Bandwidth Reduced Parallel SpMV on the SW26010 Many-Core Platform

Qiao Sun
Institute of Software,
Chinese Academy of Sciences
Haidian Qu, Beijing Shi, China
sunqiao@iscas.ac.cn

Changyou Zhang
Institute of Software,
Chinese Academy of Sciences
Haidian Qu, Beijing Shi, China
changyou@iscas.ac.cn

Changmao Wu*
Institute of Software,
Chinese Academy of Sciences
Haidian Qu, Beijing Shi, China
changmao@iscas.ac.cn

Jiajia Zhang
Institute of Software,
Chinese Academy of Sciences
Haidian Qu, Beijing Shi, China
jiajia@iscas.ac.cn

Leisheng Li
Institute of Software,
Chinese Academy of Sciences
Haidian Qu, Beijing Shi, China
leisheng@iscas.ac.cn

ABSTRACT

SpMV (Sparse Matrix-Vector multiplication), in its simplest form $y = Ax$, multiplies a sparse matrix with a dense vector and is a widely used computing primitive in the domain of HPC. On the newly SW26010 many-core platform, we propose a highly efficient CSR (Compressed Storage Row) based implementation of parallel SpMV, referred to as SW$_{CSR}$-SpMV in the sequel. SpMV in the CSR format can be trivially parallelized but its performance is majorly impeded by memory access efficiency, and therefore to leverage high-throughput memory access mechanism while avoiding redundant bandwidth usage becomes the major goal of designing high performance SpMV on the target platform. The original problem is sequentially partitioned into row-slices, each of which can reside in the fast scratchpad memory, so that the loaded x'es can be reused; meanwhile, a dynamic look-ahead scheme is applied to avoid redundant memory access; we split the many-core mesh into smaller communication scope to facilitate the sharing of the common data across the working threads via the high speed on-mesh data bus. Beyond the above, to leverage massive parallelism balanced workload is ensured by both static and dynamic means. Performance evaluation is done on a benchmark of 36 frequently used sparse matrices in the fields of graph computing, data mining, computational fluid dynamics, etc.. While the performance upper-bound is defined by the ratio between the minimal data access volume required against the practically optimal bandwidth, ignoring the computing overhead, SW$_{CSR}$-SpMV can achieve an efficiency of nearly 87%, maintaining over 75% for 1/3 of the testing matrices. SW$_{CSR}$-SpMV is further applied in a PETSc based application, a 1.75x-2.6x speedup is sustained in a multi-process environment on the Sunway TaiHuLight supercomputer.

CCS CONCEPTS

• **Mathematics of computing** → **Mathematical software performance**; • **Computing methodologies** → **Vector / streaming algorithms**; **Massively parallel algorithms**;

KEYWORDS

Sparse Matrices, Sparse Matrix-Vector Multiplication, CSR, Parallel SpMV, SW26010, Sunway TiHuLight, Many-core.

ACM Reference Format:
Qiao Sun, Changyou Zhang, Changmao Wu, Jiajia Zhang, and Leisheng Li. 2018. Bandwidth Reduced Parallel SpMV on the SW26010 Many-Core Platform. In *ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3225058.3225074

1 INTRODUCTION

SpMV (Sparse Matrix-Vector multiplication) is an important computing primitive in scientific applications and usually dominates the execution time [27]. Besides the typical usage in various iterative solvers, the SpMV operation can be found in domains such as graph analytics [13] and machine learning [17]. Due to the fundamental role SpMV plays in many applications, during the years the research on accelerating SpMV has constantly drawn great attention on various types of high performance architectures such as multi-core CPU [32, 35, 37], GPGPU [19, 40] and MIC [28, 39]. With the emergence of the SW26010 many-core CPU [15], it is of great value to design and implement a high performance SpMV primitive on this novel HPC platform.

The Sunway TaiHuLight supercomputer, the current champion in the Top-500 supercomputer list [3] in 2017, can theoretically deliver more than 100 PFlops for double-precision floating-point numbers and is designed for large scale scientific and engineering applications. At its heart is the SW26010 many-core CPU. As shown in Fig. 1, each SW26010 CPU consists of 4 CGs (Core Group), which are interconnected via an on-chip network. Each CG contains an MPE (Management Processing Element) and 64 CPEs (Computing Processing Element). The main process runs on the MPE which is

*Corresponding author: Changmao Wu (changmao@iscas.ac.cn).

ICPP 2018, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6510-9/18/08…$15.00
https://doi.org/10.1145/3225058.3225074

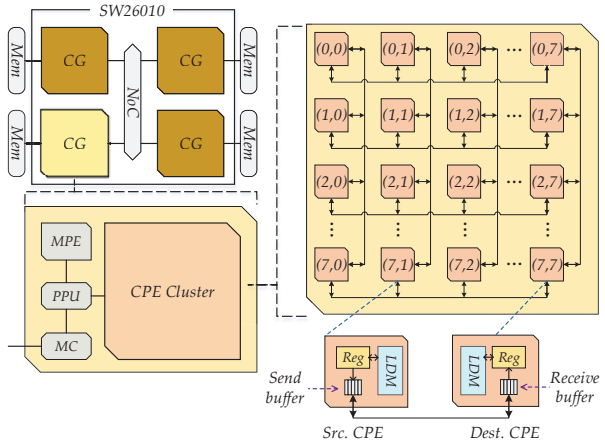


Figure 1: The architecture of a SW26010 CPU.

essentially a general-purpose processor, wherein the computation intensive workloads can be designated to the CPEs, each of which is a lightweight core and executes a single thread. A set of fast scratch-pad memory, named as LDM (Local Data Memory), is embedded in each CPE core and provides users with explicit control over data locality. The 64 CPE cores are arranged in an $8*8$ mesh and connected via row/column and signal/data buses. The MPE together with the all the CPEs in one CG share the main memory but the whole CPE set is able to sustain a much higher aggregate bandwidth than MPE does when loading/storing continuous chunks of data by the DMA (Direct Memory Access) mechanism.

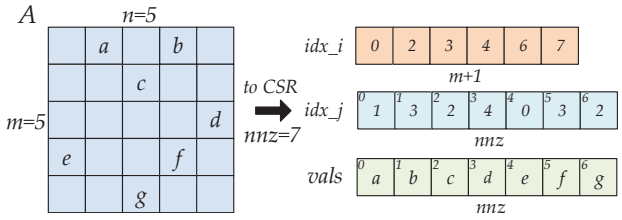


Figure 2: The standard CSR format.

CSR (Compressed Sparse Row) [21] is one of the widely used formats for storing sparse matrices. Due to the high compression ratio, it has been supported by many well-known scientific computing frameworks such as PETSc [6], Hypre [12] and Trilinos [16] etc.. As shown in Fig. 2, an $m*n$ sparse matrix "A" containing nnz (number of non-zeros) non-zero values is compressed into three arrays: all the numerical values of the non-zero elements are stored in the array "$vals$" with their column indices put in the array idx_j in the same order; the i-th element in the array "idx_i" is the offset of the first non-zero in the i-th row in both $vals$ and idx_j. Without loss of generality, in this paper we assume the non-zeros in each row are sorted in ascending order with respect to their column indices, otherwise a row-wise sorting operation can help maintain the assumption. Though, CSR-based SpMV, in its simplest form $\mathbf{y} = \mathbf{Ax}$, exposes massive parallelism and the direct row-wise parallelization to it is straightforward even on the novel SW26010 CPU, the scalability of the trivial parallelization is hardly optimal on parallel platforms, because the performance of SpMV is majorly impeded by irregular memory access patterns. Moreover, as compared to main stream many-core platforms such as GPGPU and MIC, there are several factors that make the implementation of a high performance SpMV operation on the SW26010 CPU more challenging. First, the irregularity of memory access patterns of SpMV may result in a much poorer data access efficiency when programmers are supposed to take full responsibility for both memory access (mainly by DMA) and data locality in LDM. Second, the bandwidth of DMA conflicts with the redundant loading of vector $\mathbf{x}$, i.e. acceptable bandwidth of DMA can be achieved only when accessing continuous data in large volume but a large part of the loaded data may be useless in the case of SpMV. Thirdly unlike the platforms with large shared cache where the common data can be reused by peer cores, communication scheme among the CPEs must be carefully designed to avoid repetitive access to the same data portion [14] by different CPEs. Last but not least, the SpMV operation on highly unstructured matrices will possibly cause severe load imbalance to which many-core platforms are more susceptible.

In this paper, we focus on the SpMV operation on one CG of SW26010 and propose SW$_{CSR}$-SpMV. Within a lightweight preprocess phase, we simply partition the source matrix sequentially into row-slices, so that each row-slice can reside in LDM and is handled individually by a certain CPE. In each LDM work-frame of a row-slice, we use DMA mechanism to access segments of vector $\mathbf{x}$, and meanwhile a dynamic look-ahead scheme is designed to avoid loading useless elements. To further reduce the memory footprint, worker groups of CPEs can be optionally set up, wherein the segments can be shared by all the peers via the on-mesh buses. As for load imbalance, we both make sure all the row-slices have roughly equal number of non-zeros by perfect partitioning and apply an atomic-operation based work-sharing pool at runtime. Lastly, a parameter auto-tuning framework is proposed to make SW$_{CSR}$-SpMV more adaptive to different matrices. Experimental results on 36 frequently used matrices in the fields of graph computing, data mining, computational fluid dynamics, etc. from the Tim Davis sparse matrix repository [11] suggest that all the approaches mentioned above are significantly effective. Based on the analysis on the theoretical performance upper-bound, SW$_{CSR}$-SpMV can achieve nearly 87% of the ideal performance and maintains an efficiency over 75% for 1/3 of the testing matrices. We further apply SW$_{CSR}$-SpMV to a PESTc based application which simulates the earth magnetic field, and the result shows that this application can be accelerated by 1.75x to 2.69x when invoking SW$_{CSR}$-SpMV in a multi-process environment on the Sunway TaihuLight supercomputer.

This paper is organized as follows: In Section 2 we will highlight the key features of SW26010 which determines the design and implementation of SW$_{CSR}$-SpMV. SW$_{CSR}$-SpMV will be detailed in Section 3. In Section 4 we are going to demonstrate the experimental results. The related works will be discussed in Section 5 before we draw our conclusion in Section 6.

2 FEATURES OF SW26010

There are distinctive features that have great impact on SW$_{CSR}$-SpMV. From the angle of hardware, one CG has a high computing power of 742.5 GFlops while the theoretical aggregate bandwidth produced by the 64 CPEs is only 34 GB/s. Given the large computing-bandwidth discrepancy, observable speedup can hardly be gained for bound-width bounded problems such as SpMV [33] by, vectorizing the computing kernel. As for a single CPE, the 64KB programmer-controllable LDM plays an important role in performance. Data structures should be designed to be fit within LDM and programmers are supposed to manage data access behaviors so that the cached data can be reused as much as possible. Despite intolerably high latency, data of basic or user-defined types can be directly accessed by CPE load/store instructions but the peak bandwidth of the CPE mesh can only be achieved by issuing DMA requests when accessing physically contiguous data sets. Thus, to achieve optimal bandwidth for SW$_{CSR}$-SpMV it is necessary to access all the input/output operands by DMA, although generally the useful elements of vector **x** are irregularly scattered for a row-slice. Thanks to the data/signal buses, CPEs can exchange their local data in LDM by register communication and the fine-grained synchronization barrier within each subset of CPEs in a row/column can be facilitated.

The "athread" library provides APIs for both spawning/joining the CPE threads and the DMA mechanism. However, in order to maintain balanced workload among CPEs it is required to extend the basic fork-and-join paradigm on the CG. The register communication and synchronization within a CPE subset can only be manipulated by lower level SIMD intrinsics, which also poses great challenge for implementing parallelism.

3 SW$_{CSR}$-SPMV

3.1 The Pre-Process of SW$_{CSR}$-SpMV

Table 1: The work frame of a row-slice residing in the LDM.

Name	Type (Dimension)	Content
slice_nrows	int (1)	The # of rows in the slice.
slice_nnz	int (1)	The # of non-zeros in the slice.
slice_idx_j	int (*local_nnz*)	The col. idx of the non-zeros.
slice_nz	double (*local_nnz*)	The non-zero values.
slice_idx_i	int (*local_nrows*+1)	The row offsets.
row_process	int (*local_nrows*)	The current process of each row.
x_buffer	double (*size_x*)	The buffer storing *x*'es.
y_buffer	double (*local_nrows*)	The buffer storing *y*'es.

To leverage the natural parallelism that SpMV has, we apply the row-slice based approach [5, 8, 26, 35], and it also helps data-locality for the rows within a slice may share the accessed *x*'es. As summarized in Table 1, a row-slice in SW$_{CSR}$-SpMV contains several contiguous sparse rows, which corresponds to a consecutive segment of the resultant vector **y**. Thus, both the load of the row-slices and the store of *y* segments can be done by DMA efficiently. We bound the volume of a row slice by "*slice_nnz*" and "*slice_nrow*", where *slice_nnz* is the max number of non-zero values and *slice_nrow* is the upper-bound of the row count. Both of the parameters greatly impact the overall performance of SW$_{CSR}$-SpMV, and need extensive tuning in actual use.

Given the limitation of a row-slice, it is straightforward to perform a perfect partition to the source matrix [30] with regard to the non-zero count by scanning the source matrix once. In this way, the computing workload among each slice is balanced. The row slicing is done in a lightweight pre-process procedure, where the MPE produces all row-slices and in the meantime the range of the relevant *x*'es to each row-slice is narrowed down to avoid redundancy. During the pre-process, we also need to isolate the rows containing too many non-zeros to reside in a single LDM. While they can be handled by a specific MPE or CPE procedure in parallel with other row slices, SW$_{CSR}$-SpMV runs faster when the uploaded *x*'es are better reused in each row slice.

A row-slice forms a work-frame for an assigned CPE, and after the entire row-slice is uploaded to the LDM, the CPE computes by DMA based segment-wise uploading of the vector **x**. The maximal size of the *x* segment in each DMA request (denoted by "*size_x*") is an another performance-critical parameter to be tuned in actual use. The rows in a row-slice can share each of the uploaded *x* segment with the help of an array recording the running process of each row.

3.2 Avoiding redundant load of *x*'es

By "sparsity", the non-zeros are logically scattered in a sparse matrix, which leads to the irregular access to the source vector **x**. At the row-slice level, there are often sub-intervals of useless *x*'es within the relevant *x* range, as shown in Fig. 3. Practical examples of this case are the off-diagonal matrices in PETSc storing the non-zeros from both sides of the diagonal. DMA, however, delivers a much better bandwidth when accessing contiguous data segments, which suggests that plenty of useless *x*'es will be fetched by DMA if CPEs only perform plain segment-wise upload. While the length of each **x** segments to load is set to *size_x*, the number of useless *x*'es can be reduced by speculating a better point to start.

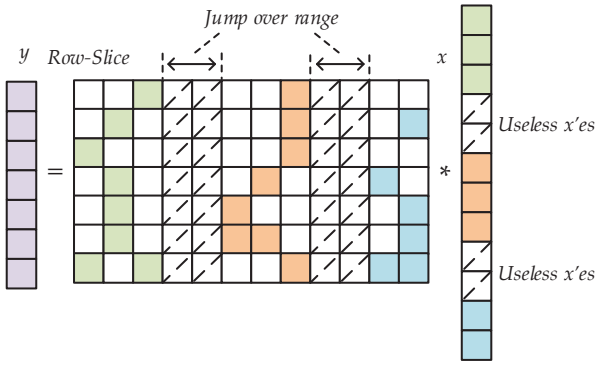
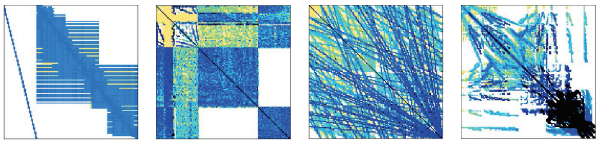


Figure 3: The conceptual illustration of useless segments of *x*'es.

During the computation across the rows in the slice, the maximal column index j_l of the used non-zeros can be determined; so the lower-bound for the following useless interval is $j_l + 1$. When computation concerning the current **x** segment ends, each element *i* in the array *row_process* points to the next-to-use non-zero in row *i* in the slice, and the minimal column index j_u of the next-to-use

x'es can be reduced. Thus the upper-bound for the following useless interval of vector $\mathbf{x}$ is $j_u - 1$. The computation can safely resume without loading the x'es indexed in the interval $[j_l + 1, j_u - 1]$, and may lead to a much reduced memory access. This on-chip speculation is done in parallel with the computation of each matrix row, and the overhead is negligible as compared with the DMA access to the operands.

3.3 SW$_{CSR}$-SpMV with Worker Group



(a) av41092. (b) poisson3Db. (c) sparsine. (d) offshore.

Figure 4: The sky-plots of the irregular sparse matrices from [11].

When matrices are highly irregular, as exemplified in Fig. 4, different row-slices may cover a wide range of common x'es and the data reuse ratio of the x segments (of length $size\_x$) may be increased if they can be shared in the runtime. To that end, on one hand the CPE mesh need splitting into smaller worker groups in the way that there is a direct bus connection among all the peers. On the other hand, in the pre-process phase, adjacent row-slices are logically grouped into sets in a way that the number of row-slices in each set equals to that of the workers in a worker group to ensure a full occupancy. The union of all the indices of the required x segments by a row-slice set can also be assembled in the pre-process phase, so that the x segments can be collectively accessed by the peers in a group.

Due to the directly supported synchronizing barrier among the CPEs in a row, we are able to split the 8 CPEs into 1,2,4 groups with 8,4,2 CPEs in each group respectively. An example of a group of 4 CPEs is shown in Fig. 5. At each time, 4 row-slices are handled by the group, and the i-th CPE is responsible for loading the $i + kn$-th x segment by DMA in the k-th iteration. To ensure a higher bandwidth occupancy, all the CPEs take part in the uploading of the x segments in the "DMA phase". When the entire group of the CPEs is fed, in the "Bcast phase" the first CPE broadcasts "seg-0" to the rest co-workers and let them resume the computation. Then the second CPE becomes the broadcaster and the phase continues in this way until all of the x segments are consumed. Synchronization barriers local to the group are put to coordinate the data exchange among the CPEs. When the number of a row-slices is smaller than the CPE count in a worker group, some phantom work-frames with the same behavior of loading x'es but without computation is patched to ensure a right synchronization.

We encapsulate the work group logic into an independent utility which can be used in parallel algorithms where multiple CPEs are accessing the same data set. On the downside, however, in this approach local barrier undermines the performance severely: the CPE with the slowest DMA operation determines the duration of each iteration, leading to a sub-optimal collective bandwidth usage.

3.4 Dynamic Load Balance

Statically, we make sure each row-slice in SW$_{CSR}$-SpMV has roughly equal number of non-zeros and thus the whole number of floating point operations as well as the size of each work frame are evenly divided. However, the actual workload of each row-slice also lies in its access patterns of the vector $\mathbf{x}$, i.e. the total number of x'es loaded and the reuse ratio, which can only be known during runtime. Similarly on the high-end GPGPU platforms where atomic operations are available, the workload of SW$_{CSR}$-SpMV can be balanced by the atomic operation based work-sharing technique [10]. When the row slicing is done, all the slices are sequentially numbered and queued. Each CPE can fetch one row-slice by issuing the atomic "fetch-and-add" instruction to a global row-slice counter. In this way, each row-slice is handled only once and each CPE can obtain a new task in time with merely negligible idleness. This approach is also applied to the worker group variation of SW$_{CSR}$-SpMV: the leader of a group is responsible for fetching new workload and wakes up the co-workers by broadcasting wake-up signals within the group.

3.5 Parameter Auto-Tuning

In order to make SW$_{CSR}$-SpMV more adaptive to various sparse matrices, an auto-tuning framework becomes a necessity. For the performance-critical parameters, $slice\_nrow$, $slice\_nz$ and $size\_x$, a framework traversing the entire parameter space (at a given stride) are supposed to record the optimal parameter combination when the highest performance is achieved. This framework is able to resume when encountering illegal parameter combination by judging if a row-slice can reside in the LDM. A similar approach is designed for the worker group version of SW$_{CSR}$-SpMV, which is omitted here for brevity.

4 PERFORMANCE EVALUATION

The experiments are conducted on one CG of the SW26010 CPU. Worth mentioning is that, the memory controller in one SW26010 chip is able to isolate the 4 CGs, so that each CG can occupy the whole bandwidth of the near-sided memory. The MPE runs at 1.45 GHz and has a theoretical memory bandwidth about 5 GB/s. Each CPE also runs at 1.45 GHz but the theoretical aggregate DMA bandwidth of the CPE mesh is 34 GB/s. The compiler "sw5cc", is customized to compile MPE or CPE codes. In the following experiments, all of the testing programs are compiled with the `-O3` optimization.

4.1 Theoretical Performance Upper-Bound

The ratio α of computation against memory access of the CSR-based SpMV on a certain sparse matrix determines the performance upper-bound, which is denoted by P_{ideal} in the sequel. For a given matrix, the total number of floating point operations is $c = 2 * nnz$ and the minimal number of bytes accessed is $v_{min} = (m + 1 + nnz) * sizeof(idx\_type) + (m + n + nnz) * sizeof(val\_type)$, assuming the platform has a set of shared cache which is large enough to accommodate the entire source and resultant vectors: $\mathbf{x}$ and $\mathbf{y}$. Combining $\alpha = c/v_{min}$ and the bandwidth B of the platform, P_{ideal} can be approximated by $\alpha * B$. Given the sparse matrix, α can easily be computed and the next subsection will detail on

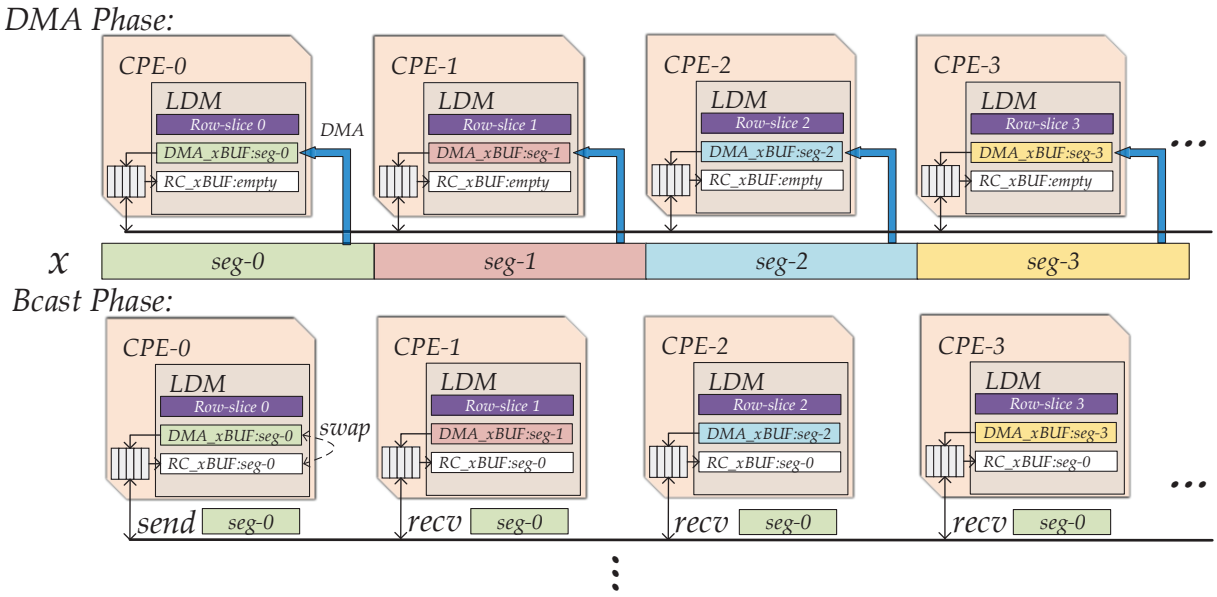


Figure 5: The key phases in the worker group version of SW$_{CSR}$-SpMV.

the practical bandwidth of the CPE mesh of SW26010. But worth mentioning is that the above assumption on the platform is over-simplified by the large and shared cache and we totally ignore the computation cost (about 5 ∼ 10% of the whole execution time in practice), which may result in an over-estimated performance upper-bound.

4.2 Bandwidth Study

Table 2: The actual DMA bandwidth in different use case scenarios, measured in GB/s. A: 256-Byte aligned; B: chunks of 256 Bytes.

Operation	A & B	!A & B	A & !B	!A & !B
Only Read	26.65	26.62	26.07	25.12
Only Write	24.41	22.29	18.23	17.51
Read & Write	22.45	21.50	20.02	19.82

As pointed out by Dang et.al [22], the practically measured bandwidth by benchmarks such as STREAM [29] is an important reference when considering the bandwidth efficiency of SpMV operations. Likewise, we conduct the test on the practical aggregate bandwidth B of the CPE mesh on the basis of a lightweight parallel benchmark program which loads/stores data chunks by DMA and reports the sustained bandwidth. The experimental results are listed in Table 2.

From Table 2 we notice that if all the 64 CPEs only perform DMA load or store operations a higher bandwidth can be sustained. The bandwidth can further increase when the size of the data chunks are multiples of 256 bytes and the heading address is 256 byte aligned. The peak aggregate bandwidth for read can achieve 26.65 GB/s which is approximately 80% of the reported theoretical bandwidth (34 GB/s). However, when the test case is mixed with both load and store operations, the sustained bandwidth is only 22.45 GB/s. Moreover, if the data chunks are not properly aligned or in irregular sizes, a further 4.3%-10.8% down will occur. Our results in Table 2 are in compliance with the reported in [38]. Likewise, we also take 22 GB/s as the practical bandwidth upper-bound B and use it to compute P_{ideal} for the testing matrices, which is reasonable considering the irregular memory access patterns in SpMV.

4.3 Performance Test on SW$_{CSR}$-SpMV

4.3.1 Effect of Optimizations. We select 36 matrices which are frequently used in previous works such as [4, 7, 33, 36, 37, 39] from the Tim Davis Sparse Matrix Repository [11], the meta information of which are listed in Table 3 and Table 4. During the test on each matrix, the optimization parameters are manually fixed to profile the gains of each optimizations, which are also recorded in Table 3 and Table 4. The dynamic look ahead scheme greatly reduces the run-time of nearly all the matrices, which suggests that a large number of extra x'es is avoided. A 0.91x ("mc2depi") to 34.07x ("Harmle3") speedup is sustained by this approach. On top of that, the dynamic work-sharing mechanism can help to boost the performance of some highly irregular matrices such as "mip1", "cont-300" and "Hamrle3" etc.. Notably, the time for the pre-process of SW$_{CSR}$-SpMV is no more than 4.2x ("dense2000") that of the SW$_{CSR}$-SpMV computing time and averagely only 0.96x.

4.3.2 Performance Overview. In Fig. 6, the final performance of all the testing matrices are summarized as well as the naive MPE SpMV reference. The matrices are listed in the same order with that in both Table 3 and 4, for ease of reference. In spite of the significant speedup against the MPE implementation, we also measure the efficiency of SW$_{CSR}$-SpMV, which is defined to be the actual performance against P_{ideal}. Thanks to exhaustive parameter auto-tuning by the framework, as we can see, the efficiency can

Table 3: The meta information of the testing square matrices and the effect of optimizations. The performance is measured by execution time in ms.

Name	M/N	NNZ	Sky-plot	Basic Impl.	Look-ahead & Static	Look-ahead & Dynamic	Pre-Process
cant	62451	4007383		2.55	2.55 (1.00x)	2.58 (0.99x)	1.95
pwtk	217918	11524432		20.89	8.83 (2.36x)	7.99 (1.10x)	5.68
af-shell9	504855	17588875		15.16	13.30 (1.14x)	11.99 (1.11x)	10.24
mip1	66463	10352819		43.23	10.46 (4.13x)	7.46 (1.40x)	9.60
pdb1HYS	36417	4344765		5.86	3.42 (1.71x)	3.19 (1.07x)	1.85
RS-b300-c2	28338	2943887		6.27	2.51 (2.49x)	2.17 (1.15x)	1.38
shipsec1	140874	7813404		8.65	7.00 (1.27x)	5.8 (1.19x)	3.94
consph	83334	6010480		10.83	5.11 (2.12x)	4.6 (1.10x)	1.91
dense2000	2000	4000000		3.24	3.25 (0.99x)	3.11 (1.05x)	12.81
thermal2	147900	3489300		19.80	3.05 (6.50x)	2.77 (1.10x)	2.67
water-tank	60740	2035281		1.98	1.93 (1.03x)	1.63 (1.18x)	1.37
nd6k	18000	6897316		11.60	5.65 (2.05x)	5.57 (1.02x)	2.23
pli	60740	1350309		1.22	1.34 (0.91x)	1.13 (1.19x)	0.74
RS-b162-c3	15374	610299		1.09	0.81 (1.35x)	0.61 (1.31x)	0.50
Si41Ge41H72	185639	15011265		61.35	16.57 (3.70x)	15.40 (1.08x)	6.74
torso2	115967	1033473		6.41	1.16 (5.51x)	1.11 (1.05x)	1.68
Ga41As41H72	268096	18488476		97.92	20.84 (4.70x)	20.06 (1.04x)	8.41
thermal1	17880	430740		0.74	0.54 (1.36x)	0.47 (1.15x)	0.51
mc2depi	525825	2100225		2.27	2.49 (0.91x)	2.44 (1.02x)	6.09
cont-300	180895	988195		7.60	1.48 (5.14x)	1.21 (1.22x)	2.31
econ-fwd500	206500	1273389		1.55	1.65 (0.94x)	1.57 (1.05x)	2.68
finan512	74752	596992		2.93	0.77 (3.81x)	0.75 (1.03x)	1.12
in-2004	1382908	16917053		810.66	34.56 (23.45x)	21.67 (1.59x)	19.38
epb2	25228	175027		0.22	0.23 (0.96x)	0.22 (0.93x)	0.40
Hamrle3	144736	5514242		465.66	13.67 (34.07x)	7.83 (1.74x)	17.81
sme3Da	12504	874887		1.39	1.42 (2.86x)	1.40 (1.02x)	0.53
bayer01	57735	277774		1.45	0.51 (2.81x)	0.45 (1.12x)	0.84

Table 4: The meta information of the testing square matrices and the effect of optimizations. The performance is measured by execution time in ms.

Name	M/N	NNZ	Sky-plot	Basic Impl.	Look-ahead & Static	Look-ahead & Dynamic.	Pre-Process
av41092	41092	1683902		8.63	3.36 (2.56x)	2.78 (1.21x)	1.20
poisson3Da	13514	352762		0.76	0.76 (1.00x)	0.73 (1.03)	0.38
TF18	95368	1597545		8.30	5.01 (1.65x)	3.63 (1.37x)	1.67
sme3db	29067	2081063		5.31	5.07 (1.05x)	4.94 (1.02x)	1.05
sme3Dc	42930	3148656		11.62	10.51 (1.11x)	10.02 (1.05x)	1.51
sparsine	50000	1548988		6.32	5.25 (1.20x)	5.14 (1.02x)	1.16
helm3d01	32226	428444		1.53	1.47 (1.04x)	1.45 (1.01x)	0.62
offshore	259789	4242673		63.46	23.68 (2.67x)	16.88 (1.40x)	4.00
poisson3Db	85623	2374949		14.78	12.74 (1.16x)	11.70 (1.09x)	1.74

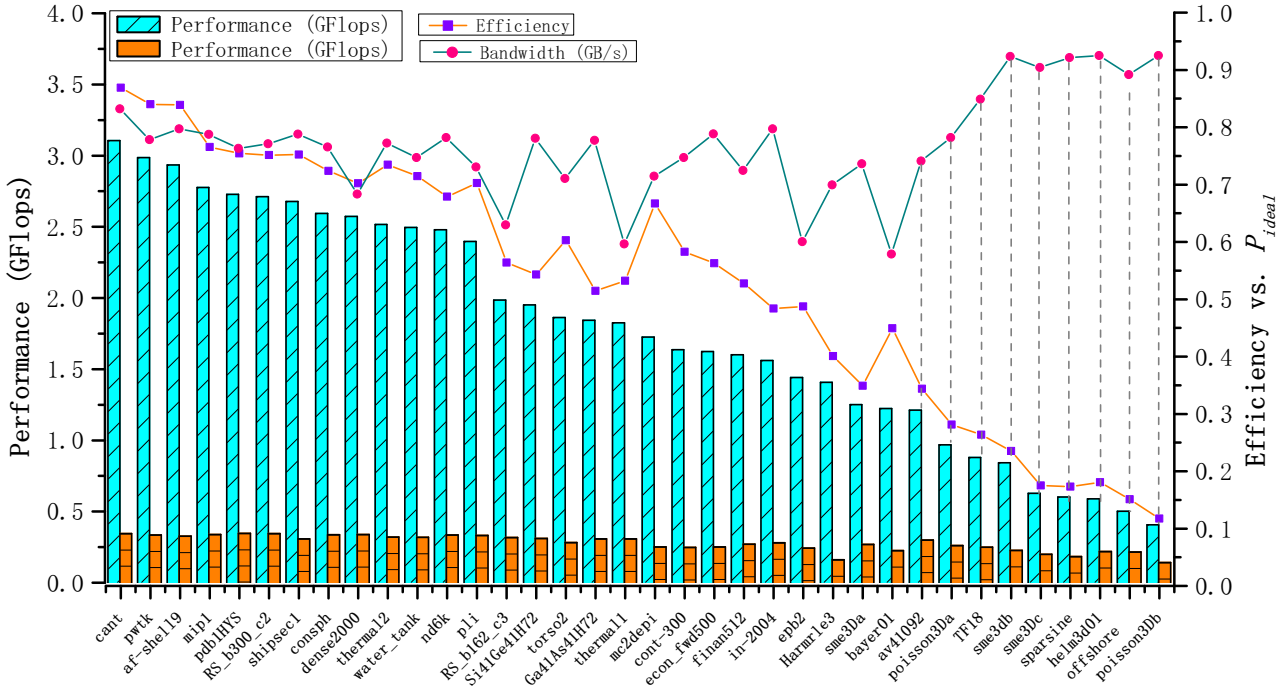


Figure 6: The overview of SW$_{CSR}$-SpMV on all the testing matrices.

achieve as high as 86.8% for the matrix "cant", and generally a much higher performance can be achieved for the (quasi-)banded matrices from numerical applications, such as "cant", "pwtk" and "water-tank". For the leading 13 matrices averagely 2.67 GFlops can be sustained, i.e. 77% of the corresponding average P_{ideal}. Wherein, a slight bandwidth dip happens to "dense2000" and this is because the LDM capacity is so limited that each row-slice is consisted of only 2 rows and DMA overhead is non-negligible when uploading too many short *slice_idx_i* vectors. The bandwidth together with the performance decline for the following 14 matrices. We believe there are two disturbing factors that contribute to the decline: despite more irregular matrix structures, some of the matrices, such as "RS-b162-c3", "thermal1", "epb-2" and "bayer01", are small in volume and

the parallelization overhead as well as the DMA latency are non-negligible. As highlighted in Fig. 6, for the last 9 matrices (in Table 4) the discrepancy between the measured DMA bandwidth and the performance is strikingly large, which implies that a large amount of x'es are redundantly accessed. As shown in the next subsection, the worker group can greatly accelerate the performance for these matrices.

4.3.3 Worker Group for Irregular Matrices. Generally the row-slices of the matrices in Table 4 cover a wide range of vector **x**. Thus the uploaded x segments by one CPE can be potentially reused by other CPEs. As shown in Table 5, the memory footprint of the matrices are significantly reduced with the increase of worker count in each group.

Table 5: The actual DMA volume (in MB) measured in normal SW$_{CSR}$-SpMV and the worker group variant.

Matrix	Normal	Group of 2	Group of 4	Group of 8
av41092	47.41	72.54	48.85	35.21
poisson3Da	13.31	10.43	7.34	5.79
TF18	73.73	82.57	56.90	41.44
sme3db	116.08	102.30	63.37	43.86
sme3c	13.31	10.43	7.34	5.79
sparsine	115.41	102.30	63.37	43.85
helm3d01	32.78	22.86	14.20	9.82
offshore	363.99	328.29	288.36	191.80
poisson3Db	275.45	234.32	113.38	81.94

As shown by Fig. 7(a), there are sounding speedups when the 8 CPEs in a row can share the segments. As for the bandwidth, however, despite the reduced memory access volume, the whole CPE mesh suffers great loss as shown in Fig. 7(b). We believe this is caused by the synchronization barrier which has to wait for the the slowest workers.

4.4 Application in PETSc Application

SW$_{CSR}$-SpMV is intended to support numerical applications ported to Sunway TaihuLight supercomputer. We choose the application of simulating the earth magnetic field. The application has two sets of linear systems to solve and SW$_{CSR}$-SpMV is applied to both of them at the same time, which put the usability of SW$_{CSR}$-SpMV to the test. We conduct tests on both the strong and weak scalability of the program and the measurement is the duration of one time step, including that of the pre-process of SW$_{CSR}$-SpMV.

In the weak-scalability test, each MPI processor compute a 3 dimensional area which is 16∗100∗100 in size, while a 64∗64∗64 area is equally divided by the processors used in the strong-scalability test. As shown in both Fig. 8(a) and Fig. 8(b), by using SW$_{CSR}$-SpMV alone the application can sustain a speed up of 1.75x-2.69x in these multi-process environments, which indicates that SW$_{CSR}$-SpMV has a high usability and can potentially accelerate a set of sparse applications. Further profiling suggests that, among other optimizations the dynamic look ahead scheme significantly helps to boost the performance by 41% to 2.x in the weak/strong scalability tests respectively.

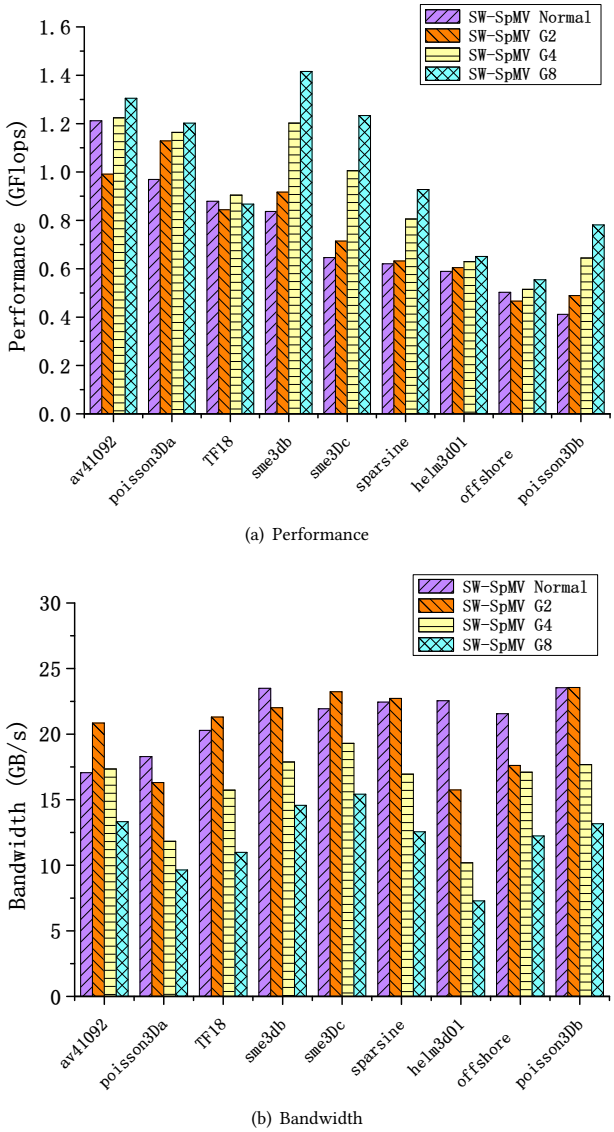
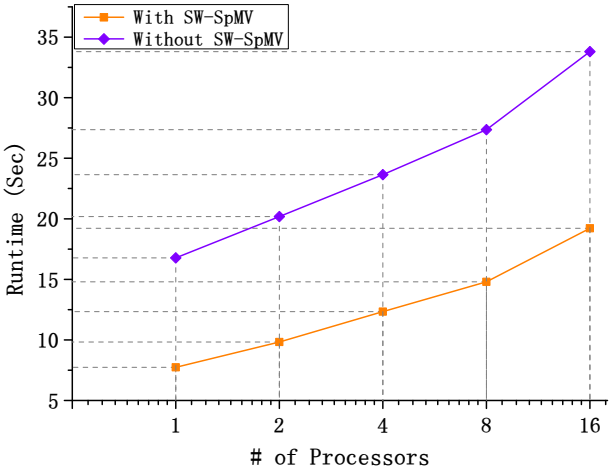


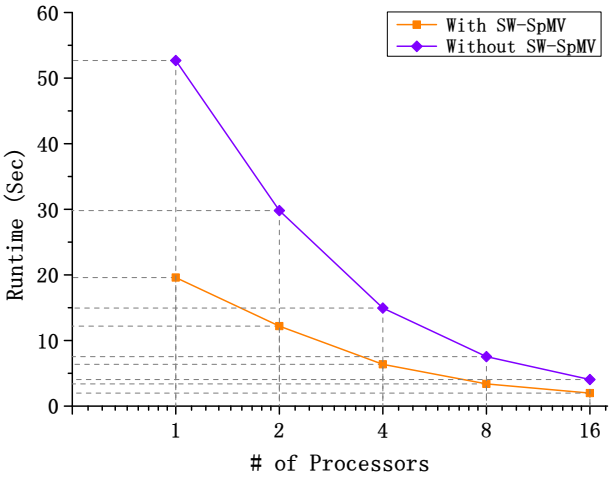
Figure 7: The performance and bandwidth profile on the worker group version of SW$_{CSR}$-SpMV.

5 RELATED-WORK

During the past years, SpMV has always been a hot research topic in the HPC domain. With the rapid development of HPC platforms, different formats for storing sparse matrices are devised to adapt SpMV with specific hardware features, such as DIA, ELL, COO and HYB [31]. These basic formats have been selectively supported by various libraries [1, 18, 31] on the main-stream platforms. To be more adapted with different optimization techniques, extensions are made and give rise to new formats such as BCCOO(+) [36], ELLPACK, ELLR [23, 34] and ELLR-T[41]. Beyond these standalone formats, hybrid formats also turns out to be effective as reported in [9]. Later the formats are dynamically combined in [33], which



(a) Weak-scalability test.



(b) Strong-scalability test.

Figure 8: The scalability test on the earth magnetic field application.

leads to the Cocktail format. By deciding the most suitable format for different matrix partitions on-the-fly, the Cocktail format can achieve slightly higher performance with the prior selected best format. Although each format is proved to be suitable for a certain set of testing matrices on a specific platform, no evidence has shown that there is a universally optimal format [22] in the case of SpMV. The CSR format, due to its simplicity and high compression ratio, is widely adopted by numerical computing frameworks such as Petsc and Hypre etc.. On the newly platform, SW26010, the first priority of optimizing SpMV should lie at the compatibility rather than demonstrating the extreme performance for some testing matrices by utilizing newly developed formats. Besides, there are disadvantages for SpMV approaches with customized sparse formats: Spatially, it is a waste of memory space when the same matrix is duplicated in different formats in applications. Temporally, these approaches also suffer from a time-demanding format-converting phase [4, 20, 22]. Therefore it is pointed out in [22] that the raw performance of the computing phase of SpMV alone cannot determine the performance of SpMV, though the typical usage of SpMV is often repetitive. In SW_{CSR}-SpMV, the setup phase is much less time-demanding, because it only scans the meta-information of the sparse operator without transferring the data to newly allocated memory spaces.

Although SW_{CSR}-SpMV is a platform-specific solution, there are also enlightening works on the main-stream platforms on the basis of an extended CSR formats. SW_{CSR}-SpMV shares similar algorithmic skeleton with CSR-scalar/v-ector [35], CSR-adaptive[8, 39] and CSR-stream[2], enjoying the natural row-wise parallelism. However, these format extensions focus on the utilization of vectorization in the computing kernel, which leads to preferable performance on MIC and GPGPUs. On the target platform, however, the memory access overhead is the main performance bottleneck. Moreover, due to the lack of vectorized scatter/gather instructions on the CPE, the direct vectorization to the computing kernel may cause performance loss by the element-wise load of the oprands. The load imbalance issue, which is caused by the fine-grained parallelism on GPUs, has been addressed by [5, 26, 27]. We adopt the atomic operation based load-balancing strategy in SW_{CSR}-SpMV, which is shown to be effective. As an across-platform approach, CSR5 based SpMV [24] has shown excellent performance on multi/many core CPUs and GPGPU, though it requires a relatively slow setup phase computing auxiliary information including the tile-wise transpose of the source matrix. Consequently, the CSR5 based approach can only deliver noticeable speedups in cases of over 50 times of repetition [24, 25].

6 CONCLUSION AND FUTURE WORK

On the novel SW26010 many-core CPU, we have proposed an efficient implementation of the parallel SpMV primitive, SW_{CSR}-SpMV. SW_{CSR}-SpMV is based on the prevalent CSR format and compatible for high-level numerical frameworks, which enables SW_{CSR}-SpMV to provide convenient solution to accelerating sparse applications on the Sunway TaihuLight supercomputer. While both the hardware features and the fundamental APIs of the platform are more suitable for regular problems, several improvements have been made in SW_{CSR}-SpMV to handle irregular memory access patterns, the redundant memory footprint and the imbalanced workload. The size of each row slice is limited so that they can reside in the LDM and the contiguously uploaded x'es can be shared with each sparse row. We design the dynamic look-ahead scheme to avoid loading of useless x'es by the DMA read operation. Based on register-communication, the issue of redundant memory footprint is also addressed at the worker group level by sharing the collectively needed data. The workload of each CPE thread, or worker group, is balanced both statically and dynamically: the original problem is perfectly partitioned, in terms of the floating-point operations, into row-slices and the atomic operation based dynamic work-sharing strategy is applied to avoid CPE idleness during the execution. Last but not least, an auto-tuning framework for the critical parameters in SW_{CSR}-SpMV is provided and the optimal combination can be easily obtained for each target matrix. Experiments are conducted

on 36 frequent used matrices and the applied optimizations are proved to be effective and SW$_{CSR}$-SpMV has demonstrated a competitive overall performance: based on the theoretical performance upper-bound, SW$_{CSR}$-SpMV can achieve an efficiency of 86.8% in the best case and 1/3 of the matrices can achieve more than 75%. For the matrices that SW$_{CSR}$-SpMV performs relatively poorly, the worker groups formed by SW$_{CSR}$-SpMV can boost the performance by 3% - 90%. Finally, applied to both of the linear systems in the earth magnetic filed simulating program, SW$_{CSR}$-SpMV has significantly accelerated the real-world application by 1.75x-2.69x.

In the future, the extension of this work will be two-fold on the target platform. On one hand, SW$_{CSR}$-SpMV will gain the ability to leverage the 4 CGs in a SW26010 CPU, when a customized threading library, instead of the MPI process, is available; on the other hand we intend to propose efficient parallel SpMV implementations in other frequently used sparse formats, and we believe this practice may benefit a wider range of applications on the Sunway TaihuLight supercomputer.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. The work is supported by the National Natural Science Foundation of China under Grant No.: 616725028, U1636213, 61379048 , the National Key Research and Development Program of China under Grant No.: 2017YFB1400902 and the Key Research and Development Plan of Hebei Province(18390308D).

REFERENCES

[1] 2014. CUSP: A C++ Templated Sparse Matrix Library.
[2] 2014. The Open Standard for Parallel Programming of Heterogeneous Systems. https://www.khronos.org/opencl.
[3] 2017. Top-500 supercomputer list in 2017. https://www.top500.org/lists/2017/06/.
[4] Pham Nguyen Quang Anh, Rui Fan, and Yonggang Wen. 2016. Balanced Hashing and Efficient GPU Sparse General Matrix-Matrix Multiplication. (2016), 1–12.
[5] Arash Ashari, Naser Sedaghati, John Eisenlohr, Srinivasan Parthasarathy, and P. Sadayappan. 2014. Fast sparse matrix-vector multiplication on GPUs for graph applications. In *International Conference for High PERFORMANCE Computing, Networking, Storage and Analysis*. 781–792.
[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. 2016. PETSc Web page. http://www.mcs.anl.gov/petsc. http://www.mcs.anl.gov/petsc
[7] Vicente H. F. Batista, George O. Ainsworth Jr, and Fernando L. B. Ribeiro. 2010. Parallel structurally-symmetric sparse matrix-vector products on multi-core processors. *Computer Science* (2010).
[8] Nathan Bell and Michael Garland. 2009. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Conference on High PERFORMANCE Computing Networking, Storage and Analysis*. 1–11.
[9] Wei Cao, Lu Yao, Zongzhe Li, Yongxian Wang, and Zhenghua Wang. 2010. Implementing Sparse Matrix-Vector multiplication using CUDA based on a hybrid sparse matrix format. In *International Conference on Computer Application and System Modeling*. V11–161 – V11–165.
[10] Hoang Vu Dang and Bertil Schmidt. 2013. CUDA-enabled Sparse Matrix–Vector Multiplication on GPUs using atomic operations. *Parallel Comput.* 39, 11 (2013), 737–750.
[11] T. Davis and Y. Hu. 2018. University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/.
[12] Robert D. Falgout and Ulrike Meier Yang. 2002. hypre: A Library of High Performance Preconditioners. In *International Conference on Computational Science*. 632–641.
[13] John R. Gilbert, Steve Reinhardt, and Viral B. Shah. 2007. High-Performance Graph Algorithms from Parallel Sparse Matrices. In *Applied Parallel Computing. State of the Art in Scientific Computing, International Workshop, Para 2006, Umeå, Sweden, June 18-21, 2006, Revised Selected Papers*. 260–269.
[14] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. 2000. Towards Realistic Performance Bounds for Implicit CFD Codes. *Parallel Computational Fluid Dynamics* (2000), 241–248.
[15] F. U. Haohuan, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, and Fangli Qiao. 2016. The Sunway Taihu Light supercomputer:system and applications. *Science China Information Sciences* 59, 7 (2016), 072001.
[16] Michael Heroux, Roscoe Bartlett, Vicki Howle, Robert Hoekstra, Jonathan Hu, Tamara Kolda, Richard Lehoucq, Kevin Long, Roger Pawlowski, and Eric Phipps. 2003. An Overview of Trilinos. *Sandia National Laboratories* 30, 1 (2003), 1095–1101.
[17] Eun Jin Im and Katherine Yelick. 2000. Optimization of Sparse Matrix Kernels for Data Mining. In *Siam Conf on Data Mining*.
[18] Intel. 2018. Math Kernel Library, MKL.
[19] Yuji Kubota and Daisuke Takahashi. 2011. Optimization of Sparse Matrix-Vector Multiplication by Auto Selecting Storage Schemes on GPU. In *International Conference on Computational Science and ITS Applications*. 547–561.
[20] Pramod Kumbhar. 2011. Performance of PETSc GPU Implementation with Sparse Matrix Storage Schemes. (2011).
[21] D. Langr and P. Tvrdik. 2016. Evaluation Criteria for Sparse Matrix Storage Formats. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (Feb 2016), 428–440. https://doi.org/10.1109/TPDS.2015.2401575
[22] D Langr and P Tvrdik. 2016. Evaluation Criteria for Sparse Matrix Storage Formats. *IEEE Transactions on Parallel & Distributed Systems* 27, 2 (2016), 428–440.
[23] Ruipeng Li and Yousef Saad. 2013. GPU-accelerated preconditioned iterative linear solvers. *Journal of Supercomputing* 63, 2 (2013), 443–466.
[24] Weifeng Liu and Brian Vinter. 2015. CSR5:An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. In *ACM on International Conference on Supercomputing*. 339–350.
[25] Weifeng Liu and Brian Vinter. 2015. Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors. *Parallel Comput.* 49, C (2015), 179–193.
[26] Xing Liu, Mikhail Smelyanskiy, Edmond Chow, and Pradeep Dubey. 2013. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *International ACM Conference on International Conference on Supercomputing*. 273–282.
[27] Yongchao Liu and Bertil Schmidt. 2015. LightSpMV: Faster CSR-based sparse matrix-vector multiplication on CUDA-enabled GPUs. In *IEEE International Conference on Application-Specific Systems, Architectures and Processors*. 82–89.
[28] Hiroshi Maeda and Daisuke Takahashi. 2016. *Parallel Sparse Matrix-Vector Multiplication Using Accelerators*. Springer International Publishing.
[29] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Technical Report. University of Virginia, Charlottesville, Virginia. http://www.cs.virginia.edu/stream/ A continually updated technical report. http://www.cs.virginia.edu/stream/.
[30] Duane Merrill and Michael Garland. 2016. Merge-based sparse matrix-vector multiplication (SpMV) using the CSR storage format. (2016), 1–2.
[31] Nvida. 2018. CUSparse Library.
[32] S Ohshima, T Katagiri, and M Matsumoto. 2014. Performance Optimization of SpMV Using CRS Format by Considering OpenMP Scheduling on CPUs and MIC. In *IEEE International Symposium on Embedded Multicore/manycore Socs*. 253–260.
[33] Bor Yiing Su and Kurt Keutzer. 2012. clSpMV:A Cross-Platform OpenCL SpMV Framework on GPUs. In *ACM International Conference on Supercomputing*. 353–364.
[34] F. Vázquez, J. J. Fernández, and E. M. Garzón. 2011. A new approach for sparse matrix vector product on NVIDIA GPUs. *Concurrency & Computation Practice & Experience* 23, 8 (2011), 815–826.
[35] S Williams, L Oliker, R Vuduc, and J Shalf. 2009. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. *Parallel Comput.* 35, 3 (2009), 178–194.
[36] Shengen Yan, Chao Li, Yunquan Zhang, and Huiyang Zhou. 2014. yaSpMV: yet another SpMV framework on GPUs. In *ACM Sigplan Symposium on Principles and Practice of Parallel Programming*. 107–118.
[37] Fan Ye, Christophe Calvin, and Serge G. Petiton. 2014. *A Study of SpMV Implementation Using MPI and OpenMP on Intel Many-Core Architecture*. Springer International Publishing. 43–56 pages.
[38] J. Zhang, C. Zhou, Y. Wang, L. Ju, Q. Du, X. Chi, D. Xu, D. Chen, Y. Liu, and Z. Liu. 2016. Extreme-Scale Phase Field Simulations of Coarsening Dynamics on the Sunway TaihuLight Supercomputer. (Nov 2016), 34–45. https://doi.org/10.1109/SC.2016.3
[39] Yunquan Zhang, Shigang Li, Shengen Yan, and Huiyang Zhou. 2016. A Cross-Platform SpMV Framework on Many-Core Architectures. *ACM Transactions on Architecture & Code Optimization* 13, 4 (2016), 33.
[40] Hong Zhou, Xiaoya Fan, and Lili Zhao. 2010. Optimizations on Sparse Matrix-Vector Multiplication Based on CUDA. *Computer Measurement & Control* 18, 8 (2010), 1906–1895.
[41] Francisco Zquez, Jos Ndez, Garz, and Ester M N. 2012. *Automatic tuning of the sparse matrix vector product on GPUs based on the ELLR-T approach*. Elsevier Science Publishers B. V. 408–420 pages.