall computational efficiency, thereby enhancing the Sunway supercomputer's capability to solve major scientific problems. The computational efficiency of SpMV under the heterogeneous many-core architecture of Sunway Supercomputer is still one of the bottlenecks in scientific and engineering computing. It is very important to study the high-performance SpMV based on SW26010P many-core processor architecture. In addition, the underlying algorithm and software ecology of Sunway supercomputer are not perfect, and only by improving the computational efficiency of the underlying algorithm can we better meet the current computing performance needs. As the underlying algorithm of numerical simulation computing, SpMV optimization strategy based on SW26010P many-core processor plays a good role in promoting the improvement of domestic software ecology<sup>22,23</sup>. The sparsity properties of sparse matrices vary greatly. A large number of discrete indirect addressing operations are introduced in the calculation process of SpMV, which makes a large number of system resources consumed in accessing memory. How to realize SpMV to read matrix data efficiently and use the local memory of SW26010P many-core processor reasonably is the problem we focus on.

SpMV transplantation and optimization work has been uninterrupted, especially with the rapid upgrade of computing hardware, new research hotspots have followed. The optimization of sparse matrix storage format and the optimization in combination with the current computer architecture are the hot topics of recent research, such as optimizing SpMV based on many-core heterogeneous platforms. In terms of optimization of sparse matrix storage format, Kreutz et al. proposed a new storage format SELL-C- $\sigma$  mainly from the perspective of improving vectorization performance, which effectively improves the storage efficiency and computational performance<sup>24</sup>. Liu et al. proposed a storage format CSR5 to improve the computation performance of irregular sparse matrix SpMV and conducted experiments on several heterogeneous many-core processors<sup>25</sup>. At the same time, he compared with the existing similar works, which showed an excellent optimization effect. Liu et al. analyzed the impact of different storage formats on SpMV and proposed an automatic SpMV tuning device SMAT, which can select and return the optimal storage format according to the characteristics of the specified sparse matrix to achieve the performance improvement of SpMV<sup>26</sup>. Bian et al. proposed a new sparse matrix storage format CSR2, which is a new single format for processor platforms with SIMD (single instruction multiple data) vectorization that has low complexity in transformation computation<sup>27</sup>. In terms of many-core heterogeneous platform optimization, Liu et al. proposed a parallel algorithm of SpMV based on CSR storage format based on SW26010 many-core processor, which was designed from the aspects of task division, LDM space division and adaptive excellence<sup>28</sup>. They innovatively proposed a dynamic and static buffer caching mechanism and a combination of dynamic and static task scheduling methods to improve the performance further. Li Y et al. took SW26010 as the platform to design and optimize the fine-grained parallel algorithm for SpMV at the thread level and instruction level parallel level, and conducted experiments on a large number of test sets, obtaining an average master–slave speedup ratio of 11.7 times<sup>29</sup>. Xiao et al. proposed a CASpMV based on the Sunway many-core processor for three main performance-limiting reasons storage limitations, load imbalance and irregular memory accesses<sup>30</sup>. They demonstrated the effectiveness and optimization efficiency of this work through extensive experiments. Sun et al. proposed an efficient SpMV calculation method SWCSR-SpMV for SW26010 processor<sup>31</sup>. They designed a dynamic preprocessing scheme to avoid the DMA reading operation to load useless x, reduce redundant memory access, and divide the many-core into smaller communication ranges.

To share the public data on the worker thread through high-speed data bus, and through a lot of experiments to prove that the scheme has a good optimization performance. Yca B et al. proposed a two-segment large-scale SpMV, called tpSpMV, and introduced a two-stage parallel execution technology for tpSpMV to solve the limit of computing scale, and proposed an adaptive partitioning method and parallelization design to alleviate the problem of high memory access delay<sup>32</sup>. The experimental results on Sunway Taihulight supercomputer show that the proposed scheme has an obvious acceleration effect.

To improve the underlying algorithm ecology of Sunway supercomputer, enhance the computational performance and speed of numerical simulation, a SpMV algorithm based on BCSR storage is designed in this paper. The rest of the paper is organized as follows. In “Proposed method” section describes the design and implementation of the program. In “Experiments” section presents the experimental setting, results and discussion. Finally, the paper concludes in “Conclusions” section.

## Proposed method

### Overall design

Considering the unique structure of SW26010P many-core processor and the special storage format of BCSR, we designed SpMV algorithm based on BCSR storage format for SW26010P many-core processor. The BCSR storage format is a method for storing sparse matrices, with advantages such as memory efficiency, computational efficiency, parallelism, reduced memory access conflicts, and ease of implementation. These characteristics align with the architectural features and optimization goals of Sunway processors, making the choice of BCSR storage format on Sunway processors aimed at maximizing performance and efficiency. In terms of memory efficiency, it is an improvement over CSR; BCSR stores the matrix by dividing it into blocks of a fixed size, which reduces gaps in memory and increases memory utilization, significantly reducing the number of times vector  $x$  is fetched during computation. Regarding computational efficiency and parallelism, the BCSR format is suitable for parallel computing because it allows multiple threads or cores to process different matrix blocks simultaneously, thereby improving parallelism. In terms of reducing memory access conflicts, due to the contiguous storage of blocks in BCSR, it can reduce conflicts between cache lines, improving cache efficiency, and the effect of BCSR is more pronounced in large-scale matrices with local density.

The algorithm model mainly comprises the following modules: storage format conversion module, task division module, LDM allocation module, calculation module, data verification module, and so on. The primary function of the storage format conversion module is to quickly convert the large-scale sparse matrix of CSR storage format into BCSR format according to the computing requirements, and make a preliminary preparation for efficient SpMV computation. The primary purpose of the task division module is to distribute computing tasks to 64 CPEs as much as possible to achieve load balancing and improve computing efficiency. Therefore, this paper adopts the dynamic partition strategy to achieve load balancing. The specific implementation method is to divide the computation tasks into the same block and put it into the task pool established in advance. Each CPE obtains the computation data from the task pool in turn. At the same time, the problem of resource conflict between processes is avoided by locking. The main purpose of the memory allocation module is to use the memory of SW26010P many-core processor properly. Due to the LDM space memory limit of SW26010P many-core processor, we have refined the LDM memory allocation of CPE to achieve the calculation performance of (SpMV)  $y = Ax$ . The computing module and data verification module are the calculation of tasks and verification of data correctness. The algorithm implementation model is shown in Fig. 2.

### Converting storage formats

#### Implementation of format conversion

This paper is based on the storage format of CSR to achieve BCSR format conversion to achieve SpMV optimization. Therefore, the first step in this paper is to convert the large-scale sparse matrix in CSR storage format into BCSR format for storage. In the BCSR format,  $val$  stores nonzero elements stored in a block in a row-first fashion,  $ind$  stores the index of the first nonzero element in the block, and  $ptr$  stores the starting index of the first block in  $ind$  for the current row-slice. Firstly, the non-zero elements of the input large-scale sparse matrix are scanned, and the distribution of non-zero elements is analyzed to obtain the best size of BCSR data block  $block\_size * block\_size$ . The number of existing data blocks in the sparse matrix  $block\_n$  is counted by the row-slice  $ls_i$ . Record the starting column label  $block\_col_i$  for each block. We call the number of the block in the slice of the current row  $ls\_block_i$ , where  $i \in [0, ptr[i+1]-ptr[i]]$ ; Further, we iterate over the data in the original CSR format to obtain the row number  $data_{row}$  and column number  $data_{col}$ . Calculate the block number  $block\_i$ , the row number  $block\_in_{row}$ , and the column number  $block\_in_{col}$  of the data block where the current nonzero element should be stored by scanning the matrix. Where  $block\_i$ ,  $block\_in_{row}$  and  $block\_in_{col}$  are calculated according to (1), (2) and (3):

$$block\_i = ptr[data_{row}/block\_size] + ls\_block_i \quad (1)$$

$$block\_in_{row} = data_{row} - block\_size * ls_i \quad (2)$$

$$block\_in_{col} = data_{col} - block\_col_i[ls\_block_i] \quad (3)$$

The above values of  $block\_i$ ,  $block\_in_{row}$  and  $block\_in_{col}$  can be used to obtain the  $val_{index}$  of  $val$  corresponding to BCSR and assign the value. The calculation formula is shown in (4).

$$val_{index} = [(block\_i * block\_size) + block\_in_{row}] * block\_size + block\_in_{col} \quad (4)$$

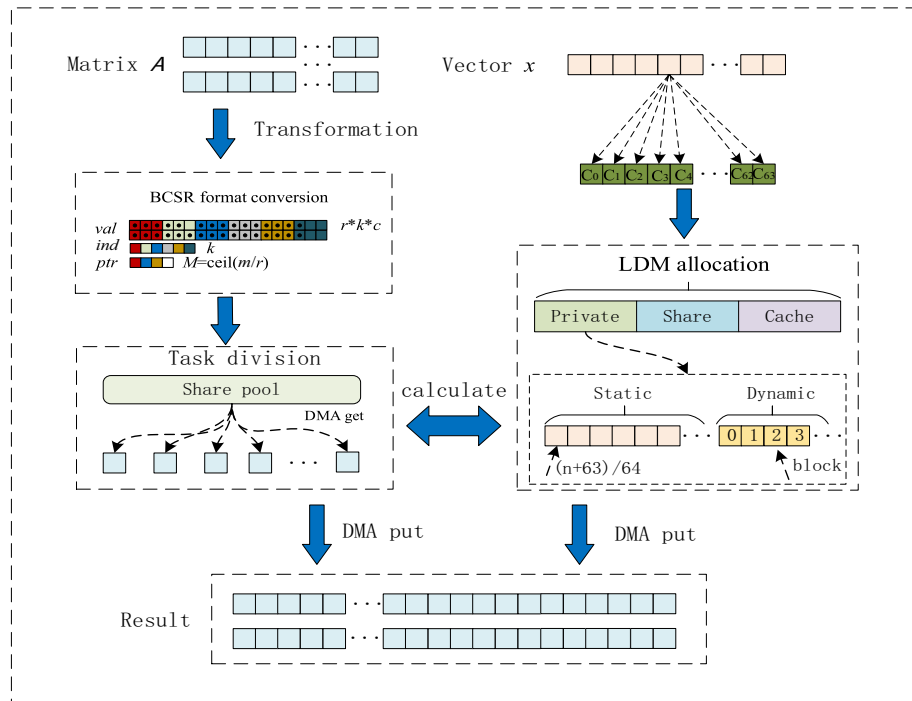


Figure 2. The SpMV algorithm implementation model.

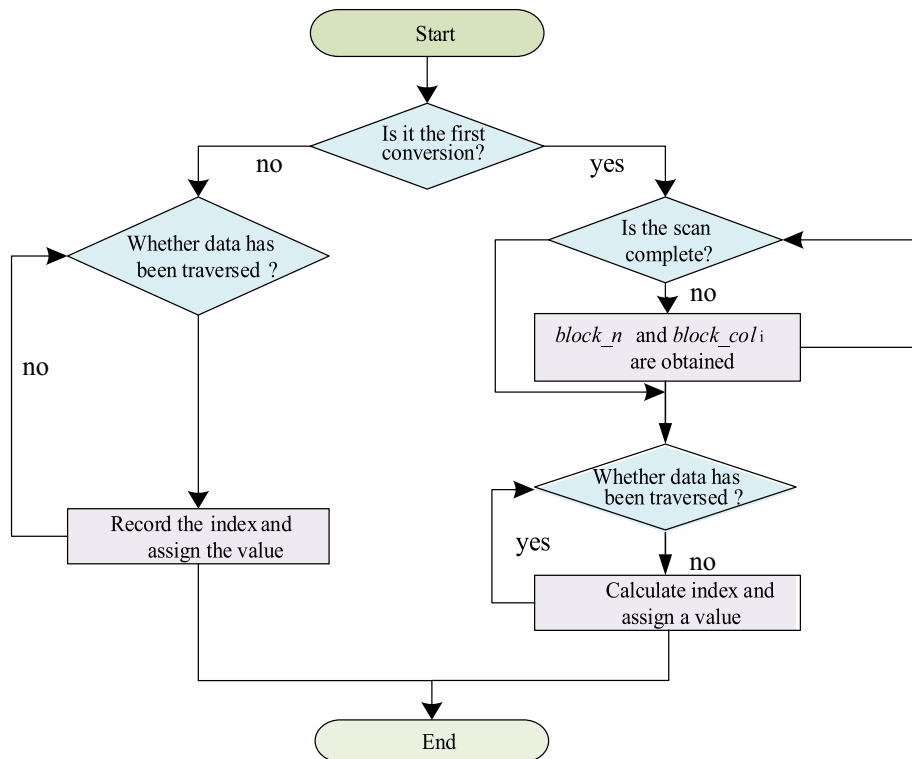


Figure 3. Transformation of CSR to BCSR storage format.

Finally, the map array and data array corresponded one to one, and the  $val_{index}$  of  $val$  where the corresponding data is stored is recorded. The specific transformation process is shown in Fig. 3.

*Data storage transformation strategy*

In practice, the sparse matrix is not computed by one operation but needs a continuous iterative process, and each iteration will update the value of the non-zero element and vector  $x$  in the matrix. To further improve the data transformation efficiency and reduce the format transformation time, we designed a memorized data storage transformation strategy. We added the  $data\_map$  array to the structure of BCSR storage to record the subscript of  $val$  array corresponding to each non-zero data after the first CSR transformation to BCSR. As shown in Fig. 4, we first determine whether it is the first conversion, if not, we can directly load the data of CSR into the corresponding  $block$  of BCSR when performing iterative calculation (storing the non-zero elements directly into the  $val$  array according to the subscripts recorded in  $data\_map$ ), eliminating the process of constantly scanning the matrix to calculate the index of non-zero elements, reducing the format conversion time and greatly improving the conversion efficiency.

**Data storage transformation strategy**

Analyzing the heterogeneous structure of SW26010P many-core processor, one core (CG) of SW26010P many-core processor includes one master core (MPE) and one slave core array, and one slave core array contains 64 slave cores (CPE), which are mainly responsible for the operation of data. To make full use of the computing resources of CPE and divide the computing tasks equally among the slave cores as much as possible, this paper adopts dynamic task division to reduce the waiting time of each CPE and realize load balancing. Firstly, combined with the special structure of SW26010P many-core processor and the storage characteristics of BCSR storage format, the sparse matrix of  $N*N$  is divided into  $m*n$  task  $ls\_block$  and numbered as follows:

$$Task = \{ls\_block_1, ls\_block_2, ls\_block_3, \dots, ls\_block_k\}$$

The 64 CPE independently obtained data for calculation at the same time, and the starting position of each CPE to obtain data was  $core_{index}$  which was calculated according to the slave core number ( $\_PEN$ ) and the number of row-slices in the task  $block$  ( $lsn$ ). The specific calculation method is shown in formula (5).

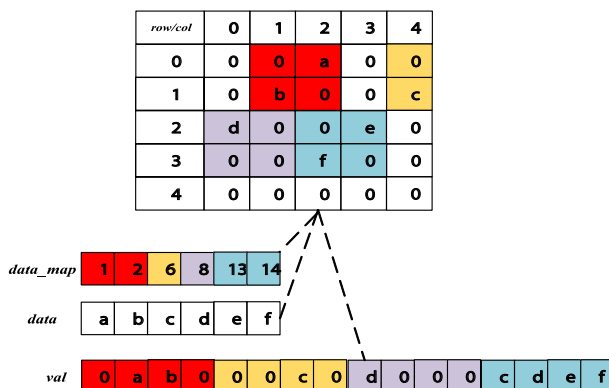
$$core_{index} = ptr[\_PEN * (ls_n + 63)/64] \tag{5}$$

The global index is used in task assignment, while the local index is used in CPE calculation. According to formulas (4) and (5), the conversion equation from local index to global index is shown in formula (6).

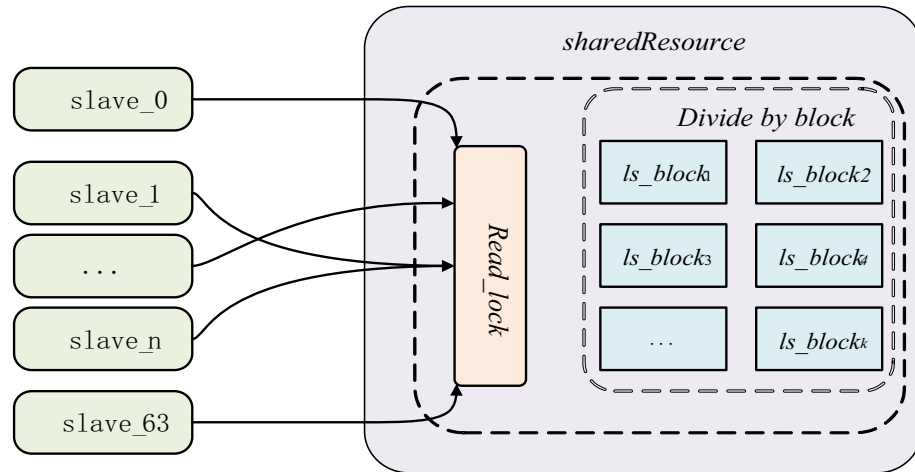
$$index_{global} = core_{index} + val_{index} \tag{6}$$

MPE schedules each CPE for calculation. When the CPE computes the allocated  $ls\_block_i$ , it obtains the next task  $block$  to be calculated from the task pool according to the task number. Because the memory of the LDM is limited, data needs to be obtained multiple times. Therefore, ensure that data must be obtained in data blocks each time.

After the task  $block$  data is obtained based on the task number, the  $block$  number  $task_i$  of the calculated task  $block$  must be accumulated. This paper uses parallel computing, to prevent the thread from reading the wrong task, and guarantee at the same time there can be only one value of the CPE change  $task_i$  before the implementation accumulative operation of variable  $task_i$  locked, and after completion of the procedure to unlock and synchronous operation, in order to prevent the resource conflicts and repeated calculation, the specific process as shown in Fig. 5.



**Figure 4.** The memory data storage transformation strategy.



**Figure 5.** Schematic diagram of dynamic partitioning.

**Vector X access optimization strategy**

It is well known that in SpMV algorithm, the vector  $x$  corresponding to the large-scale sparse matrix also occupies a huge space. Due to the limited LDM space of CPE, vector  $x$  is allocated to all CPEs. If the current coefficient vector  $x$  required from the kernel exists locally, it is directly obtained; If it does not exist, it will find the CPE number stored in the required vector  $x$ , and then obtain it from the corresponding CPE through RMA communication. The specific process is shown in Fig. 6.

In the calculation process of SpMV based on SW26010P many-core processor, the communication time occupies most of the overall time. To reduce the communication time and make the best use of LDM memory, the vector  $x$  access optimization strategy is improved. This algorithm is improved by referring to the idea of literature<sup>25</sup>. We change the static cache storage scheme, set multiple dynamic blocks, and adjust the size of a single dynamic block according to the processor’s performance. To make full use of the space and reduce communication times, we adopted the double cache strategy based on slave core architecture. We set two buffers and one static buffer  $buffer_{static}$  with a fixed size of  $x_{ssize}$ , and the proportion of  $x_{ssize}$  in the LDM space is 20%. For static storage of most vector  $x$  for reuse; The other uses the dynamic buffer  $buffer_{dynamic}$ , and the size is set to  $x_{dsize}$ , which consists of multiple dynamic blocks. We call the first address of each CPE static space storage vector  $x$  as  $x_{ss}[_PEN]$ , and the starting position of the static space vector  $x$  acquired by each CPE as  $x_s[_PEN]$ ,  $x_s[_PEN]$  is calculated as shown in formula (7).

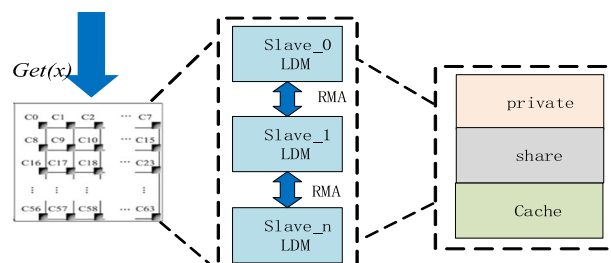
$$x_s[_PEN] = _PEN * (row + 63)/64 \tag{7}$$

In the calculation process, first query whether the required vector  $x$  is in  $buffer_{static}$ , if not, then go to  $buffer_{dynamic}$  to query, if still not, then need to get the size data of a dynamic block from other slave cores to get the vector  $x$  by RMA, and then update  $buffer_{dynamic}$ , the flow is shown in Fig. 7.

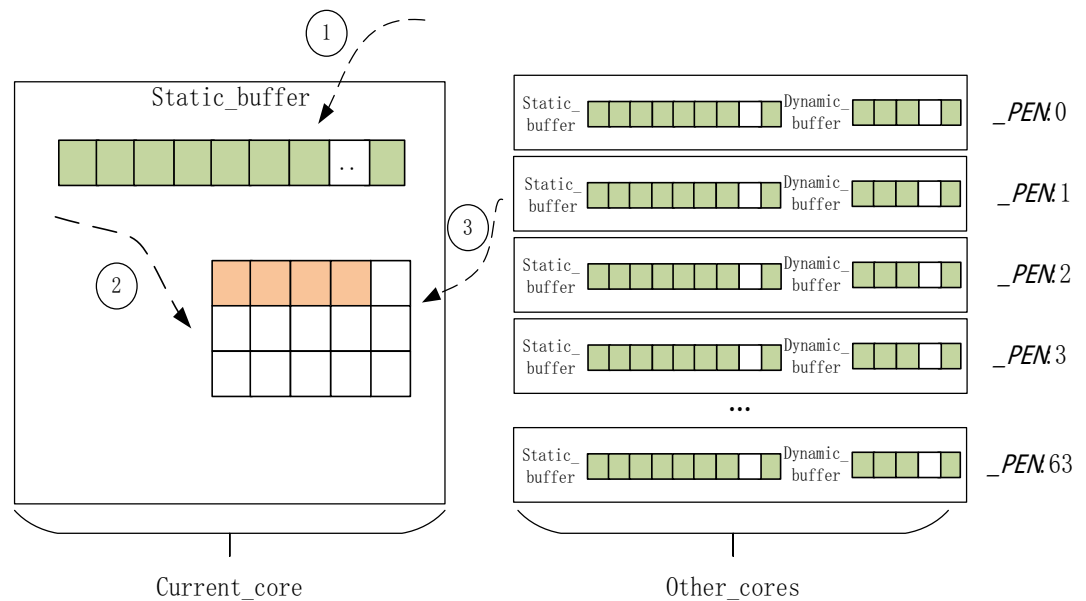
The RMA algorithm flow of CPE to realize dynamic vector  $x$  update among each other is as follows: Firstly, the target slave kernel number  $g\_id$  for data acquisition is calculated as shown in formula (8), where the  $index$  represents the position of the required vector  $x$ . Then, the starting position of dynamic vector  $x$  is obtained as  $x_d$ , and the calculation is shown in formula (9).

$$g\_id = index / (row + 63) / 64 \tag{8}$$

$$x_d = x_{ss}[g\_id] + index - x_s[g\_id] \tag{9}$$



**Figure 6.** Vector  $x$  acquisition process.



**Figure 7.** Vector  $x$  optimization algorithm.

Once the  $g_{id}$ ,  $xd$  is obtained, the data is stored in the  $buffer_{dynamic}$  of the current slave core through RMA and is used for calculation.

## Experiments

This section focuses on the design of experimental environment and test scheme in detail. To verify the effectiveness of the scheme, as many matrix properties as possible are covered. In this experiment, matrices of different shapes and sizes in the sparse Matrix library of Matrix Market<sup>33</sup> are selected as test sets to verify the optimization of this scheme, and the experimental data are analyzed in detail.

### Experimental Environment

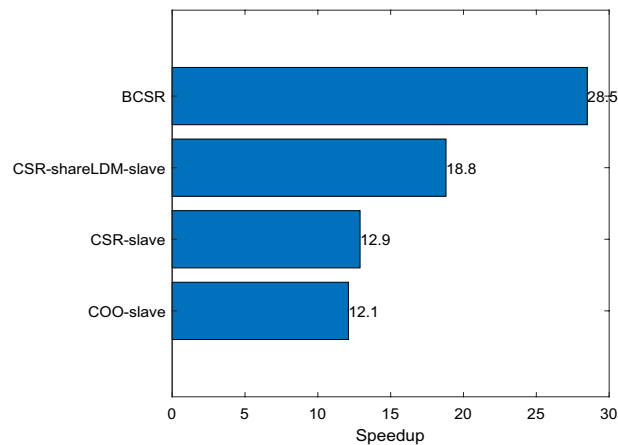
Matrix Market collects large sparse matrices obtained by discretizing many practical problems in science and engineering. To verify the effectiveness and universality of this scheme, a core group of the SW26010P many-core processor is used as the test platform for this experiment. The computational tasks are loaded asynchronously to the slave core for execution with the help of the high-performance threading library *Athread*. The experiment mainly analyzes and compares the two important indexes of the optimized calculation time and the master–slave speedup ratio. In addition, we consider that the *block\_size* of BCSR storage format greatly impacts the calculation performance, so this scheme carries out a lot of tests and analysis on the selection of *block\_size*. For the test matrix of this scheme, sparse matrices of different shapes in Matrix Market set with matrix size ranging from thousands to hundreds of thousands and the number of non-zero elements ranging from tens of thousands to millions are selected for a large number of tests.

### Experimental test and analysis

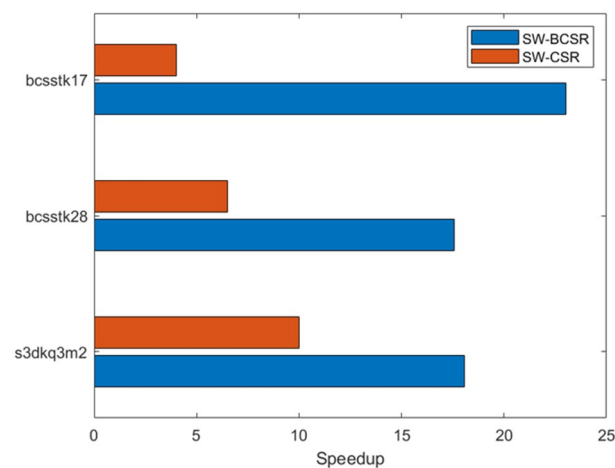
#### *BCSR storage advantages*

This paper employs the BCSR sparse matrix storage format, which helps to improve storage efficiency by dividing the matrix into smaller blocks, especially when the matrix has locally dense regions. Since BCSR divides the matrix into blocks of a fixed size, it can reduce the number of times vector  $x$  is fetched during the computation process, thereby enhancing memory computation efficiency. To demonstrate the effectiveness of the scheme, tests were conducted using the classic matrix in the pimplefoam solver from the OpenFoam software, with a matrix size of  $9216 \times 9216$  and 61,696 non-zero elements. Experiments were conducted using CSR format, COO format, BCSR format, and so on. In particular, an optimized CSR format data storage was designed, called the CSR-shareLDM-slave scheme, which used a shared LDM approach to store matrix data for testing, with results as shown in Fig. 8. The experimental results prove that the proposed BCSR format can achieve the best acceleration ratio, with an acceleration ratio of up to 28.5, effectively improving the speed of fluid dynamics simulations.

This paper implements the SpMV computation under the BCSR storage format on the domestically produced Sunway supercomputer, with a very noticeable acceleration effect. Currently, there is still relatively little work based on the new generation of domestic Sunway supercomputers. To prove the effectiveness of this method, a comparison was made with the CSR format SpMV on the Sunway TaihuLight supercomputer as presented in literature<sup>28</sup>. Three large-scale sparse matrices were selected for comparative experiments, and the results are shown in Fig. 9. The results show that among the three matrices, the BCSR storage format proposed in this



**Figure 8.** Acceleration ratios of different storage formats.



**Figure 9.** BCSR's acceleration effect on SW26010P processor.

paper has a significant advantage, with the maximum acceleration ratio reaching 23.03, which is a very good optimization effect.

#### *Master–slave speedup ratio test*

The article uses the self-developed SW26010P many-core processor from China as the experimental platform. This processor is composed of 6 core groups (CGs), each containing 1 computational control core (MPE) and 64 computational cores (CPE). The processor adopts a heterogeneous acceleration programming model, as shown in Fig. 10. The MPE is responsible for the distribution and management of data and accelerated tasks, using the relevant interfaces of *athread* to manage the acceleration thread tasks. Depending on the different functionalities of the interfaces, the accelerated thread tasks will be executed on one or more thread arrays. The CPE mainly realizes the acceleration of the core functional modules.

If 6 CGs are used for computation, since the inter-group process-level parallelism is carried out through MPI, the matrix data and vectors need to be evenly divided according to 6 processes, which may result in the data required for computation within process A being present within process B. Inter-process data transfer takes a long time and occupies a lot of memory, leading to low CPU utilization. For matrices at the scale of tens of millions, using one core group (CG) for computation can greatly reduce the time required for data transmission, thus optimizing performance.

Speedup is the ratio of the time consumed by the same computation task in many-core processors and single-core processors. It measures the performance and effect of parallelization of parallel programs. In the field of parallel computing, the speedup ratio is one of the important indexes to measure the performance of parallel processing. The computational performance of SpMV is affected by many factors, such as the shape of the sparse matrix, the degree of data sparsity, and the size of the matrix. To test the effectiveness and universality of the proposed scheme, sparse matrices are firstly simply classified according to size, shape and size. Then the calculation time and speedup ratio are tested. Detailed parameters of the sparse matrix selected in this scheme are shown in Table 1.

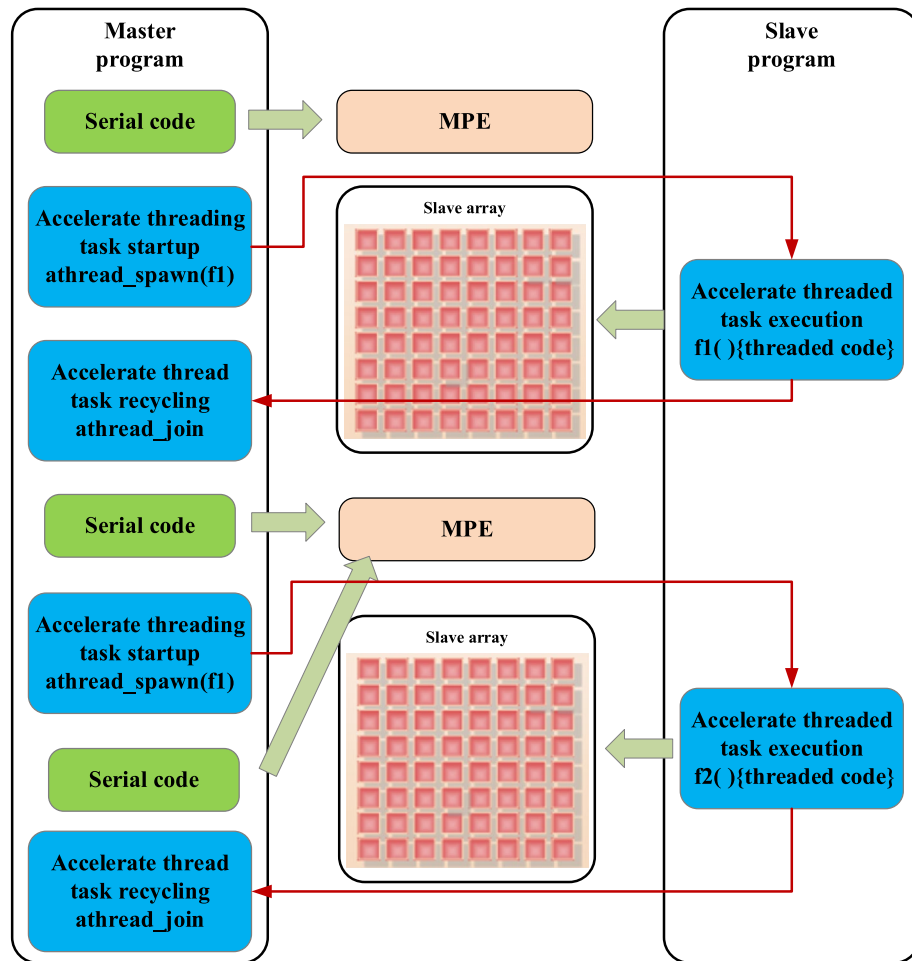


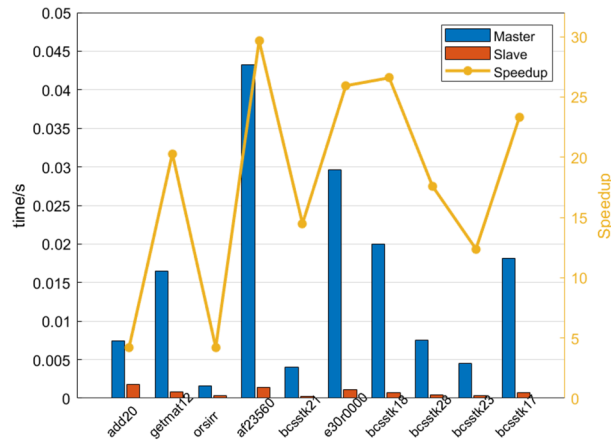
Figure 10. Sunway supercomputing’s heterogeneous acceleration programming model.

NO	Name	Size	Non-zero element
1	add20	2395 * 2395	17,319
2	getmat12	4929 * 4929	33,044
3	orsirr_1	1030 * 1030	6858
4	af23560	23,560 * 23,560	460,598
5	bcsstk21	3600 * 3600	15,100
6	e30r0000	9661 * 9661	306,356
7	bcsstk18	11,948 * 11,948	149,090
8	bcsstk28	4410 * 4410	111,717
9	bcsstk18	11,948 * 11,948	219,024
10	bcsstk17	10,974 * 10,974	219,812

Table 1. The first set of test matrices.

We use the control variable method, set the *block\_size* of all the test matrices to 5\*5, and then count the computation time of all the sparse matrices. To avoid errors, we calculated the average value for 20 times and obtained the acceleration ratio according to the calculated results. The statistical results are shown in Fig. 11.

Based on the results of the first set of tests, it is easy to conclude as follows: In the selected 10 sparse matrix samples, the computational performance of our scheme in this paper is significantly improved compared with the traditional computational methods. However, due to the difference in sparse matrix shape, matrix scale and element sparsity, the master–slave speedup ratio is also different. In this group of examples, the master–slave acceleration ratio is 28.81 times, and the average value is 18.83 times, which has a good acceleration effect and universality.



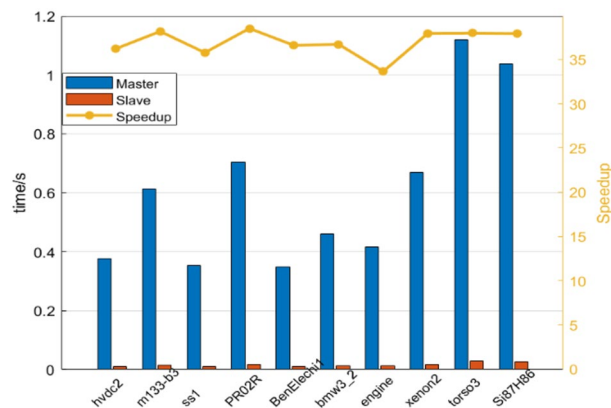
**Figure 11.** The results of the first set of test matrix.

To further test the effectiveness of this scheme, we reselected 10 larger sparse matrices from the Matrix Market as the second group of test samples, and the detailed parameters of the selected sparse matrices are shown in Table 2. The matrix size of the selected second group of test samples is larger and the number of non-zero elements can reach millions. Similarly, we used the control variable method, set the *block\_size* of all test matrices to 5\*5, and then counted the computation time of sparse matrices in the test set. To avoid errors, we also adopted the strategy of calculating the average value for 20 times and finally calculated the speedup ratio according to the calculated results. The statistical results are shown in Fig. 12.

Based on the results of the second set of tests, it is easy to conclude as follows: In the selected 10 sparse matrix samples, the computational performance of our scheme in this paper is significantly improved compared with

NO	Name	Size	Non-zero element
1	hvdc2	189,960 * 189,860	1,347,273
2	m133-b3	200,200 * 200,200	800,800
3	ss1	205,280 * 205,282	845,089
4	PR02R	161,070 * 161,070	8,185,136
5	BenElechi1	245,874 * 245,874	6,698,185
6	bmw3_2	227,362 * 227,362	5,757,996
7	engine	143,571 * 143,571	2,424,822
8	xenon2	259,156 * 259,156	3,866,688
9	torso3	259,156 * 259,156	4,429,042
10	Si87H86	240,369 * 240,369	5,451,000

**Table 2.** The second set of test matrices.



**Figure 12.** The results of the second set of test matrix.

the traditional computational methods, and the acceleration effect is more noticeable compared with the matrix selected by the first group. In the test samples of this group, the master–slave acceleration ratio of this scheme is up to 38.51 times, and the average value is up to 36.95 times, which has an excellent acceleration effect. It can be seen that the acceleration effect of this scheme is more evident for large matrix sizes and many non-zero elements.

#### The selection of *block\_size*

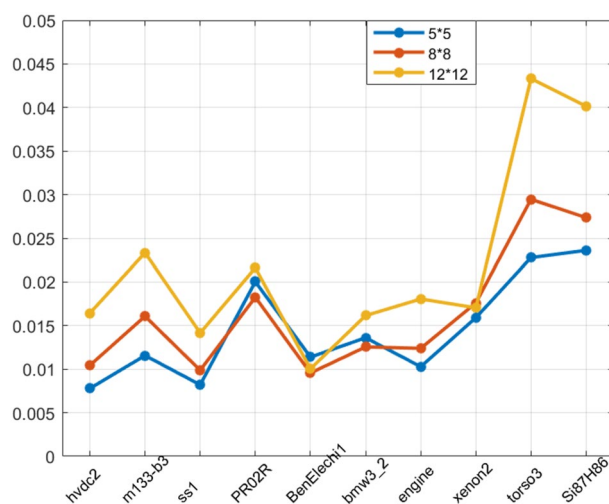
In order to achieve the best performance of sparse matrix–vector multiplication, the *block\_size* plays an essential role in BCSR storage format. The distribution of non-zero elements of sparse matrix affects the best selection of *block\_size*. To test the impact of *block\_size* on the optimization performance, we randomly selected a group of sparse matrices from the Matrix Market and tested them under three different *block\_size*. The specific parameters of the sparse matrix and *block\_size* are shown in Table 3.

Through the sparse matrix selected in this experiment, we successively select *block\_size1*, *block\_size2* and *block\_size3* for execution and then calculate the computation time of the sparse matrix under different *block\_sizes*. In addition, in order to avoid errors, we also take the method of calculating the average value for 20 times. The statistical results are shown in Fig. 13.

From the test results, it can be concluded that *block\_size* has a great impact on the calculation performance of SpMV. As shown in Fig. 13, when *block\_size* is 5\*5, the calculation performance is the best. This is because the density of the selected sparse matrix is low. If the *block\_size* is too large, a large number of invalid calculations will be carried out to reduce the calculation performance. Therefore, selecting the appropriate *block\_size* for SpMV optimization based on BCSR storage format is very important. The sampling rate is the proportion of the selected matrix samples to the total size of the matrix. Through experiments, we conclude that when the sampling rate reaches a certain value, the *block\_size* obtained in most cases will be stable, and the selection result of *block\_size* will not be changed even if the sampling rate continues to be increased. Therefore, the estimation method can be used in practical applications to obtain the optimal *block\_size* value.

No	Name	block_size1	block_size2	block_size3
1	hvd2	5*5	8*8	12*12
2	m133-b3	5*5	8*8	12*12
3	ss1	5*5	8*8	12*12
4	PR02R	5*5	8*8	12*12
5	BenElechi1	5*5	8*8	12*12
6	bmw3_2	5*5	8*8	12*12
7	engine	5*5	8*8	12*12
8	xenon2	5*5	8*8	12*12
9	torso3	5*5	8*8	12*12
10	Si87H86	5*5	8*8	12*12

**Table 3.** Sparse matrix *block\_size* selection table.



**Figure 13.** The *block\_size* selection test.

## Conclusions

In this paper, we propose a SpMV algorithm based on BCSR storage format for SW26010P many-core processor. In order to quickly transform the matrix in CSR storage format into BCSR storage, we design a memorized data storage transformation strategy, which dramatically reduces the transformation time and improves the efficiency of data transformation. In addition, by studying the master–slave architecture of SW26010P many-core processor, we adopted the dynamic task scheduling method to realize the load balance between each CPE during computation and reduce the waiting time between each CPE. To improve the utilization of LDM storage space and the hit ratio of vector  $x$ , we divide the LDM memory space reasonably, and improve the LDM dual cache optimization strategy to further improve the SpMV computing performance. To verify the effectiveness of our work, we selected a large number of representative sparse matrices of different shapes and sizes from the Matrix Market for testing. The results show that the method in this paper has an obvious acceleration effect on sparse matrices of various sizes and sizes, and the effect is more evident for large matrices, the master–slave acceleration ratio can be up to 38 times, which has a significant optimization effect. The optimization method used in this paper has implications for other complex applications of SW26010P many-core processor. More significantly, this is of great significance to the software ecological construction and performance optimization of self-developed supercomputers. Finally, there are still many shortcomings in our work, such as how to adaptively select the size of `block_size` and how to improve the hit rate of vector  $x$  further.

## Data availability

The data that support the results of this study are available from the Matrix Market, and can be accessed via the website <https://math.nist.gov/MatrixMarket/>. The majority of the data comes from “<https://math.nist.gov/MatrixMarket/extreme.html>” and “<https://math.nist.gov/MatrixMarket/searchtool.html>” of this website.

Received: 25 March 2024; Accepted: 11 July 2024

Published online: 17 July 2024

## References

- Zhou, Y. *et al.* Parallel compilation optimization method for sunway’s high performance multi-core processors. *Comput. Eng.* <https://doi.org/10.19678/j.issn.1000-3428.0062139> (2022).
- Lu, H. *et al.* Redesigning Vina@QNLN for ultra-large-scale molecular docking and screening on a sunway supercomputer. *Front. Chem.* **9**, 750325. <https://doi.org/10.3389/fchem.2021.750325> (2021).
- Liu, Y. *et al.* Closing the “quantum supremacy” gap: Achieving real-time simulation of a random quantum circuit using a new sunway supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* 1–12 (2021).
- Zheng, J. *et al.* A parallel approach for oil palm tree detection on a SW26010 many-core processor. In *IGARSS 2022–2022 IEEE International Geoscience and Remote Sensing Symposium, Kuala Lumpur* 1548–1551 (2022). <https://doi.org/10.1109/IGARSS46834.2022.9884785>
- Hao, X. *et al.* swMPAS-A: Scaling MPAS-A to 39 million heterogeneous cores on the new generation sunway supercomputer. *IEEE Trans. Parallel Distrib. Syst.* **34**, 141–153. <https://doi.org/10.1109/TPDS.2022.3215002> (2023).
- Xie, X. *et al.* SpaceA: Sparse matrix vector multiplication on processing-in-memory accelerator. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul* 570–583 (2021). <https://doi.org/10.1109/HPCA51647.2021.00055>
- Shan, M., Gurevin, D., Nye, J., Ding, C. & Khan, O. MergePath-SpMM: Parallel sparse matrix-matrix algorithm for graph neural network acceleration. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Raleigh* 145–156 (2023). [10.1109/ISPASS57527.2023.00023](https://doi.org/10.1109/ISPASS57527.2023.00023)
- Mohammed, T. *et al.* DIESEL: A novel deep learning-based tool for SpMV computations and solving sparse linear equation systems. *J. Supercomput.* **77**, 6313–6355. <https://doi.org/10.1007/s11227-020-03489-3> (2021).
- Zhao, Y., Zhou, W., Shen, X. & Yiu, G. Overhead-conscious format selection for SpMV-based applications. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Vancouver* 950–959 (2018). <https://doi.org/10.1109/IPDPS.2018.00104>
- Coronado-Barrientos, E., Indalecio, G. & García-Loureiro, A. AXC: A new format to perform the SpMV oriented to Intel Xeon Phi architecture in OpenCL. *Concurr. Comput. Pract. Exp.* **31**, e4864. <https://doi.org/10.1002/cpe.4864> (2019).
- Page, B. A. & Kogge, P. M. Scalability of hybrid SpMV on intel Xeon Phi knights landing. In *2019 International Conference on High Performance Computing & Simulation (HPCS), Dublin* 348–357 (2019). <https://doi.org/10.1109/HPCS48598.2019.9188154>.
- Elafrou, A., Goumas, G. I. & Koziris, N. BASMAT: Bottleneck-aware sparse matrix-vector multiplication auto-tuning on GPGPUs. In *Proceedings of PPOPP* 423–424 (2019).
- Benatia, A., Ji, W., Wang, Y. & Shi, F. Sparse matrix partitioning for optimizing SpMV on CPU-GPU heterogeneous platforms. *Int. J. High Perform. Comput. Appl.* **34**(1), 66–80. <https://doi.org/10.1177/1094342019886628> (2020).
- Wu, T., Wang, B., Shan, Y., Yan, F., Wang, Y. & Xu, N. Efficient PageRank and SpMV Computation on AMD GPUs. In *2010 39th International Conference on Parallel Processing, San Diego* 81–89 (2010). <https://doi.org/10.1109/ICPP.2010.17>.
- Bouvier, D., Cohen, B., Fry, W., Godey, S. & Mantor, M. Kabini: An AMD accelerated processing unit system on a chip. *IEEE Micro* **34**(2), 22–33 (2014).
- Nguyen, T., Williams, S., Siracusa, M., MacLean, C., Doerfler, D. & Wright, N. J. The performance and energy efficiency potential of FPGAs in scientific computing. In *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)* 8–19 (2020). <https://doi.org/10.1109/PMBS51919.2020.00007>.
- Jain, A. K., Lloyd, G. S. & Gokhale, M. Performance assessment of emerging memories through FPGA emulation. *IEEE Micro* **39**(1), 8–16 (2019).
- Zhang, J. *et al.* Extreme-scale phase field simulations of coarsening dynamics on the Sunway TaihuLight supercomputer. In *Proceedings of SC* 34–45 (2016).
- Alsalti Baldellou, A. Development of linear solvers for large-scale CFD simulations on hybrid supercomputers. Tesi doctoral, UPC, Departament de Màquines i Motors Tèrmics (2023). <https://doi.org/10.5821/dissertation-2117-405131>.
- Chu, G. *et al.* Efficient algorithm design of optimizing SpMV on GPU. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing* 115–128 (2023). <https://doi.org/10.1145/3588195.3593002>.
- Usman, S. *et al.* ZAKI: A smart method and tool for automatic performance optimization of parallel SpMV computations on distributed memory machines. *Mob. Netw. Appl.* **28**(2), 744–763. <https://doi.org/10.1007/s11036-019-01318-3> (2023).

22. Pan, J. *et al.* hsSpMV: A heterogeneous and SPM-aggregated SpMV for SW26010-Pro many-core processor. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Bangalore* 62–70 (2023). <https://doi.org/10.1109/CCGrid57682.2023.00016>.
23. Xiao, G., Chen, Y., Liu, C. & Zhou, X. ahSpMV: An autotuning hybrid computing scheme for SpMV on the Sunway Architecture. *IEEE Internet Things J.* **7**(3), 1736–1744. <https://doi.org/10.1109/JIOT.2019.2947257> (2020).
24. Kreuzer, M. *et al.* A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. *SIAM J. Sci. Comput.* **36**(5), C401–C423 (2014).
25. Liu, W., Vinter, B. CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication. In *Proceedings of the ACM International Conference on Supercomputing* 339–350 (2015).
26. Jiajia, Li. *et al.* Study on choosing the optimal storage format of spare matrix vector multiplication. *J. Comput. Res. Dev.* **51**(004), 882–894 (2014) (**in Chinese**).
27. Bian, H. *et al.* A simple and efficient storage format for SIMD-accelerated SpMV. *Cluster Comput.* <https://doi.org/10.1007/s10586-021-03340-1> (2021).
28. Fangfang, L. *et al.* General SpMV implementation in many-core domestic sunway 26010 processor. *J. Softw.* **29**(12), 3921–3932 (2018) (**in Chinese**).
29. Yiyuan, Li. *et al.* Performance optimization for spare matrix-vector multiplication on Sunway architecture. *J. Comput.* **52**(8), 1707–1721 (2015) (**in Chinese**).
30. Xiao, G., Li, K., Chen, Y., He, W., Zomaya, A. Y. & Li, T. CASpMV: A Customized and Accelerative SpMV framework for the Sunway TaihuLight. In *IEEE Transactions on Parallel and Distributed Systems* vol. 32, no. 1, 131–146 (2021). <https://doi.org/10.1109/TPDS.2019.2907537>.
31. Sun, Q., Zhang, C.-Y. Bandwidth reduced parallel SpMV on the SW26010 many-core platform. In *Proceedings of the 47th International Conference on Parallel Processing Eugene* 54 (2018).
32. Yca, B. *et al.* tpSpMV: A two-phase large-scale sparse matrix-vector multiplication kernel for manycore architectures. *Inf. Sci.* **523**, 279–295 (2020).
33. Boisvert, R. *et al.* *Matrix market* (National Institute of Standards and Technology (NIST), 2011). <http://math.nist.gov/MatrixMarket> (2004).

### Author contributions

M.M. wrote all manuscript text. X.H. proposed innovative ideas and assisted in the experimental design. D.J. and J.X. have jointly participated in proofreading the manuscript. All authors reviewed the manuscript.

### Funding

This work was supported by the National Key Research and Development Program of China (2021YFF0704000) and the National Key Research and Development Program of China (2020YFB0204804).

### Competing interests

The authors declare no competing interests.

### Additional information

**Correspondence** and requests for materials should be addressed to X.H.

**Reprints and permissions information** is available at [www.nature.com/reprints](http://www.nature.com/reprints).

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2024