Journal of Theoretical and Applied Information Technology
31 August 2012. Vol. 42 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OPTIMIZING SPARSE MATRIX-VECTOR MULTIPLICATION BASED ON GPU

[1, 2] MENGJIA YIN, [2] TAO ZHANG, [1, 3] XIANBIN XU, [1] JIN HU, [1, 4] SHUIBING HE

[1] School of Computer, Wuhan University, Wuhan 430074, Hubei, China

[2] School of Computer and Information Science, Hubei engineering University, Xiaogan 43200, Hubei, China

[3] School of Computer Science, Wuhan Donghu University, Wuhan 430074, Hubei, China

[4] Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 10000, China

ABSTRACT

In recent years, Graphics Processing Units(GPUs) have attracted the attention of many application developers as powerful massively parallel system. Computer Unified Device Architecture (CUDA) as a general purpose parallel computing architecture makes GPUs an appealing choice to solve many complex computational problems in a more efficient way. Sparse Matrix-vector Multiplication(SpMV) algorithm is one of the most important scientific computing kernel algorithms. In this paper, we proposed new parallelization algorithms that CSR-M based on CSR format and ELLPACK-R based on ELLPACK format, which are realized the parallelism kernel on GPU with CUDA. We discussed implementing optimizing SpMV on GPUs using CUDA programming model, the optimization strategies including: mapping thread, mergering access, reusing data, avoiding branch, optimization thread block. The experiment results showed the proposed optimization strategies can improve performance, memory bandwidth and reduce the execution time of kernel.

Keywords: *Sparse Matrix-vector Multiplication, Computer Unified Device Architecture, Graphics Processing Unit*

1. INTRODUCTION

With the rapid growth of computing complexity and data, general CPU computing power has failed to meet its needs. The speed of Graphics Processing Unit (GPU) development is more than Moore's Law, and the computing performance, memory bandwidth is far more than the development speed of the CPU. As modern GPUs have become increasingly powerful, inexpensive and relatively easier to program through high level API functions, they are increasingly being used for nongraphic or general purpose applications (called GPGPU computing).

Sparse Matrix-vector Multiplication (SpMV) operation is widely used in solving large-scale linear system and solving matrix eigenvalues problems [1], especially in iterative method, it is a key step that influences the computing performance. SpMV is a typical of memory bottleneck operation, namely the rate of computing and memory is low, ALU is seriously unsaturated, and it is difficult to achieve the throughput of high floating-point operations. SpMV has the nature of parallelism, how to use modern multi-processor platform research the parallelism of SpMV is one of the feasible direction to improve performance.

According to the deficiency of the traditional parallel strategies, we also proposed new parallelization algorithms that CSR-M based on CSR format and ELLPACK-R based on ELLPACK format, and realized the parallelism kernel in GPU with CUDA. Then we propose the more efficient performance optimization strategies: mapping thread, mergering access, reusing data, avoiding branch, optimization thread block, which can be realized based on CSR format and ELLPACK format. The experiment results showed the proposed optimization strategies can be improved performance, memory bandwidth and reduce the execution time of kernel.

The rest of this paper is organized as follows: section II introduces related work, improved Sparse Matrix format is detailed in section III, parallel computing for SpMV model based on GPU is proposed in section IV, the optimization strategy is proposed in section V, section VI contains our results and evaluation, conclusion is shown in section VII.

Journal of Theoretical and Applied Information Technology
31 August 2012. Vol. 42 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. RELATED WORK

2.1 GPU PROGRAMMING WITH CUDA

CUDA is a parallel computing architecture developed by NVIDIA Corporation [2], and allows writing and running general-purpose applications on the NVIDIA GPU's. CUDA uses threads for parallel execution, and GPU allows thousands of threads for parallel execution at the same time.

On GPU, there is a hierarchy of memory architecture to program on it, we propose the memories in our implementation: registers, shared memory, global memory, constant memory, texture memory. In the memory architecture, the fastest memories are the shared memories and registers. The other memories are all located on the GPUs main RAM. The constant memory is favourable when multiple processor cores load the same value from cache. Texture cache has higher latency but it has a better acceleration ratio for accessing large amount of data and non-aligned accessing. The memory architecture of GPU is described in Figure 1.To gain better performance, we must manage the shared memory, registers, and global memory usage.

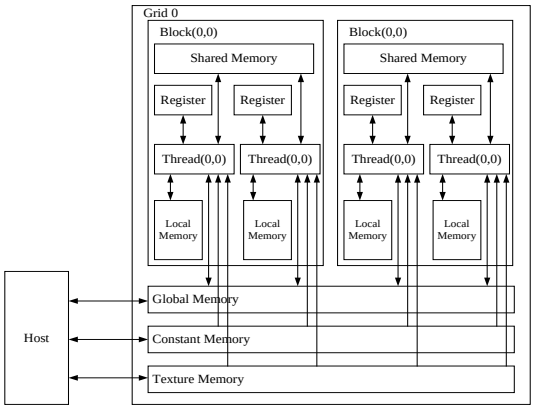

Figure 1: Memory architecture of CUDA

2.2 SPARSE MATRIX-VECTOR MULTIPLICATION

The bottleneck problems of memory are those algorithms that each floating point operations need to multiple-access memory, SpMV is one kind of this algorithm [3]. In the past twenty years, there have been a lot of work for the optimization of the SpMV algorithm, from the point of view of the memory ,optimization is mainly to improve the computation performance [2] [4] [5], which most of the optimization work is focused on CPU that generalization system structure [6]. But the optimization strategy can't be directly used in GPU. GPU is massively parallel systems, it has multi-stage storage system structure. In order to play the advantages of GPU memory high bandwidth, we need to accord to the characteristics to design different optimization strategy.

In [7], Nathan Bell and Michael Garland provided data structures and algorithms for SpMV that are efficiently implemented on CUDA platform for the fine-grained parallel architecture of the GPU. They emphasized memory bandwidth efficiency and compact storage formats when given the memory-bound nature of SpMV. They also developed methods to exploit several common forms of matrix structure while offering alternatives which accommodate greater irregularity.

In [8], with indirect and irregular memory accesses resulting in more memory access per floating point operation, Baskraran proposed optimizations to effectively develop a high-performance SpMV kernel on NVIDIA GPUs. The optimizations including: exploiting synchronization -free parallelism, optimized thread mapping based on the affinity towards optimal memory access pattern, optimized off-chip memory access to tolerate the high access latency, exploiting data reuse.

Based on the above, this paper emphasize its optimization strategy in the process of SpMV algorithm on GPU, the optimization strategy is aimed at the system structure of the GPU, and consider the GPU complex storage management and the mapping optimization between threads.

2.3 SPARSE MATRIX FORMAT

In scientific computing, SpMV has been proven to be a special important of numerical algorithm [9], it has the characteristics of high intensity calculation, high parallel degree and simply control, so matrix calculation is very suitable to GPU for parallel computing. How to play the powerful computing ability of GPU in sparse matrix vector algorithm is need to deal with.

Sparse matrix has several storage formats such as ELLPACK, COO, CSR, and Hybrid and so on. These storage formats are described detailed in [10]. Each format is different in storage requirements, calculation characteristics, access and operation of the matrix element method. Different storage formats are determined by the sparse matrix mode, that is, the distribution of non-zero elements in the matrix. In this paper, we discussed optimization strategies based on CSR and ELLPACK storage format to suit the GPU architecture.

CSR format is the more popular storage format[10][11],it is a line of compressed format

Journal of Theoretical and Applied Information Technology
31 August 2012. Vol. 42 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

which alter storing two-dimensional array of sparse matrix into three one-dimensional arrays: A, Col_Idx, Row_Ptr. Scan follow the line width for the sparse matrix, and will stored the zero elements in array A; An array of Col_Idx stored column index of non-zero elements in array A corresponding location in the original matrix; An array of Row_Ptr stored an index of every row in the first non-zero elements in an array of A in primitive sparse matrix. For M * N matrix, the length of Row_Ptr array is M+1, the offset of i-th row stored in Row_Ptr [i], the last Row_Ptr [M] in sparse matrix stored the total number of non-zero elements. We give an example of 5 * 4 sparse matrix, as shown in Figure 2, Figure 3, how to use the CSR storage formats to show the original sparse matrix.

0	2	0	4	0
1	2	3	0	0
0	1	0	0	0
0	0	0	1	1

Figure 2: 5 * 4 sparse matrix

A =	2	4	1	2	3	1	1	1
Col_Idx =	1	3	0	1	2	1	3	4
Row_Ptr =	0	2	5	6	8			

Figure 3: CSR storage format of sparse matrix

ELLPACK format is not general storage format, because this kind of storage format has certain requirement to the sparse matrix, which demands the number of non-zero element has little change. ELLPACK format has only two arrays, which respectively are: A, Col_Idx.

The format structure mode of ELLPACK is similar to CSR, the non-zero elements are moved to the left side of matrix, and the zero elements are moved to the right side of the matrix, the Figure 4 is shown in CSR storage format of sparse matrix.

2	4	0	0	0
1	2	3	0	0
1	0	0	0	0
1	1	0	0	0

Figure 4: CSR storage format of sparse matrix

In determining the max length of row, the zero elements that smaller than max will be discarded. Finally, according to the column-wise scan the results matrix, all elements of the results matrix are stored in the array A, An array of Col_Idx stored column index of non-zero elements in array A corresponding location in the original matrix. Here, we are not consider X in an array of Col_Idx, because the zero elements filled in array A is not important, is only convenient to the same addressing way to element, Figure 5 and Figure 6 is original sparse matrix 2 D ELLPACK array and 1 D ELLPACK array.

A =	2	4	0	Col_Idx =	1	3	X
	1	2	3		0	1	2
	1	0	0		1	X	X
	1	1	0		3	4	X

Figure 5: 2D ELLPACK array

A =	2	1	1	1	4	2	0	1	0	3	0	0
Col_Idx =	1	0	1	3	3	1	X	4	x	2	X	X

Figure 6: 1D ELLPACK array

For the M * N sparse matrix that each line at most K nonzero elements, ELLPACK format need to use dense array A[M*K] to store, when the number of nonzero elements is less than K, we filling with zero elements.

3. IMPROVED SPARSE MATRIX FORMAT

3.1 IMPROVED CSR FORMAT

CSR-M format is proposed on the basis of CSR format, which is improved based on the characteristics of unique hierarchy structure of memory in GPU. At the same time, it is also based on general purpose, so CSR-M format may be considered to be general storage formats. CSR-M format is composed of three array, they are respectively: A, Col_x, Row_Ptr.

In CSR format, an array of Col_Idx stored column index of non-zero elements in each row corresponding location in the original matrix. But in this paper, array of Col_Idx no longer stored index value of column, but stored vector X elements that corresponding index position on the nonzero elements multiplication, the name of array modification of Col_x.

After modification the CSR format, only need copy array Col_x to the GPU, no longer need copy vector X from CPU to GPU. This way reduces data

Journal of Theoretical and Applied Information Technology
31 August 2012. Vol. 42 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

transfer between CPU and GPU, at the same time, it also reduces the kernel access the global memory. The Global memory can provide high memory bandwidth, but memory access latency is very high, this improvement reduced access time latency of global memory and improved computing performance of the sparse matrix vector multiplication. The optimization of the CSR storage formats in the Figure 7. We hypothesis input vector X is [1-5].

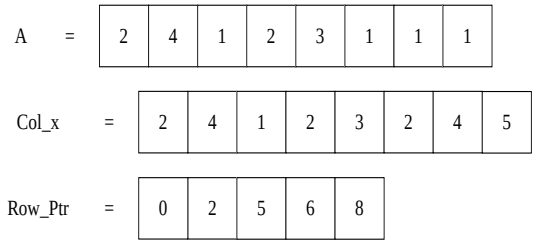

Figure 7: CSR-M storage format of sparse matrix

3.2 IMPROVED ELLPACK FORMAT

ELLPACK-R format is based on ELLPACK format, it is designed to SPMV in GPU. Its structure is not different in ELLPACK, it increased the array of RL, the size of the array RL is the number of lines N in matrix, array RL is stored the number of nonzero elements in each line, figure 8 and figure 9 is original sparse matrix 2 D ELLPACK array and 1 D ELLPACK array.

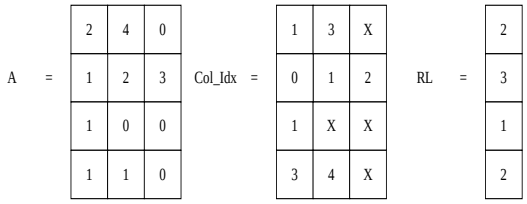

Figure 8: 2D ELLPACK array

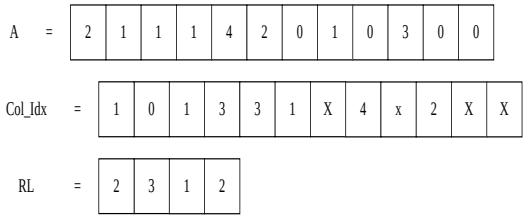

Figure 8: 1D ELLPACK array

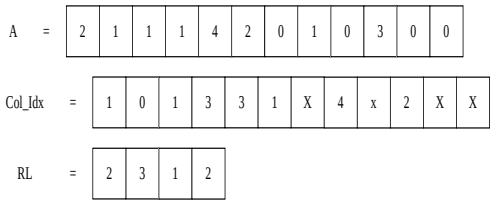

Figure 9: 1D ELLPACK array

4. PARALLEL COMPUTING FOR SPMV MODEL BASED ON GPU

4.1 CPU REALIZED SPMV

The serial algorithm based on the CSR-M format is shown in table I, this algorithm is realized its parallelization in multiple processors, parallelism is realized in outer loop, so the single processors is responsible for computing the row of matrix.

Table I
Serial algorithm based on the csr-m format

```
{ for i=0 to rows
{y(i)=0;
 for j= Row_Ptr(i) to (Row_Ptr(i+1)-1)
  {y(i)=y(i)+A(j-1)*x(Col_Idx(j-1));}
}
}
```

The SPMV serial algorithm based on ELLPACK-R format is shown in table II. This algorithm in multiprocessors realized parallelization, parallelism realization in inner circulation.

Table II
Serial algorithm based on the ELLPACK-R format

```
{the length of RL array is same to the number of line
MAX=The maximum row length;
for i=0 to rows
{y(i)=0;}
for i=0 to MAX
 {for j=0 to Rows
   {if i<RL(j)
y(j)=y(j)+A(j+i*Rows)*x(Col_Idx(j+i*Rows));} }}
```

4.2 GPU REALIZED SPMV

Nathan Bell and Michael Garlandy proposed the parallelism SPMV kernel in NVDIA GPU use CUDA [7], the kernel covered a variety of sparse matrix storage formats. Here we only introduced SPMV parallelization kernel algorithm of CSR-M format and ELLPACK-R format.

The SPMV parallelization kernel algorithm based on CSR-M format in GPU is shown in table III. Here the method of parallelism is: use a Warp threads to responsible for computing non-zero elements in sparse matrix, don't need to fill zero elements to align, the intermediate results put on sharing memory, and then accumulate the intermediate results through reduction summation, finally through thread 0 to get the final results.

The SPMV parallelization kernel algorithm based on ELLPACK-R format in GPU is shown in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

table IV. Here the method of the parallelism is: use a thread to responsible for computing nonzero elements of one line in matrix, align ELLPACK-R array with filling the zero elements, the ELLPACK-R kernel is not need to continuous access to x vector.

Table III
Spmv kernel for the csr-m sparse matrix

```
{ _global_void spmv_csr-m_Kernel(const int num_rows,
const int * Row_Ptr, const int *Col_Idx, const Float * A,
const Float * x, Float* y)
{ __shared__ Float vals[];
    int thread_id=blockDim.x*blockIdx.x+threadIdx.x;
    int Warp_id= thread_id/ 32;
    int lane = thread_id & (32-1);
  int row = Warp_id;
 if(row < num_rows)
{ int row_start = Row_Ptr[row], row_end =
Row_Ptr[row+1];
    vals [threadIdx.x] =0;
    for(int j = row_start +lane; j < row_end; j +=32)
     {vals [threadIdx.x] += data[j] * x[indices[j]];}
if (lane<16) vals [threadIdx.x] += vals [threadIdx.x +
16];
            if (lane<8) vals [threadIdx.x] +=
sdata[threadIdx.x + 8 ];
            if (lane<4) vals [threadIdx.x] += vals
[threadIdx.x + 4 ];
            if (lane<2) vals [threadIdx.x] += vals
[threadIdx.x + 2 ];
            if (tlane<1) vals [threadIdx.x] += vals
[threadIdx.x + 1 ];
      if (lane == 0)  y[row] += vals [threadIdx.x];     }
}
```

Table IV
Vector spmv kernel for the ellpack-r sparse matrix

```
{ _global_void spmv_ellpack-r_Kernel(const int
num_rows, const int num_cols, const int
*Col_Idx, const int *RL, const Float * A, const
Float * x, Float* y)
{ int row= thread block Dim.x* thread block
Idx.x+threadIdx.x;
 If(row<num_rows)
{float dot=0;
  for(int n=0;n<RL[row];n++)
   {int col=Col_Idx[num_rows*n+row];
    float val=A[num_rows*n+row];
    if(val!=0) dot+=val*x[col];
    }
y[row]+=dot;}}}
```

5. OPTIMIZATIONS FOR SPMV MODEL BASED ON GPU

The CUDA programming model greatly simplified the difficulty of using the GPU for general purpose computing, but compared to isomorphic system that only included CPU, it is more complicated to program in the heterogeneous system based on the CPU-GPU, and the program's performance optimization is even more difficult. Generally affect the performance of CUDA program includes the main three factors: memory access latency, load balance and global synchronous spending [12]. These factors in different computing platforms the causes and the corresponding optimization method is not same.

5.1 OPTIMIZATION CSR-M SPMV FORMAT

CSR-Vector used one warp calculating elements of one line according to the circle. In the process of calculation, in order to get the output vector, we reduce summing in shared memory. However, if the number of nonzero elements in the row is less than 32, the computing performance of CSR-Vector will drop. When the number of nonzero elements in the row is more than 32, the computing performance will be good. For the various shortcomings of the CSR-Vector kernel, we propose the following optimization strategy:

(1) Thread mapping

In [7], taken the method that calculation of each warp corresponds to each element in the output vector y in the kernel, the natural features of synchronization improve the computation performance. The shortcoming is when the number of non-zero elements of each row in sparse matrix is less (each row only the individual non-zero elements), it will waste of computing resources. In this paper, we use the half-warps as a unit, set the number of threads in one line mapping as 16.

(2) Merger access

In the CSR-M format of the SPMV, the array A sequentially stored the nonzero elements of sparse matrix according to line way, so the thread access the elements of each row also meet the requirements of the merger memory access. If the number of nonzero elements in each line is more than 16, memory access of one line will be splitted into multiple memory access, each access are meet to merge access

(3) Data reusing

In the CSR-M format of the SPMV, we put the intermediate results into the shared memory, and finally summing the intermediate results in the shared memory. In this paper, we use the method of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

reduction summing, when the block address does not conflict of, it will very high-speed. The final output vector y is written to global memory.

(4) Avoiding branch

When the total number of rows modulo for the rows of each thread block, the result is not equal to 0, in the last thread block, the elements that calculated will be filled with 0.Thus reducing the judgment of branch in kernel, and improving computational efficiency. When reducing summing, in order to avoid the judgment of branch, the size of array that stored intermediate results is set to multiple of 16.

(5) Optimization thread block

Thread block is set to 256 threads, and minimize the occupation of the registers and shared memory

5.2 IMPROVE CSR- SPMV OPTIMIZATION ALGORITHM

The achievement of Kernel first needs several threads that responsible for computing an element of the output vector. When the number of non-zero elements that one line contains is less, or is not multiple of 16, this strategy will cause wasting the thread to calculate the resources. In this paper, we propose a new calculation method: array A in CSR sparse matrix is divided into certain length fragments, the length of the fragment is an integer multiple of the number of threads in the thread blocks, a thread block calculate element of an array fragment. The intermediate results stored in shared memory, and finally through accumulated calculation the intermediate results to complete the output element y [13]. This method is equally distributed computing tasks, and can effectively improve the operation efficiency.

Because there is difference in the number of nonzero elements of sparse matrix each row, CSR-M-SpMV kernel is difficult to average assign computing tasks to each thread, and cause computing resources free. To solve this problem, this paper takes the method that each thread block calculates the 1024 nonzero elements, the last fragment is filled with 0, as shown in Figure 10.

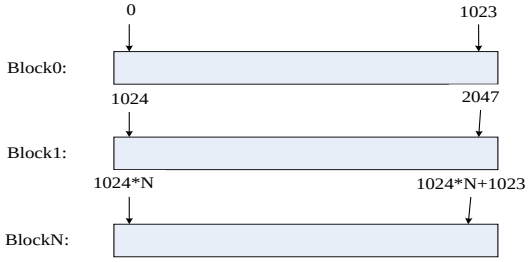

Figure 10: Each thread block calculates the 1024 nonzero elements

On the base of the CSR data structure, we added an int2 type array Bound, the length of array Bound is the number of array fragments that are divided (the number of thread block). Bound [i] is correspond to thread block that index is i, the member of x stored row number where the first element corresponding thread block ,the member of y stored row number where tail element. This paper only generate Bound array through a simple judgment on each element values in Row_Ptr, as shown in Figure 11.

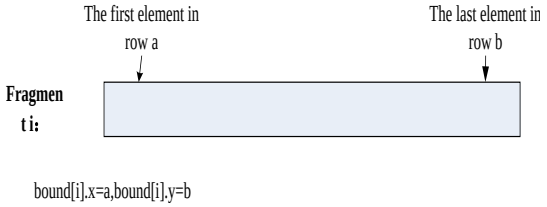

Figure 11: Generate Bound array

The above process by the two Kernel function：the first step calculates the incomplete result and auxiliary vector result_aid; the second step merged the result_aid into the result, so obtain the final result.

Table V
The first kenerl

1. Calculate the product of 1024 elements and the corresponding vector elements, saved to the shared memory. 2. According to the boundary row number that Bound recorded, read the value of rpos. Assume the fragment contains 100 lines, then read the adjacent 100 Half-walf thread . If the number of line is more than the number of Half-Warp, through the cycle solution. 3.Assuming that this fragment contains 100 lines, the first 100 Half-Warp product accumulating corresponding single element ,then the result will put into the registers. The first and the last row that corresponding Half-Warp will accumulate results in result_aid, the rest of the corresponding Half-Warp will write the result. When the number of line is more than the number of Half-Warp, we also used the method of cycle.

Table VI
The second kernel

```
1.thread i corresponding Bound[i]
2.if(Bound[i].x== Bound[i-1].y) thread is not work;
else if(Bound[i].x!= Bound[i-1].y)
{thread work;
  while(Bound[i].y== Bound[i+1].x ) i++;  }
```

Journal of Theoretical and Applied Information Technology
31 August 2012. Vol. 42 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5.3. OPTIMIZATION ELLPACK-R SPMV ALGORITHM

In this paper, we realized ELLPACK-SPMV parallelization kernel on GPU, with a thread responsible for calculating one element in vector y, this thread is marked as x = i.The ELLPACK-SPMV kernel is suitable for calculating the sparse matrix of regularization data structure, which is similar to the dense matrix, namely there is not much difference among the number of zero elements in each line.

ELLPACK-R format is modified for realize parallelize SPMV kernel on GPU based on ELLPACK format. Here we mainly introduced optimization strategy of ELLPACK-R-SPMV kernel, the optimization strategy is proposed aimed at the shortcomings of ELLPACK-SPMV kernel.

(1) Optimization thread

According to the difference in thread mapping in each row, we can realize SPMV kernel based on ELLPACK-R format. When use T threads to calculate element y[i], we need to access elements in row i, the kernel is called ELLR-T. In this way, elements in row i are divided into T subsets. In order to calculate the output vector element y[i], T threads need RL[i] times internal iteration, the results in each thread will put the shared memory. Finally, in order to get element y[i], we need to reduction summing the T results in share memory. To achieve the best performance calculation, for each different sparse matrix, the values of parameters T are often not the same.

(2) The merger visiting

Reading all of the elements in array A, Col_Idx and RL is satisfied with the requirements of merger visiting and memory alignment, this is because ELLPACK-R format stored element use wide column, filling zero elements of each line to make the number of elements is just multiple of 16. So we can make as high as possible memory bandwidth on GPU.

(3) Avoiding branch

When executing ELLPACK-R-SPMV kernel, the threads that belong to the same warp will not enter branch. The code does not contain flow instructions, the flow instruction in the warp can produce serialization, because each thread is carry out the same cycle, but the number of iterations is not identical. When the circulation stop, thread will stop; however, those threads that not end of the circulation will continue.

(4) Optimization thread block

By optimizing the size of the thread block, we can obtain a higher share of the SM when the size of thread block in ELLR-T kernel is 128.

6. EXPERIMENT RESULTS

6.1 EXPERIMENT PLATFORM

We experimentally evaluated our system using NVIDIA Tesla C1060, connected to Windows 7 system. The development environment is VS2010 IDE. The CUDA kernels were complied using NVIDIA CUDA Complier (nvcc) to generate the device code that was then launched from the GPU. The host programs were complied using the C language. We used CUDA used version 4.0 for our experiment. The architectural configurations are presented in Table VII.

Table VII
Test platform specifications

matrix	row（ column）	The number of non-zero	The number of non-zero each line
Protein	36,417	4344765	119.3
PEM/Spheres	83,334	6010480	72.1
FEM/Cantilever	62,451	4007383	64.1
Economics	206,500	1273389	6.1
Epidemiology	525,825	2100225	3.9
FEM/Accelerator	121,192	2624331	21.6

We use 6 sparse matrices from the sparse collection described in [14]. The benchmark program in NVIDIA SpMV library is also used the matrix as a test matrix, the selected sparse matrices represent a wide variety of real applications. Every matrix has properties of number of rows, columns, and elements of matrix, NNZ represents the zero number of elements. The properties of 6 matrices are showed in Table VIII.

Table VIII
Test matrix sets

GPU	NVIDIA Tesla C1060
CPU	Intel(R) core(TM) i7 920
OS	Windows 7
CUDA	CUDA 4.0
IDE	Microsoft Visual Studio 2010

6.2 PERFORMANCE MEASUREMENTS

(1) Computing performance

The computing performance of SpMV kernel is measured by GFLOPs (giga floating point operations per second). Floating point operand is equal to the number of NNZ element multiplied by 2. Computational performance is equal to the floating-point operand divided by run time of kernel, it is shown as formula 1:

Journal of Theoretical and Applied Information Technology
31 August 2012. Vol. 42 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$PF = \frac{2FP * NNZ}{T * 10^9} \quad (1)$$

The running time of kernel is represented by T, which is calling SpMV kernel execution many times, and then takes the average of time, the unit is Ms (milliseconds). Here the FP is floating-point operand.

(2) Memory bandwidth

In order to calculate the memory bandwidth, we need to calculate the number of memory transactions, and then divided by the kernel execution time T. The unit of memory bandwidth is GBytes, the unit of memory transactions is Byte, and the number of memory transactions depends on the realization of the algorithm. Of course, this paper is realized by the single precision floating-point values, which means that every matrix elements need 4 Bytes, integer index also need 4 Bytes, matrix A, input vector x and output vector y are floating point, the index is an integer.

The formula of calculation memory bandwidth based on CSR format is shown as formula 2, in CSR format, the length of array of A, Col_Idx and Row_Ptr is respectively NNZ, NNZ and Rows + 1, the length of input vector x is also equal to NNZ.

$$BW_{CSR} = \frac{3 * 4Bytes * NNZ + 4Bytes * (Rows * 3 + 1)}{T * 10^6} \quad (2)$$

The formula of calculation memory bandwidth based on CSR-M format is as shown as formula 3, in CSR-M format, the array of column index is no longer store index, but the value of the storage input vector x.

$$BW_{CSR-M} = \frac{2 * 4Bytes * NNZ + 4Bytes * (Rows * 3 + 1)}{T * 10^6} \quad (3)$$

The formula of calculation memory bandwidth based on ELLPACK format is as shown as formula 4, the array of ELLPACK data structure has only two A and Col_Idx.

$$BW_{ELLPACK} = \frac{2 * 4Bytes * NNZ + 2 * 4Bytes * Rows}{T * 10^6} \quad (4)$$

The formula of calculation memory bandwidth based on ELLPACK-R format is as shown as formula 5, the array of ELLPACK data structure has only two A and Col_Idx and RL.

$$BW_{ELLPACK-R} = \frac{2 * 4Bytes * NNZ + 3 * 4Bytes * Rows}{T * 10^6} \quad (5)$$

(3) Running time

TGPU is the running time of parallel kernel on GPU. The running time of kernel does not include the preprocessing time of the sparse matrix format, nor including the time of the transfer and copy matrix data between the CPU memory and GPU memory.

6. 3. EXPERIMENT RESULT

According to the difference in the SpMV sparse matrix CSR format, we respectively marked SpMV as CSR-R-GPU, CSR-O-GPU, CSR-B-GPU, and CSR-M-GPU. CSR-R-GPU is not optimized SpMV kernel. CSR-O-GPU is realized by Optimization strategy, CSR-M-GPU is realized based on CSR-M format. CSR-B-GPU is realized by a new algorithm that proposed in this paper, which is introduced a new data structure contains Bound.

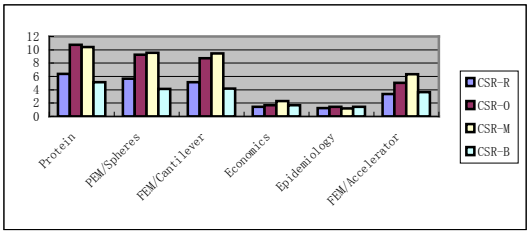

Figure 12: Computing performance based on CSR SpMV kernel on GPU

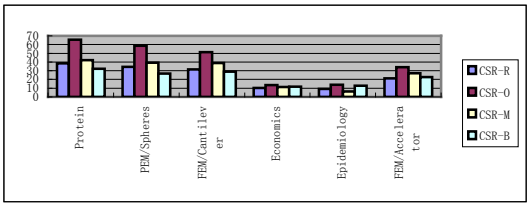

Figure 13: Memory bandwidth based on CSR SpMV kernel on GPU

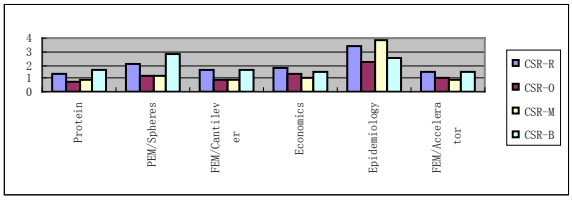

Figure 14: Running time based on CSR SpMV kernel on GPU

Through the analysis, we can get the following conclusion:

For CSR-O-GPU kernel, the execution time, memory bandwidth and computing performance is obviously higher than CSR-R-GPU, so the optimization strategy is effective.

For CSR-M-GPU kernel, we can get better performance compared with the CSR-O-GPU, especially in the matrix PEM/Spheres, Economics, FEM/Cantilever and FEM/Accelerator. But in memory bandwidth, the CSR-M-GPU kernel is lower than the CSR-O-GPU kernel. Its performance is higher than CSR-R-GPU kernel.

For CSR-B-GPU kernel, the performance in the matrix Protein, PEM/Spheres, FEM/Cantilever and FEM/Accelerator is obviously lower than other kernel. But, in Economics, Epidemiology matrix, the performance of CSR-B-GPU kernel is higher

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

than other kernel. When the average number of nonzero elements in each line is little, the performance is higher than other kernel. But when the average number of nonzero elements in each line is more, its performance is not ideal. One of the reasons is the kernel use many shared memory, so the optimization algorithm also needs to improve.

The NON-CSR storage formats that realization of SPMV on GPU have ELLPACK, ELLPACK-R format, the corresponding kernel performance results marked as ELLPACK-GPU, ELLPACK-R-GPU. For ELLPACK-R-GPU kernel, because use the optimization strategy, its performance better than ELLPACK-GPU. It is different from other formats kernel, the sparser matrix, the better the performance of calculation.

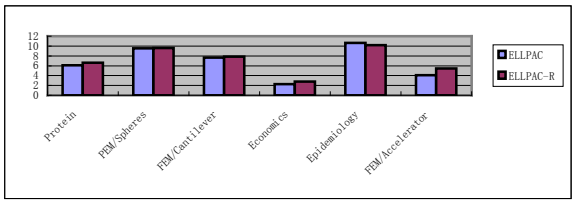

Figure 15: Computing performance based on ELLPACK SpMV kernel on GPU

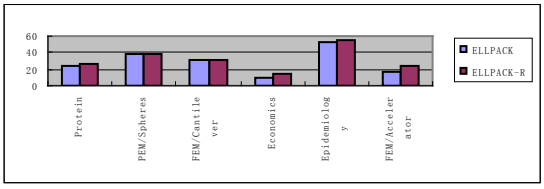

Figure 16: Memory bandwidth based on ELLPACK SpMV kernel on GPU

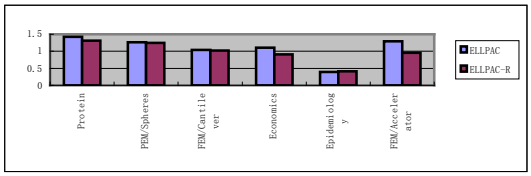

Figure 17: Running time based on ELLPACK SpMV kernel on GPU

7. CONCLUSION

In this paper, we proposed new parallelization algorithms that CSR-M based on CSR format and ELLPACK-R based on ELLPACK format, which realized the parallelism kernel on GPU with CUDA. We also proposed optimizations of sparse matrix vector multiplication on NVIDIA GPUs using CUDA programming model. The optimization strategies including: optimization thread mapping, merger access the global memory, data reusing in the share memory, through the filling zero elements to achieve aligned so as to avoiding branch, and optimization thread block to improve SM processor share. The experiment results showed the proposed optimization strategy can be used on CSR and ELLPACK format, the strategy can be improved performance, memory bandwidth and reduce the execution time of kernel.

ACKNOWLEDGEMENTS

This work is supported by Fundamental Research Funds for the Central Universities (Grant No.3101012), and by Key Laboratory of High Confidence Software Technologies Program (Grant No.HCST201104).

REFRENCES:

[1] M. Shereshevsky, B. Cukic, J. Crowel et al., Software Aging and Multifractality of Memory Resources, *Proceedings of DSN 2003*, pp. 721-730, 2003.

[2] A.J.C.Bik and H.A.G.Wijshoff, "Automatic Data Structure Selection And Transformation For Sparse Matrix Computations", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No.2, pp.109-126, 1996

[3] F. V'azquez, E. M. Garz'on, J. J. Fern'andez, A Matrix Approach To Tomograhpic Reconstruction And Its Implementation On Gpus, *Journal of Structural Biology*, Vol. 170, No. 1, pp.146-151, 2010

[4] K. Kourtis, G. Goumas, N. Koziris, "Optimizing sparse matrix-vector multiplication using index and value compression", *Proceedings of the 5th conference on Computing frontiers*, pp. 87-96, 1999.

[5] B.C. Lee et al., "Performance Model for Evaluation and Automatic Tuning of Symmetric Sparse Matrix-Vector Multiply", *International Conference on Parallel Processing*, pp. 169-176, 2004.

[6] K. Fatahalian, J. Sugerman, P. Hanrahan, "Understanding the efficiency of GPU algorithms for matrix-matrix multiplications", *Proceedings of the ACM SIGGRAPH /EUROGRAPHICS conference on Graphics hardware*, pp.133-137 2004.

[7] N. Bell and M. Garland, Efficient Sparse Matrix-Vector Multiplication on CUDA, *NVIDIA Technical Report NVR-20080004*, NVIDIA Corporation, 2008

[8] M. M. BASKARAN, R. BORDAWEKAR, Optimizing Sparse Matrix-Vector Multiplication on GPUs, *IBM Research Report RC24704*, April 2009.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[9] R.Shahnaz, A. Usman, I. R. Chugtai, "Review of Storage Techniques for Sparse Matrices", *Proceedings of the 9th International Multitopic Conference*, pp.1-7, 2005.

[10] R. Barrett et al., Templates for the solution of Linear Systems: Building blocks for Iterative Methods, SIAM Press, Philadelphia, 1994.

[11] R. Shahnaz, A. Usman, I. R. Chugtai, "Implementation and Evaluation of Parallel Sparse Matrix-Vector Products on Distributed Memory Parallel Computers", *Proceedings of 2006 IEEE International Conference on Cluster Computing*, 2006.

[12] B. Chen, "Research on Performance Optimization of Heterogeneous Platform based on CPU-GPU and Multicore Parallel Programming Model", Mster Thesis, University of Science and Technology of China, pp.30-45, 2011

[13] H.Chen, "Parallel Technology For Implementing Sparse Matrix Vector On GPU", Mster Thesis, School of Computer, Wuhan University, China, 2012,

[14] S. Williams, et al., "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms", *Parallel Computing*, Vol. 35, No. 3, pp.178-194, 2009.