SIMD Acceleration of SpMV Kernel on Multi-core CPU Architecture

J.Saira Banu and Dr M.Rajasekhara Babu

School of Computing Science and Engineering, VIT University, Vellore, India.

Abstract
As the field of Sparse Matrix vector multiplication (SpMV) matures and its breadth of application increases, the need for parallel implementation becomes necessary. SpMV is proved to be a bottleneck due to its irregular access patterns. Various storage formats for sparse matrices have been proposed to solve this issue. The Quad tree-Compressed Row Storage (QCSR) format shows good performance improvement over other formats such as Compressed Row Storage (CSR) and Blocked CSR (BCSR) for SpMV. This paper extends QCSR format to exploit the Single Instruction Multiple Data (SIMD) registers which are available in current processors. Programming with SIMD registers in a single core processor achieves parallelism with reduced power and without any additional hardware requirement, as in the case of Graphics Processing Unit (GPU) computing. To program effectively in SIMD units Intels Streaming SIMD Extension (SSE) instructions are used. In this paper computational performance of QCSR-SpMV is determined for over a collection of 10 benchmark matrices on SIMD units of X86 architecture. Experimental results demonstrate QCSR-SIMD achieves significant average speedup of 2.0 x compared to CSR-SIMD.
Keywords OpenMp; GPU; Sparse Matrix; SpMV; CSR; QCSR; SIMD; SSE; Performance; Programming model; Optimization; Parallel computing.

1 Introduction

There are multiple ways of accomplishing parallel computing in single machine to cloud environment. In single machine, parallel computing is achieved through thread level parallelism with Open Multiprocessing (OpenMP), message passing method with Message Passing Interface (MPI) and through SIMD technique with SSE instruction. Increasing clock speed has been ceased recently which gave a way for multi or many cores CPU. The other alternative method to achieve many core computing is through GPU. In GPU, parallelization is achieved with the expense of new hardware. Parallel computing in cloud environment is realized by distributing the computing task to various computing resources and gathering the result. In cloud computing allocating task is a major issue. Guiyi Wei, et al addressed this issue of allocating the task to various computing resources in the cloud[1]. It involves allocation matrix and expense matrix. If a matrix used in an algorithm is dense then there is no issue but algorithm involving sparse matrix is a concern one.

A lot of work is being done in the field of sparse matrices and SpMV, to improve the efficiency. Emphasis has been laid on the parallel implementation of SpMV in the past decade. Sparse matrix structures are present in various applications, and efficient method for optimizing the performance of these applications is a critical issue. SpMV is an operation found in many computational science applications. Improving the speedup of SpMV boosts the performance of these applications. Extensive research has been done in improving the performance of SpMV using thread level and distributed memory parallelism. Samuel Williams et al. examined SpMV on various multicore design namely AMD quad core, INTEL quad core[2], etc. Shengfei Liu, et al. worked on SpMV with Compressed Sparse Row (CSR) and Blocked CSR (BCSR) format[3]. They have implemented multithreaded SpMV using OpenMP. Partitioning the matrices in to sub matrices and hybrid programming model with MPI and OpenMP are the future methods proposed in the paper to increase the performance of the kernel. Dakuan discussed SpMV with MPI[4]. Gerald Schubert et al. examined the performance of SpMV kernel with hybrid programming (OpenMP and MPI) model[5]. This paper does not explore the load balancing issues. M.Krotiewski and M.Dabrowski presented a parallel implementation of SpMV on multicores with well- known optimization such as matrix reordering, blocking and prefetching[6].

With the rise of high performance computing, most of the focus is now on GPU computing. In literature, different formats of storage have been developed to either improve the space efficiency, or the time of access of non-zero elements for various operations. Some formats are specific to the execution environment such as CPU or GPU. Tomas Oberhuber et al. proposed a new format such as new row grouped CSR which performed well in GPU devices[7]. Jilin Zhang et al. have implemented SpMV using a new type of storage format, Quad tree CSR (QCSR) format using GPUs[8]. This format outperforms the Blocked CSR (BCSR) format.

Recently vectorization capability using SSE instructions of the current processor proved to be better technology to improve the performance of many applications. Susana Ladra, et al. exploited SIMD instructions in current processors to improve the classical string algorithms[9]. Kai Zeng et al. proved the usage of SIMD technique in computed tomography CT reconstruction is efficient rather than GPU implementation[10]. Image processing applications such as feature detection, stereo vision class model estimation, and object detection achieved better performance on the Compressed Sparse Extended (CSX) SIMD architecture compared with GPU[11]. S.J.PennyCook et al. explored the use of SIMD registers for molecular dynamics problem set[12].

To obtain better performance on numerical kernels, Martin Kong et al. proposed a 3-step framework such as data locality, multicore parallelism and SIMD execution of programs[13]. Libo Huang et al. presents a dynamic vectorization method to address the constraints in current SIMD engines such as register variation in the processors requires changing of vector operands and aligned memory accesses[14]. NaserSedaghati proposed an extension to the instruction available in Instruction Set architecture (ISA) to overcome the disadvantage of SIMD for stencil computation application[15]. Neil G. Dickson et al. found that explicit vectorization on the CPU gave 9x-12x speedup over the original CPU version, and was also found to be 2x faster than the fully optimized GPU version that uses explicit memory coalescing. These papers signifies the importance of single core optimization on CPU[16].

Ji-Lin Zhang et al. have implemented SpMV using CSR format with SIMD methodology and thread-level parallelizing[17]. They found out that these methods gave a speed up of around 2.11 over that of the non-optimized method. Kai Zhang et al. improved upon this and broke the performance bottlenecks of the SIMD processors like the low utilization of SIMD processors and the memory bandwidth[18]. Their method showed a good speedup over the normal CSR vector kernel. NazliGoharian et al. demonstrated that CSR is the best storage format for information retrieval and query processing among various storage format of sparse matrix[19].These paper explored SIMD in SPMV and its importance.

In this paper, we investigate the impact of SIMD acceleration on SpMV Kernel. For this, SpMV kernel with CSR and QCSR format has been accelerated, i.e., SIMD is applied to all suitable operations and implemented in Haswell processor using SSE instructions.
The main contribution of this paper are as follows.

1. Made an extensive study on strorage formats of sparse matrix structure and analyzed the space complexity for benchmark matrices from different applications.
2. SIMD acceleration is presented for all the data-parallel kernnels of SpMV with CSR and QCSR format.
3. Implementation and optimization is performed with SSE instructions.
4. SIMD implementation of CSR outperforms the na?ve and thread level parallelism.
5. With SIMD optimization,SpMV with QCSR format gives 2 fold of speedup compared to CSR-SIMD.

The rest of the paper is organized as follows: Section 2 deals with architecture specifications, Section 3 describes the SpMV algorithm with its storage format analysis; Section 4 explores about SIMD optimization of SpMV kernel with CSR and QCSR storage formats; Section 5 deals with the results and observations; Section 6 talks about the conclusion.

2 Architecture Specification

Since the beginning of the processor, its performance has been steadily increasing. Developments in semiconductor capability are the main reason for this remarkable change. Advancement in architectural features like instruction- level parallelism, data-level parallelism (DLP) thread-level parallelism (TLP) and memory-level parallelism (MLP) brought remarkable improvement in performance. Here, in this paper DLP is achieved through SIMD, and TLP is done through OpenMP.

2.1 DLP-SIMD

SIMD is one of the classifications of the Flynns taxonomy of computer architec− ture[20]. It exploits DLP by performing the same operation on multiple data points simultaneously. SIMD architecture consists of multiple processing elements that perform identical task concurrently on the data elements as shown in Fig.1.

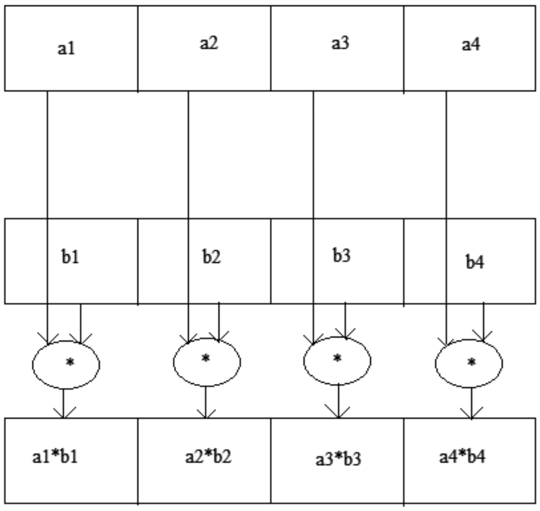

Fig. 1 SIMD architecture.

SSE instructions are now available in current generation processors to program effectively with SIMD registers. These instructions are used effectively to vectorise the code and to modify the data layout. Automatic vectorization has certain limitations such as handling loops of irregular access patterns, pointers, etc. Since Automatic vectorization has restricted applicability, programmers write vectorised code using intrinsic which is directly expanded to machine instructions. Programming using intrinsic are tied to a specific Instruction set. With the help of these SSE instructions, SIMD Programming is performed in high level programming languages such as C. Compared to GPU, there is no overhead incurred, like moving data from device to device or thread processing.

It can also be combined with thread level parallelism technique to increase the speed up further. SSE is included from Pentium III processors by adding 128 bit registers and the instructions that can operate on them. Advanced Intel processors supports AVX which includes 256 bit register with extra SIMD instructions in addition to SSE instruction set. SSE instruction set includes integer and floating point arithmetic Instructions, comparison, shuffling, data type conversion, bitwise operations, minimum, maximum, conditional copies, CRC 32 and population count. Currently in many applications such as gaming, graphics, physics, and Mathematics, SIMD instructions are used to perform shuffling, scalar product, checksum calculations and complex operations. There are various versions of SSE such as SSE- SSE4.2 which are capable of processing 2 double precision floating point numbers or 4 single precision 32 bit floating point numbers. SSE intrinsic is used to improve the performance of an application which has fine grain parallelism in it. When dealing with sparse matrices, one of the most important restrictions is that the average width of non-zero elements per row must be greater than the SIMD width of the computer device. Else, the SIMD registers are only partially utilized, and do not give a considerable increase in the efficiency.

2.2 TLP - OpenMP

Thread level parallelism is achieved when a single process is divided into various sub- parts, and each part is carried out by a different thread, so at the end, it could be combined to give the result. OpenMP is an API that provides shared memory multi-processing programming using high level languages such as C, C++ and FORTRAN. There are various methods to perform thread level parallelism for SpMV-CSR kernel. Parallelism can be done row wise or column wise or block wise. Here, we have performed row wise parallelism, where each row is assigned to a thread. The accesses to the data are independent and hence can easily be parallelized. One advantage of using the row wise partitioning is that individual threads operate on different parts of the final resultant array unlike that of the column wise partitioning, where all threads have to write to all parts of the array.

3 SpMV Algorithm

SpMV forms a basic computational kernel in many scientific and industrial applications. Modern High Performance computer systems (HPC) such as multicores, GPU,SIMD computation using coprocessor and special XMM registers rely on Sp-MV computation for numerous task. SpMV is usually represented as $Y = A * X$ where A represents a sparse matrix and Y and X represents a vector. Sparse matrix is a matrix which has more number of zero elements than the non-zero values. Storing a sparse matrix as it is in a memory is a space overhead and in some applications it doesnt fit in to the available memory. To effectively reduce the storage space many data structures have been proposed in the literature.

These data structure stores only the non-zero elements of the matrix and thereby reduces the computation of SpMV algorithm. This is important in HPC community since all modern computer system is equipped with small storage devices. The compression of sparse matrix using efficient storage format influence the performance of SpMV operation.For the remainder of the paper, we use the following notation:

N : Total number of non-zero values

n : order of the matrix

S( ): Storage space occupied

Nr : Number of rows

3.1 Storage formats

Sparse matrices are stored in computer memory with specific formats that give high performance of SpMV operation. Few of the popular formats of storing sparse matrices with space and time complexity are discussed here.Equations 1-5 represents the space complexity of various storage formats.

3.1.1 The Coordinate Format (COO)

This is one of the most general purpose formats. It consists of three 1-dimensional arrays - one to store the non-zero values, one to store the corresponding column index, and the other to store the corresponding row index [21].

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Fig. 2 Example Matrix.

For example, the COO format for the matrix shown in Fig.2 is represented as : Data = [1 1 1 1 1] Col = [1 1 2 2 4] Row = [1 2 2 3 3].

a ***Time complexity***: The time complexity to convert this format to/from the Compressed Row Storage (CSR) format is O (N+n)

b ***Space complexity***: The space complexity of this format excluding the data array is given by

$$S(COO) = 2 \cdot N \cdot S(n) \tag{1}$$

3.1.2 The Compressed Row Storage Format (CSR)

This is one of the most efficient and hence most popular storage formats. It consists of three 1-dimensional arrays C one to store the non-zero value, one to store the corresponding column index, and the other to store the row pointer value, that gives the total non-zero values in each row [21] for the example in figure 2:

CSR format is given as:
Data = [1 1 1 1 1]
Col = [1 1 2 2 4]
Row-Ptr = [0 1 3 5 5]

a. ***Time complexity***: The time complexity to convert this format to/from the Coordinate (COO) format is O (N+n).

b. ***Space complexity***: The space complexity of this format excluding the data array is given by

$$S(CSR) = N \cdot S(n) + n \cdot S(N) \tag{2}$$

3.1.3 The Quadtree storage format

Quadtree is a recursive tree data structure given by Ivan Simecek [22]. The matrix is recursively divided into four quadrants until each block is equal to a predefined size (called density). These blocks or nodes can be of three types:

a. Empty: The entire node is made up of zeroes.

b. Mixed: The node consist of a mixture of zeroes and non-zero elements.

c. Full: The entire node id made up of non-zero elements.

Empty nodes are ignored, and the mixed and full nodes are stored in any format that is found to be appropriate for the application.The matrix in Fig.1, is divided in to four quadrants and each quadrant is checked for the node types. Here the second quadrant is empty and it is ignored and all other quadrants are further retrieved through the indices for its operation as shown in Fig.3. The advantages of this format are:

a. Easy conversion from popular formats.

b. Easy modifications to the data.

c. The recursive style of programming leads to better performance due to better cache memory utilization.

a. ***Time complexity:*** The entire algorithm can be broken down into smaller functions. The time complexity of each such function is as given below:

(1) To find the total non-zero values in each quadrant: O (average_per_row. (y2 - y1 + 1)).
(2) To check the given quadrant is empty or not:O (log2 average_per_row. (y2 - y1 + 1)).
(3) For converting the matrix into Quadtree format, the complexity depends on the density.

b. ***Space complexity:*** The space complexity completely depends on the density chosen and the creation of leaves.

3.1.4 The QCSR (Quadtree Compressed Row Storage) format

The QCSR format is a combination of the Quadtree format and the CSR format(Jilin Zhang, etal, 2013)[8]. After dividing the matrix into various nodes using the Quadtree logic, the Mixed and Full nodes are stored in the CSR format. Though it is found to have a space overhead, the implementation of SpMV is much faster in this format.

a. ***Time complexity***: The time complexity of this format is the same as that of the Quadtree format, in addition to which each quadrant is converted into the CSR format.

b. ***Space complexity***: This format gives an overhead in space.The maximum overhead over the CSR format is given by:

$$Soh(QCSR) = (2Sr + Sl)xO(4d - 1) \tag{3}$$

Where, Sr is the space occupied by the index pointer to the region, Sl is the region length and d is the maximum depth of the tree.

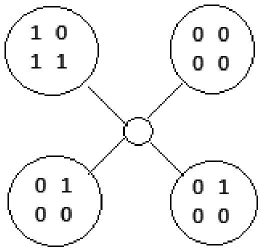

Fig. 3 Splitting of matrix.

3.1.5 Minimal Quadtree format:

This format is developed by I.Simecek et al.[23]. It is a derivative of the Quadtree format and consists of a bit stream of 1s and 0s. The matrix is recursively divided into blocks as in Quadtree format, but here, each block is represented by a single bit- 0 if all the elements in the block are zeroes, 1 otherwise. The example given in Fig. 1, can be divided into 4 blocks if the density is 2, and can be represented as 4 bits- 1011 for only the second block is an empty block, and the others have non-zero elements.

a. ***Time complexity***: The total time complexity for this conversion is

$$O(N(n+\sqrt{N}))\cdot \log_2 avg\_per\_row$$

b. ***Space complexity***: The minimal size of the MQT format is given by:

$$S(MQT_{\min}) = 4\cdot(\frac{N}{3}+\log_4(\frac{n^2}{N})) \tag{4}$$

The maximal size of the MQT format is given by:

$$S(MQT_{\max}) = 4\cdot(\frac{1}{3}+\log_4(\frac{n^2}{N})) \tag{5}$$

As minimization of memory is an important criterion for SpMV operation, we compared thespace complexity of various formats for the benchmark matrices obtained from University of Florida database [24]. Table 1 shows the properties of benchmark matrices used. Table 2 depicts the space occupied for these matrices in COO, CSR, QCSR and MinQCSR format.

Table 1 Overview of the benchmark matrices

Benchmark Matrices	**Number of rows**	**Number of columns**	**Number of non- zero elements**
bcsstk07	420	420	4140
adder_dcop_08	1813	1813	11242
mhd4800b	4800	4800	16160
Meg4	5860	5860	26324
gemat11	4929	4929	33185
Cell1_b	7055	7055	34855
Ex12	3973	3973	42092
SiNa	5743	5743	102265
Na5	5832	5832	155731

From table 2, it is evident that Minimal QCSR, which is just a series of bits is well suited for storing the structure of matrices and hence not suitable for SpMV operation. CSR is the next format that has the least space complexity and proved to be well utilized in SpMV operation. QCSR format has the space overhead but that is not accounted for SpMV operation. Hence, among all the formats that have been developed so far, the CSR format is the most efficient and effective format. A lot of improvements have been done on this format to improve its efficiency. In this paper, we analyse the CSR format along with the QCSR format for SpMV computation in SIMD architecture.

Table 2 The space complexity of various formats for benchmark matrices

Benchmark Matrices	**Original**	**COO**	**CSR**	**QCSR**	**MinQCSR**
bcsstk07	391KB	97KB	76KB	118KB	394b
adder_dcop_08	6.35MB	326KB	176KB	265KB	999b
mhd4800b	44MB	436KB	263KB	416KB	810b
Meg4	65.7MB	425KB	386KB	854KB	739b
gemat11	46.5MB	648KB	506KB	1.2MB	367b
Cell1_b	95.2MB	587KB	386KB	854KB	739b
Ex12	30.4MB	1.13MB	635KB	845KB	349b
SiNa	63.6MB	2.95MB	1.52MB	3.19MB	46b
Na5	65.9MB	4.54MB	2.3MB	4.25MB	384b

3.2 SpMV - CSR

SpMV kernel acts as a core kernel of many iterative algorithms and scientific applications. It is one of the time consuming kernel in these methods. Optimizing this kernel plays a vital role in improving the performance of these applications. The performance of SpMV kernel is based on the storage format used. CSR format has been proved to be the best format with space efficiency. Algorithm 3.1 and 3.2 shows SpMV with CSR format in naive and thread level parallelism.

Algorithm 3.1 SpMV with CSR- Naive

```
Inputs
    Ptr - Pointer Array
    Data - Data Array
    Column - Column index array
    X - Vector for multiplication
Output:
    Z - Result Array

1. Begin
2. for each item i from 0 - Nr do
3. Z[i] = 0
4. for each item j in Ptr[i] to Ptr[i+1] -1 do
5. temp = Column [j]
6. Z[i] = Z[i]+ (Data[j] * X[temp])
7. end for
8. end for
```

```
9. End
```

Algorithm 3.2 SpMV with CSR-Thread level parallelism

```
Inputs
    Ptr - Pointer Array
    Data - Data Array
    Column - Column index array
    X - Vector for multiplication
Output:
    Z - Result Array

1. Begin
2. ♯ pragma omp parallel num_threads(4)
3. ♯ pragma omp parallel for private (k, j, i) schedule(static, 10)
4. for i = 0 to N - 1 do
5. Z[i] = 0
6. for j = Ptr[i] to Ptr[i+1]-1 do
7. temp = $col_j$
8. Z[i] = Z[i] + (val[j] * X[temp])
9. end for
10. end for
11. End
```

3.3 SpMV - QCSR

As elucidated in Section 3.1, the QCSR format is a combination of the Quadtree format and the CSR format. The given matrix is first divided into various quadrants using the Quadtree format, and the mixed and full nodes are converted into the CSR format as shown in algorithm 3.3 and 3.4 [8]. Here, sr stands for Start Row, er for End Row, sc for Start Column, ec for End Column, a[][] is the input matrix, and density is the size of the quadrant. QCSR format compared to CSR format has space overhead. This overhead is due to storage of block and intermediate node information. This overhead accounts for transformation and representation of matrix and not in SpMV Kernel. This implies that QCSR is suitable for SpMV Kernel with all general cases.

Algorithm 3.3 Matrix Conversion to QCSR format

```
Inputs
    sr - Start Row
    sc - Start Column
```

er - End Row
ec - End Column

```
1. Begin
2. int mr=(sr+er)/2;
3. int mc=(sc+ec)/2;
4. int r=er-sr;
5. int c=ec-sc;
6. if( r≥ density && c ≥ density )
7. {
8. if( !isEmpty(sr,mr,sc,mc) )
9.     trans(sr,mr,sc,mc);
10. if( !isEmpty(sr,mr,(mc+1),ec) )
11.     trans(sr,mr,(mc+1),ec);
12. if( !isEmpty((mr+1),er,sc,mc) )
13.     trans((mr+1),er,sc,mc);
14. if( !isEmpty((mr+1),er,(mc+1),ec) )
15.    trans((mr+1),er,(mc+1),ec);
16. }
17. else if(c>density)
18. {
19.     trans(sr,er,sc,mc);
20.     trans(sr,er,(mc+1),ec);
21. }
22. else if(r>density)
23. {
24.     trans(sr,mr,sc,ec);
25.     trans((mr+1),er,sc,ec);
26. }
27. else
28. {
29. if( !isEmpty(sr, er, sc, ec))
30.     Convert the quadrant into CSR format
31. }
32. End
```

Algorithm 3.4 isEmpty(sr,er,sc,ec)

Inputs

sr - Start Row
sc - Start Column

er - End Row
ec - End Column

```
1. Begin
2. int isEmpty()
3. for all elements i from sr to er
4. {
5.     for all elements j from sc to ec
6.     {
7.         if( a[i][j] != 0)
8.         return 0
9.     }
10. }
11. return 1
12. End
```

4 SIMD Optimization

4.1 SpMV CCSR

SIMD optimization is performed with the help of SSE 4.2 instructions. Here, Nr is the total number of rows in the matrix, val is the array used to store the non-zero elements, col is the array to store the column indices, row is the row pointer array, and X is the vector for multiplication. Algorithm 3.5 gives SIMD optimization of SpMV kernel with CSR format. In vectorization using SIMD technique, four values are operated simultaneously, as the size of the SIMD register is 4. The non-zero elements were loaded into the data register, and the corresponding vector elements in the x register. Here data and X are the special XMM registers available in the current generation processors. These values are multiplied simultaneously, hence decreasing the total number of operations performed.

Algorithm 3.5 SpMV with CSR-SIMD

Inputs

Nr C Number of rows
Val C Data Array
row C Pointer array
X - Vector for multiplication

Output:

Z - Result Array

```
1. Begin
```

```
2. for every element i from 0 to Nr-1 do
3. for every element j in row[i] to row[i+1]-1 do
4. data = {val[j] , val[j+1], val[j+2], val[j+3]}
5. x = {X[col[j]] , X[col[j+1]], Xcol[j+2]], X[col[j+3]]}
6. y = mm_mul_ps(data, x)
7. $Z_i$ = sum{y}
8. j = j + 4
9. end for
10. end for
11. End
```

4.2 SpMV -QCSR

QCSR format is an improvisation on the Quad tree format, in which each of the blocks is stored in the CSR format. We have performed sparse matrix vector multiplication using this format. One condition that has to be met to maximize efficiency is that, the total non-zero values in each block should be greater than or equal to the size of the SIMD register. If not, the remaining bits in the SIMD register must be padded with zeros which results in time overhead. QCSR with SSE instructions is specified in algorithm 3.6. The following instructions are used to vectorize the SpMV implementation of QCSR format.

1. _mm_loadu_ps(array) : This is used for loading the registers with the elements in the array.
2. _mm_mul_ps(reg1, reg2) : This is used to perform vectorized multiplication of the elements in the two mentioned registers. The result is stored in reg1.
3. _mm_hadd_ps(reg1, reg2) : This is used to perform vectorized addition of the elements in the two mentioned registers. The result is stored in reg1.
4. _mm_storeu_ps(ptr, reg) : This is used to store the results in the register to the pointer.

Algorithm 3.6 QCSR with SIMD optimization

Each block in QCSR format consists of the following components:

Startrow: The row index of the first row in the block.
Startcolumn: The column index of the first column in the block.
Endrow: The row index of the last row in the block.
Endcolumn: The column index of the last column in the block.
structure csr: CSR representation of the block, which contains

nz: The total non-zeroes values in the block
data[n]: Data array for the block.
ptr[n]: Row-pointer array for the block
col[n]: Column indices array for the block

MULTIPLY (EACH BLOCK)

i: To keep track of the number of rows in the block.
k: To keep track of the number of non-zero elements in each row.
t: To keep track of the ptr arrays index.
temparr: To store the padded data values.
data1, vect1, temp, res1: SIMD registers.
res: To store the multiplication result.

```
1. Begin
2. for i from startrow to endrow, do
3. for k from csr.ptr[t] to csr.ptr[t+1], do
4. q= csr.ptr[t+1] C csr.ptr[t];
5. a = minimum ( q-k, 4)
6. for j from 0 to a, do
7. temparr[j]= csr.data[k+j];
8. if j < 4
9. Pad the remaining places with 0.
10. end if
11. data1=_mm_loadu_ps ( temparr )
12. vect1=_mm_loadu_ps ( v[col[k]], v[col[k+1]], v[col[k+2]], v[col[k+3]] )
13. temp=_mm_mul_ps (data1 , vect1)
14. res1=_mm_hadd_ps (temp , temp )
15. res1=_mm_hadd_ps (res1 , res1)
16. _mm_storeu_ps ( c , res1)
17. res[i] = res[i] + c
18. end for
19. t++
20. k=k+4
21. end for
22. End
```

5 Results and Discussion

In this section we analyse the performance improvement of CSR-SPMV and QCSR- SpMV format using instructions included in SSE 4.1/4.2 extension. To carry out experiments, we have used Intel(R) Core^{TM} i5-4200U CPU@ 1.60 GHz (2 cores) with 4 GB RAM. Visual Studio 2010 compiler was used for compilation. The width of the SIMD register in the test system is 4. In this experiment, the density size has been chosen to be 16 such that the non- zero elements in each block are equal to or greater than the width of SIMD register.

Speed-up is a performance index, which is a ratio of the time taken to implement a program sequentially to the time taken to do the same in parallel. We first performed SpMV using the CSR format both sequentially and in parallel using OpenMP. In parallel implementation, we have considered the block size to be 10. Each thread is given a single row and multiplication of these rows is performed in parallel. One disadvantage of this method is, if the total non-zero elements in one of the rows are much greater than that in the other rows, then stalling of the other threads occurs. The remaining threads have to be idle until the thread with that particular row finishes execution.

Next, we implemented the same using SIMD vectorization and compared this with that of sequential execution. The SIMD registers are of size 128 bits, and hence 4 32-bit integers can be loaded in it at a given time. Each row is considered separately, and the non- zero elements in it are loaded into the register, 4 at a time. If the number of non-zero elements left is less than 4, it is padded with zeroes. This overcomes the disadvantage of stalling that occurs in thread level parallelism. Another overhead that this method avoids is the time taken for the creation of threads. The efficiency of the execution of SpMV using SIMD completely depends on the arrangement of the non-zero elements in the matrix. Fig. 4 shows results obtained when implementing SpMV using CSR format with SIMD and OpenMP techniques.
From the graph shown in Fig. 4, we determine that the SIMD vectorization technique gives better results when compared to that of sequential and parallel execution. Hence, we implemented SIMD version of SpMV for various benchmark matrices using the QCSR format using this technique of SIMD vectorization. In this method of implementation, each quadrant is considered separately and is treated as a separate matrix.

CSR SpMV using vectorization, as discussed above, is then carried out for each quadrant. Here, the size of the quadrant depends on the density chosen. Since the density is usually in powers of two and much smaller than the original matrix size, the probability that the total non-zero elements in each row in the quadrant is less than 4 is quite high. Hence the number of times the SIMD registers have to be loaded and padded is reduced. These results were compared with the results obtained from implementing SpMV with regular CSR format with SIMD vectorization. Fig. 5 shows the graph of the speed-up obtained by the QCSR format. The graph clearly shows that the QCSR format when implemented with SIMD is more efficient than that of the CSR format with SIMD.

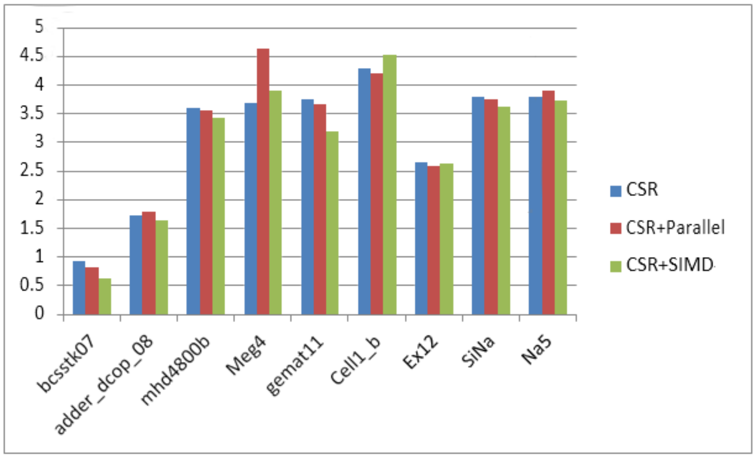

Fig. 4 Computing time for optimization schemes using CSR.

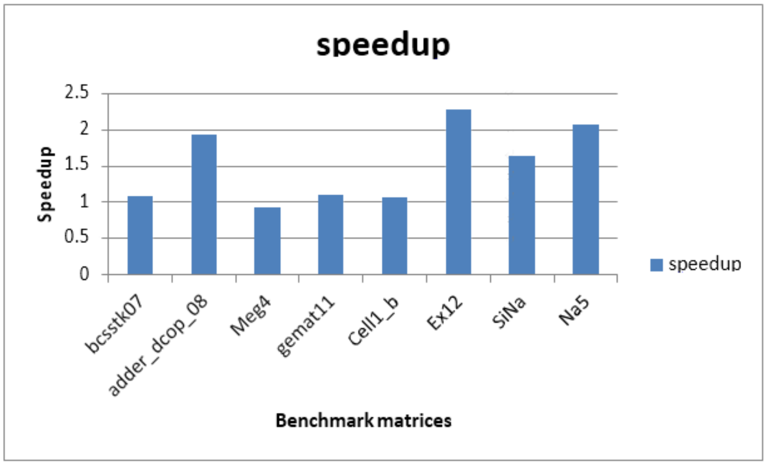

Fig. 5 Computing time for optimization schemes using CSR.

6 Conclusions

This paper explores the use of the SIMD technique provided in current processors, to accelerate the performance of SpMV using QCSR format. As SIMD units are readily available with the current generation processors, and do not incur additional processing cost unlike GPU programming and threading, it is necessary to avail the feature of this hardware to improve the performance of various applications with reduced power . Results show that the speedup of SpMV QCSR with SIMD is an average of 2, compared to that of SpMV CSR with SIMD, for the Benchmark matrices chosen. It is also apparent that the efficiency of this technique completely depends on the arrangement of the non-zero elements per row.

References

[1] Guiyi Wei, Athanasios V. Vasilakos , Yao Zheng and Naixue Xiong, (2010), "A game-theoretic method of fair resource allocation for cloud computing services", *The Journal of supercomputing*, Springer, Vol. 54, No. 2, pp. 252-269.

[2] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, James Demmel (2009), "Optimization of sparse matrixCvector multiplication on emerging multicore platforms", *Journal Parallel Computing*, Vol. 35, No.3, pp. 178-194.

[3] Shengfei Liu, Yunquan Zhang, Xiangzheng Sun and RongRong Qiu (2009), "Performance evaluation of multithreaded sparse matrix-vector multiplication using openMP," *11 th International Conference on High Performance Computing and Communications.*

[4] Dakuan (2009), "CUI Parallel sparse matrix algorithms - for numerical computing Matrix- vector multiplication", *IIMS Postgraduate Seminar.*

[5] Schubert, G., Hager, G., Fehske, H. and Wellein, G. (2011), "Parallel sparse matrix-vector multiplication as a test case for hybrid MPI+ OpenMP programming". In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), *2011 IEEE International Symposium*, pp. 1751-1758 . IEEE.

[6] M.Krotiewski, M.Dabrowski (2010), "Parallel Symmetric sparse matrix- vector product on scalar multi-core CPUs", *Journal Parallel Computing*, Vol. 36, No. 4, pp. 181-198.

[7] Tomas Oberhuber, Atsushi Suzuki and Jan Vacata, (2011), "New Row-grouped CSR format for storing sparse matrices on GPU with implementation in CUDA", *CoRR journal*, Vol abs/1012.2270, 2010.

[8] Jilin Zhang, Enyi Liu, Jian Wan, Yongjian Ren, Miao Yue and Jue Wang,(2013), "Implementing sparse matrix-vector multiplication with QC-

SR on GPU", *International Journal on Applied Mathematics & Information Sciences*, Vol.7, pp. 473-482.

[9] Susana Ladra, Oscar Pedeira, Jose Duato and Nieves R.Brisaboa, (2012), "Exploiting SIMD instructions in current processors to improve classical string algorithms", *Advances in Databases and Information Systems, Lecture Notes in Computer Science*, Vol.7503, pp. 254-267.

[10] Kai Zeng, Erwei Bai and Ge Wang,(2007), "A fast CT reconstruction scheme for a general multi-core PC", *International Journal of Biomedical Imaging*, Vol. 2007 No. 1, pp. 1-1.

[11] Amir Fijani, Fouzhan Hosseini,(2011), "Image processing applications on a low power highly parallel SIMD architecture", *AERO '11 Proceedings of the 2011 IEEE Aerospace Conference*, pp. 1-12.

[12] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy and S. A. Jarvis, (2013), "Exploring SIMD for molecular dynamics, using intel ® xeon®processors and intel® xeon phi TM coprocessors", *IPDPS '13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 1085-1097.

[13] Martin Kong, Richard Veras and Kevin Stock (2013), "When polyhedral transformations meet SIMD Code Generation", *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pp. 127-138.

[14] Libo Huang, Zhiying Wang,Nong Xiao, Yongweng Wang and Qiang Dou, (2013), "Dynamic streamization model execution for SIMD engines on multicore architectures", *IEEE Transactions On Computer -Aided Design Of Integrated Circuits and Systems*, Vol. 32, No. 11.

[15] NaserSedaghati, Renji Thomas, Louis Pouchet, Radu Teodorescu and P. Sadayappan,(2011), "StVEC: A vector instruction extension for high performance stencil computation", *PACT '11 Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, pp. 276-287.

[16] Neil G. Dickson, Kamran Karimi, FirasHamze (2011), "Importance of explicit vectorization for CPU and GPU software performance", *Journal of Computational Physics*, Vol. 230, No. 13, pp.5383-5398.

[17] Ji-Lin Zhang, Li Zhuang, Jian Wan, Xiang-Hua Xu, Cong-Feng Jiang and Yong-Jian Ren, (2011), "Combine optimized sparse matrix-vector multiplication for CSR format", *Proceeding CHINAGRID '11 Proceedings of the 2011 Sixth Annual ChinaGrid Conference*, pp. 124-129.

[18] Kai Zhang, Shuming Chen, Yaohua Wang and Jianghua Wan,(2013), "Breaking the performance bottleneck of sparse matrix vector multiplication on SIMD processors", *IEICE Electronics Express*, Vol. 10, No.9 pp. 1-7.

[19] NazliGoharian, Ankit Jain and Qian Sun (2012), "Comparative analysis of sparse matrix algorithms for information retrieval", http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.8914.

[20] Roland Leiba, Sebastian Hack and Ingowala,(2012), "Extending a C-like language for portable SIMD programming", *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pp. 65-74.

[21] Nathan Bell and Michael Garland, (2008), *Efficient Sparse Matrix-Vector Multiplication on CUDA*, NVIDIA Technical Report NVR-2008-004.

[22] Ivan Simecek, (2009). "Sparse matrix computations using the quadtree storage format", *SYNASC '09 Proceedings of the 2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 168-173.

[23] I.Simecek, D.Langer, P.Tvrdik, (2012), "Minimal quadtree format for compression of sparse matrices storage", *SYNASC '12 Proceedings of the 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 359-364.

[24] T.Davis (1997), "The university of florida sparse matrix collection" [EB/OL], http://www.cise.ufl.edu/research/sparse/matrices.

Corresponding author
J.Saira Banu can be contacted at: jsairabanu@vit.ac.in