A MMDBM Classifier with CPU and CUDA GPU computing in various sorting procedures

Sivakumar Selvarasu[1], Ganesan Periyanagounder [1], and Sundar Subbiah [2]
[1]Department of Mathematics, Anna University, India
[2]Department of Mathematics, Indian Institute of Technology, India

Abstract: *A decision tree classifier called Mixed Mode Database Miner (MMDBM) which is used to classify large number of datasets with large number of attributes is implemented with different types of sorting techniques (quick sort and radix sort) in both Central Processing Unit computing (CPU) and General-Purpose computing on Graphics Processing Unit (GPGPU) computing and the results are discussed. This classifier is suitable for handling large number of both numerical and categorical attributes. The MMDBM classifier has been implemented in CUDA GPUs and the code is provided. We used the parallelized algorithms of the two sorting techniques on GPU using Compute Unified Device Architecture (CUDA) parallel programming platform developed by NVIDIA Corporation. In this paper, we have discussed an efficient parallel (quick sort and radix sort) sorting procedures on GPGPU computing and compared the results of GPU to the CPU computing. The main result of MMDBM is used to compare the classifier with an existing CPU computing results and GPU computing results. The GPU sorting algorithms provides quick and exact results with less handling time and offers sufficient support in real time applications.*

Keywords: *Classification, Data Mining, CUDA, GPUs, Decision tree, Quick sort, Radix sort.*

Received July 29, 2014; accepted April 12, 2015

1. Introduction

Data mining is a process to extract unknown predictive information from bulky data sets. The data mining tools are used to expect the future trends and performances. They are also used as computerized decision support system. Data mining is also used to discover unknown arrangements in huge data sets [7, 19]. In classification, we are given a set of example records or the input data, called the training data set, with each record containing a number of attributes or features. An attribute can be either a numerical attribute or a categorical attribute. If the value of an attribute belongs to an ordered domain, the attribute is called a numerical attribute (e.g., age, weight, sports, sleep and drink). A categorical attribute, on the other hand, has values from an unordered domain (e.g., sex, Blood Pressure (BP)). One of the categorical attributes is nominated as the classification attribute; its values are called class labels. The class label shows the class to which each record belongs. The objective of classification is to analyze the input data and to develop an exact explanation or model for each class using the features present in the data. Once such a model is raised, future records, which are not in the training set, can be classified using the model. The objective of classification is to use the training dataset to build a model of the class label such that it can be used to classify new data whose class labels are unknown [7]. The decision tree learning algorithm is a very well-known learning model in classification.

Many studies are a source of motivation on improving the performance of decision tree [1, 6, 7, 18]. However, those algorithms are based on a distributed system. The cost of those devices is very high.

The abbreviation for Compute Unified Device Architecture (CUDA) is a parallel computing design developed by NVIDIA Enterprise [4, 11]. Associated to traditional GPGPU methods, CUDA has many advantages, such as distributed reads, common memory, quicker downloads and read backs to or after the GPU, and full support for integer and bitwise tasks. These features create CUDA an efficient parallel computing architecture, which can easily drain the calculating capacity of recent GPUs. A full introduction to programming with CUDA can be found in NVIDIA Corporation, 2008 [3, 17].

The general-purpose computing on graphics processing units GPGPU has subsequently developed the extremely parallelization and powerful computing capability of float point. Some documents display the computing power of GPUs which can now infinitely outstrip CPU [8, 12, 16]. More and more non-graphic applications which required quantities of computation are employed on GPU. Subsequently GPGPU developed a tendency, NVIDIA affords platform to GPGPU which is called Compute Unified Device Architecture. Various applications and researches of machine learning use CUDA as their GPGPU platform.

In this paper, we conversed an effective parallel all sorting algorithm on GPGPU computing and linked the results of GPU to the CPU computing. A case study of MMDBM is used to compare the classifier with an existing CPU computing results and GPU computing results. The GPU sorting algorithms affords quick and exact results with minimum processing period and provides a good support in real time applications.

2. Related works

We present background to our research work. Particularly, we describe the data classification and a generally used solution for Supervised Learning in Quest (SLIQ) and Mixed Mode Database Miner (MMDBM) classifier algorithm for decision tree learning and also we compare to CPU computing and GPU computing .

2.1. Decision tree classification and Algorithm

The classification of an unidentified input vector is done by travelling the tree from the root node to a terminal node. A record enters the tree at the root node. At the root node, a test is applied to determine which child node the record will come across subsequently. This method is repeated until the record reach the destination at a terminal node. All the records are ending up at a given terminal node of the tree are classified in the same method [1, 6, 9, 14].

Algorithm 1: Decision Tree algorithm

MakeTree (Training Data T)
Partition (T);
BuildTree(Data set S)
If (all records in S are in same class)
return;
for each attribute A
Use best split found to partition S_1 into S_2;
Partition (S_1);
Partition (S_2);

2.2. Splitting points

A splitting point is used to evaluate the "goodness" of the different splits for an attribute. We use the *gini* index, initially proposed in [5, 7, 18, 20], based on our knowledge with SLIQ and Scalable PaRallelizable INndution of decision Trees (SPRINT). If a data set S contains n classes, $gini(S)$ is defined as

$$gini(s) = 1 - \sum p_j^2 \qquad (1)$$

where p_j is the relative frequency of class j *in* S.

If a split divides *S* into two subsets $S1$ *and* $S2$, the index divided data $gini_{split}(S)$ is given by

$$gini_{split}(S) = \frac{n_1}{n} gini(s_1) + \frac{n_2}{n} gini(s_2). \qquad (2)$$

The benefit of this index is that it requires computation requires only at the distribution stage of the class values in each of the partitions.

To discover the best split point for a node, we search each of the node's attribute lists and calculate split based on that attribute. The attribute containing the split point with the lowest value for the *gini* index is then used to split the node. We used two types of attributes (i) Numerical attribute is a binary split of the form $A \le v$, where v is a real number, is used for numeric attributes (e.g. age, weight, sports, drinks). (ii) Categorical attributes If $S(A)$ is the set of possible values of a categorical attribute *A* (e.g. BP, sex).

3. CUDA Architecture

After implementing the MMDBM Classifier with the help of GPU, we compared different types of sorting procedure about CPU computing, GPU computing and CUDA. We start NVIDIA GeForce GT 525M with Fermi based GPU architecture [12, 13]. The NVIDIA GeForce GT 525M is a relatively fast mid-range laptop graphics card and the inheritor to the GeForce GT 425M. It is based on the GF108 core as measure of the Fermi architecture. Consequently, it supports DirectX 11 and OpenGL 4.0. Likened to the GT 425M, core clock rates of the GT 525M have been increased by about 7 percent.

3.1. GF108 architecture

The GF108 core of the GT 525M is connected to the GF100 core makes in the GeFore GTX 480M and offers 96 shaders and a 128 Bit memory bus for DDR3 VRAM. Except for the memory controllers, the GF108 can basically be measured a halved GF106. Hence, the architecture is not directly equivalent to the old GT215 (e.g., GeForce GTS 350M) or GT216 (e.g., GeForce GT 330M) cores. Unlike the GF100, the smaller GF104, GF106, and GF108 cores were not only summarized, but also considerably adjusted. In dissimilarity to the GF100, which was measured for qualified applications, these final chips target the consumer market. They feature more shaders (3x16 instead of 2x16), more texture units (8 instead of 4) and more Special Function Unit (SFU) per Streaming Multi-processor (SM). As there are still only 2 warp schedulers (versus 3 shader groups), Nvidia now uses superscalar execution in order to utilize the higher amount of shaders per SM more efficiently. In theory, the performance per core should be greatly improved over previous generations [10, 11, 12, 13].

CUDA is a general purpose parallel computing architecture containing a new parallel programming model and an instruction set architectures [17]. CUDA is an extension of the C language. A CUDA program mostly contains of CPU code and at least one kernel, i.e. void returning function to be implemented by the

GPU. The key words __global__ qualifier to the kernel function which is called by CPU, we executed the function on our GPU. The __device__ keyword lets us mark functions as callable from threads executing on the device by GPU. The __host__ keyword is for function can only be called by CPU. Both __host__ and __device__ keyword is for function as qualifiers can be combined. Note that the function __global__ and __device__ functions have access to these automatically defined variables [10, 11]. A variable is given by dim3 gridDim- Dimensions of the grid in blocks (at most 2D), dim3 blockDim -Dimensions of the block in threads, dim3 blockIdx -Block index within the grid, dim3 threadIdx- Thread index within the block with the keyword __ shared__ indicates that it will be stored in the shared memory of SM. The number of blocks in a grid and threads in a block should be declared by using dim3 announce, while CUDA kernel. Refer Figure 1.

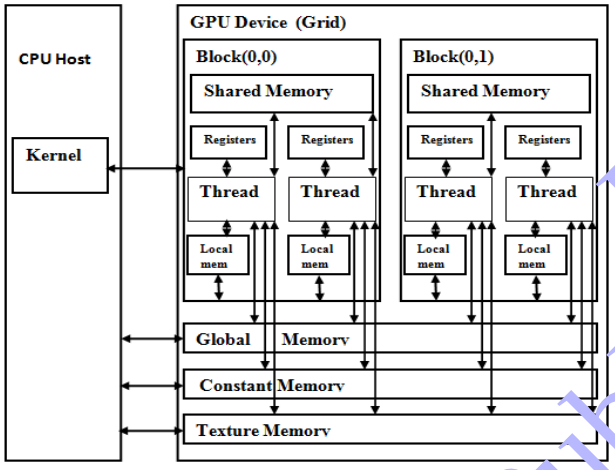

Figure 1. CUDA memory mode.

The variable dim3 should be incorporated as a parameter as follows:
dim3gridDim(i,j,k);
dim3blockDim(p,q,r);
kernel function <<<gridDim,blockDim>>> (a,b,c);
Where*i*, *j* and *k* are the number of blocks in *x*, *y* and *z* directions in grid. *p*, *q* and *r* are the number of threads in *x*, *y* and *z* directions in a block. *a*, *b* and *c* are the parameters of the kernel. The CUDA function calls differ from C function call only by the part <<<gridDim, blockDim>>>. This kernel is executed on GPU and called from CPU [13]. This kernel function should be declared with the Keyword __global__. The CUDA API essentially comprises functions for memory manipulation in VRAM: cudaMalloc to allocate memory, cudaFree to free it and cudaMemcpy to copy data between RAM and VRAM and vice-versa. We will end this section by explaining how a CUDA program is compiled. Compiling is done in several levels. In the first level, the code dedicated to CPU is extracted from the file and passed to the standard compiler. In the next level, the code dedicated to the GPU is converted into an intermediate language PTX which is like an assembler. Finally, the last level translates this intermediate language into commands that are specific to the GPU and encapsulates them in binary form which is executable [2, 16, 17].

Algorithm 2: MMDBM algorithm

Input: A *is the attributes containing n attributes* $A=\{a_1,a_2.....,a_n\}$ *in parallel*

Output: Distribution of the node count and construction of the decision tree.

1. *Initialize threads in GPU.*
2. *Data value were generate randomly in database.*
3. *Transfer the data from GPU device to CPU host (cudaMemcpy), dispatch the value in arrays. (Refer Figure 1 and 2).*
4. *Copy to GPU device and quick sort the random data from data base inside the device GPU.*
5. *Copy to CPU host and get the midpoint value of each and every attribute. (Refer Figure 1 and 2).*
6. a_i *is the attribute name and* v_i *is the midpoint value of each attribute,* $C=0$ *is the Class value and* $M=0$ *is the Missing value.*

   For $i=1$ *To* N *// N is the Number of the attribute nodes*

   Scan the attribute of all the records

   IF $a_i \le v_i$ *is true goto left child node and travel up to* N *number of the Node*

   IF $(x_1 \le v_1)$ *AND* $(x_2 \le v_2)$ *AND..... AND* $(x_n \le v_n)$ *THEN* C

   Count the class value, if the same data exists then update the appropriate class count value.

   $C=C+1;$

   else

   IF $a_i \le v_i$ *is false goto right child node and travel up to N number of the Node*

   Count the class value, if the same data exists then update the appropriate class count value.

   $C=C+1;$

   else

   Count the missing value and update the class count value.

   $M=M+1;$

   End If

   End If

   End For
7. *Transfer the data from CPU host to GPU device, classify the data, and Compute the node count and class count arrays. (Refer Figure 1 and 2)*
8. *Copy the result to CPU host, generate the distribution of the node count and construct of the decision tree (cudaMemcpy GPU device to CPU host).*

4. Design for MMDBM Classifier on CPU Computing and GPU Computing

The classification proceeds in four different phases: Attribute selection, Implementation of algorithm CPU

and GPU computing, Split point detection and Acceleration Ratio.

4.1. Attribute selection

The first phase is the attribute selection. This is done by accessing the randomly generated database and detecting the attribute values and the type of every attribute. Once an attribute is detected, its information is stored in a list called “Attribu”, which shown in Table 1.

Table 1. Attribute selection

Attribu
Categorical {Boolean} Name{String}
Initialize{Name, Data type}

Then sorting of the random data from the database inside the GPU device takes place. For every numerical attribute, this is done and the result is stored in an another array structure in the host. Once all the data is sorted, the split point can be found by accessing the middle element of the sorted array and it is stored in a variable called Mid* where * represents the name of every attribute, which shown in Table 2.

Table 2. Sorting Attributes

Sort Attributes (GPU- Radix and Quicksort)
Value{ Integer } Index { Integer }

Once pre-sorting is complete, the arrays containing the corresponding attribute values are created. This array has been loaded in to memory for classification. The leaf entry of each class list is initialized to '0', the root node of the tree.

4.2. Implementation of Algorithm CPU and GPU Computing

Once attribute selection is complete implementation of the algorithm starts. For each attribute, the corresponding attribute array is encumbered and the data is passed to the node pointed to by the leaf in the corresponding class list entry. Quick sort algorithm code samples have been provided for CPU and GPU computing.

Algorithm 3: CPU Code for Implementation of Quick sort algorithm

```
public static void quicksort(double[] a)
{ shuffle(a);
  quicksort(a, 0, a.length - 1);   }
public static void quicksort(double[] a, int left, int right)
{ if (right <= left) return;
  int i = partition(a, left,
  right);
          quicksort(a, left, i-1);
          quicksort(a, i+1, right);
}
private static int partition(double[] a, int left, int right)
{ int i = left - 1, int j = right;
while (true) {
while (less(a[++i], a[right]))
while (less(a[right], a[--j]))
          if (j == left) break;
          if (i >= j) break;
          exch(a, i, j);
}exch(a, i, right);
                    return i;
}  private static boolean less(double x, double y)
  {       comparisons++;
                    return (x < y);
          }
private static void exch(double[] a, int i, int j)
{         exchanges++;
          double swap = a[i];
          a[i] = a[j]; a[j] = swap;
}
private static void shuffle(double[] a)
{ int N = a.length;
 for (int i = 0; i < N; i++) {
 int r = i+(int)(Math.random()*(N-i));
 exch(a, i, r);
} }
```

4.2.1. Algorithm 3: GPU Code for Implementation of Quick sort Algorithm

Algorithm 4: GPU Code for Implementation of Quick sort algorithm

```
__global__static void quicksort   (int*values)
{
 int pivot, L, R;
 intidx =  threadIdx.x + blockIdx.x
 * blockDim.x;
 int start[MAX_LEVELS], int end[MAX_LEVELS];
 start[idx] = idx;
 end[idx] = SO - 1;
 while (idx>= 0) {
 L = start[idx];R = end[idx];
 if (L < R) {
          pivot = values[L];
           while (L < R){
          while (values[R] >=pivot && L< R)
          R--;
          if(L < R)
          values[L++] = values[R];
  while (values[L] <pivot && L < R)
          L++;
          if (L < R)
                    values[R--] = values[L];
}values[L] = pivot;
start[idx + 1] = L + 1;
end[idx + 1] = end[idx];
end[idx++] = L;
if (end[idx] - start[idx] > end[idx -1]- start[idx - 1])
{         swap start[idx] and start[idx-1]
          inttmp = start[idx];
          start[idx] = start[idx - 1];
          start[idx - 1] = tmp;
          tmp = end[idx];
          end[idx] = end[idx - 1],      end[idx - 1] = tmp;
}         }
```

```
else                    idx--;
}}
```

4.3. Split point detection

In our algorithm, we discover the mid-point by sorting the array and sorting the middle element which is sorted in array. After the midpoint is calculated, the arrays are accepted to the GPU classification function along with the midpoint which classifies the records. The GPU code for finding the mid-points is given below.

Algorithm 5: CPU Code for Split point detection of Quick sort algorithm

```
quicksort(arr);
for (int i = 0; i <dringVal; i++)
{       if (i == count || i==nextVal)
{           age=(int) arr[i];
            result = result+age;
}
}           midAge= result/2;, return midAge;
}
quicksort(arr);
for (int i = 0; i <dringVal; i++)
{           if (i == count || i==nextVal)
{           weight=(int) arr[i];
            result = result+ weight;
}           midWeight = result/2;
}           return mid weight;
}
```

Algorithm 6: GPU Code for Split point detection of Quick sort algorithm

```
quick_sort      <<<     MT      /       cThreadsPerBlock,MT/c
ThreadsPerBlock, N >>> (d_values);
cudaMemcpy(sag,d_values,size,cudaMemcpyDeviceToHost) ;
printf("\n\nMidPoint of the AGE is:\t");
MA=sag[SO/2];//Mid-point of the Age
printf("%d\n",MA);

quick_sort<<<           MT          /           cThreadsPerBlock,
MT/cThreadsPerBlock,N       >>> (r_values);
cudaMemcpy(swt,r_values,size,cudaMemcpyDeviceToHost) ;
printf("MidPointoftheWEIGHT is\t");
MW=swt[SO/2];//Midpoint of the Weight
printf("%d\n",MW);

quick_sort<<<           MT          /           cThreadsPerBlock,
MT/cThreadsPerBlock,N       >>> (a_values);
cudaMemcpy(sslp,a_values,size,cudaMemcpyDeviceToHost) ;
printf("MidPointofthe SLEEP is\t");
MSLP=sslp[SO/2];Midpoint of the Sleep
printf("%d\n",MSLP);

quick_sort<<<           MT          /           cThreadsPerBlock,
MT/cThreadsPerBlock,N >>> (s_values);
cudaMemcpy(sdr,s_values,size,cudaMemcpyDeviceToHost) ;
printf("MidPoint ofthe DRINK is:");
MDR=sdr[SO/2];/Midpoint of the Drinks
printf("%d\n",MDR);

quick_sort<<<           MT/cThreadsPerBlock,        MT          /
cThreadsPerBlock, N >>> (v_values);
cudaMemcpy(ssp,v_values,size,cudaMemcpyDeviceToHost) ;
printf("MidPointofthe SPROTS is\t");
MSP=ssp[SO/2];Mid-point of the Sports
printf("%d\n",MSP);
```

After getting the midpoint values and scanning the attributes of the all the records from connected data sets, we classify the node using $IF\,(x_1 \le v_1)\,AND\,(x_2 \le v_2)\,AND\ldots..\;AND\,(x_n \le v_n)$ $THEN\;C$ (class value) rule [1, 6, 7, 20]. *IF* this rule is true *goto* left child node and travel up to *N* number of nodes and finally count the class value, if the same data exists then the count value and update the appropriate class count. *IF* condition is false *goto* right child node and travel up to *N* number of nodes and finally count the class value, if the same data exists then the count value and update the appropriate class count and else update the missing count. Finally distribution of the node counts are evaluated based on the Predicted Rules (Figure 9) and the histogram of the classified nodes are calculated with construction of the decision tree (Figure 4). The design of MMDBM algorithm is presented in Figure 2.

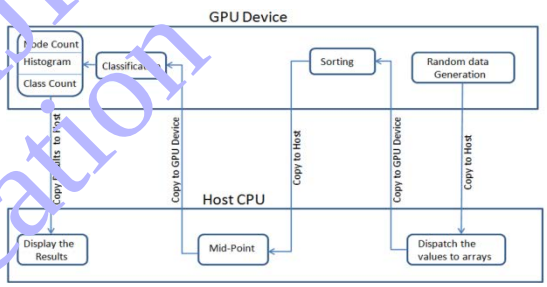

Figure 2. Design for GPU MMDM Classifier.

4.4. Acceleration Ratio for GPU

GPU Performance: To test the acceleration performance, an acceleration ratio (speed-up) γ is defined as equation 3.

$$\gamma = \frac{t_{CPU}}{t_{GPU}} \tag{3}$$

where the total processing time on the CPU, t_{CPU}, comprises only the time of main loop executed while the total processing time on the GPU, t_{GPU}, includes additional time of transferring data between Host and Device in the interest of fairness. γ first rises as the number of threads increases. There are two reasons for the changes in γ. First, if the number of threads is less, the time spent on data transferring between Host and Device takes up a considerable proportion of the total processing time of the GPU. As the number of threads increases, the proportion decreases rapidly. Second, only after all blocks in a kernel executed the next kernel can be launched in the GPU. The time consumed on kernel launching can be roughly considered as a fixed cost. Using more blocks means a

reduction in the percentage of kernel launching time [15].

$$CPU\ Time = Generate\ the\ random\ Values + Sorting\ Time + Classification\ Time \quad (4)$$

$$GPU\ Time = Data\ transfer\ from\ Host\ to\ Device\ and\ Device\ to\ Host \quad (5)$$

$$Acceleration\ Ratio = CPU\ computation\ Time / GPU\ computation\ Time \quad (6)$$

We calculated CPU, GPU and Acceleration ratio time with all the records sorting time is 0.00, because number thread has been created.

Table 3. Acceleration Ratio time for Quick sort

Quick sort GPUs Times	Records Sec / 20000	Records Sec / 40000	Records Sec / 60000	Records Sec/ 80000	Records Sec / 100000
random Values	0.100	0.210	0.300	0.4000	0.5100
Sorting Time	0.000	0.000	0.000	0.000	0.010
Classification Time	0.510	1.010	1.530	2.0300	2.580
CPU Time	0.610	1.230	1.840	2.4500	3.100
GPU Time	0.510	1.010	1.540	2.030	2.590
Acceleration Ratio	1.196	1.217	1.194	1.2068	1.1969

Table 4. Acceleration Ratio time for Radix sort

Radix sort GPUs Times	Records Sec / 20000	Records Sec / 40000	Records Sec / 60000	Records Sec / 80000	Records Sec / 100000
Random Values	0.090	0.210	0.320	0.4100	0.5100
Sorting time	0.000	0.000	0.000	0.000	0.010
Classification Time	0.510	1.010	1.530	2.0400	2.590
CPU Time	0.610	1.220	1.850	2.4600	3.110
GPU Time	0.510	1.010	1.530	2.040	2.5900
Acceleration Ratio	1.196	1.207	1.209	1.2058	1.2[illegible]

Comparison of acceleration ratio time for CPU andGPU Quick sort with Radix sort algorithm, which shown in Figure 3.

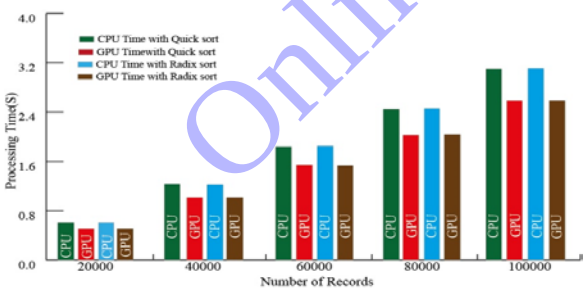

Figure 3. Scalability of the CPU and GPU processing time.

5. Main Result

We have tested the efficiency of our classification algorithm and have been implemented in different types of sorting algorithms (quick sort and radix sort) and compared the results of CPU computing with GPU computing. We now consider Medical database for BP where data mining techniques are applied. Test has been carried out to estimate the perfection of classification and the classification handling time. The medical database for Blood pressure where both the algorithms quick sort and radix sort are given the task to predict risk of the person for having high BP, low BP, normal BP based on seven different types of attributes.

5.1. Medical database for BP

The medical database contains data from reviews conducted among patients. The database holds records of the following Attributes.

1. *Sex*: Categorical [M/F];
2. *Age*: representing the age of the person Numeric [years];
3. *Weight*: The weight of the person; Numeric [Kilo grams];
4. *Sports*: The extent of exercise a person, Numeric [1-10]
5. *Sleep*: The number of hours a person sleeps on an average: numeric [0, 24];
6. *Drink*: The extent of drinking of a person; Numeric [1-5];
7. *BP*: Categorical [HP, LP, NP], this is class values;

As explained in the algorithm of MMDBM algorithm, the data that satisfy the condition and the data that does not satisfy the condition, correspondingly the missing data have been recorded. Based on that, a histogram is developed for distribution of the nodes. Since decision tree is a binary tree, the number of nodes in the tree is 2^n -1, where n is the number of attributes. In our case, the number of attribute is 7 so the number of nodes is 127. As it is not possible to show all the distribution of the node, the histogram of total count for all nodes and three distributions are listed below

5.2. Histogram of all the nodes

All the attribute node value is taken as (sex, age, weight, sleep, sports, drink and BP). Out of One Lakh patients, the numbers of patients with high BP are 36160, the numbers of patients with normal BP are 31673, the numbers of patients with low BP are 31733 and the numbers of missing values are 434. It is depicted as histogram, which shown in Figure 5.

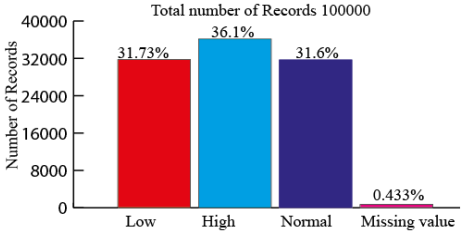

Figure 5. Total count for the distribution node.

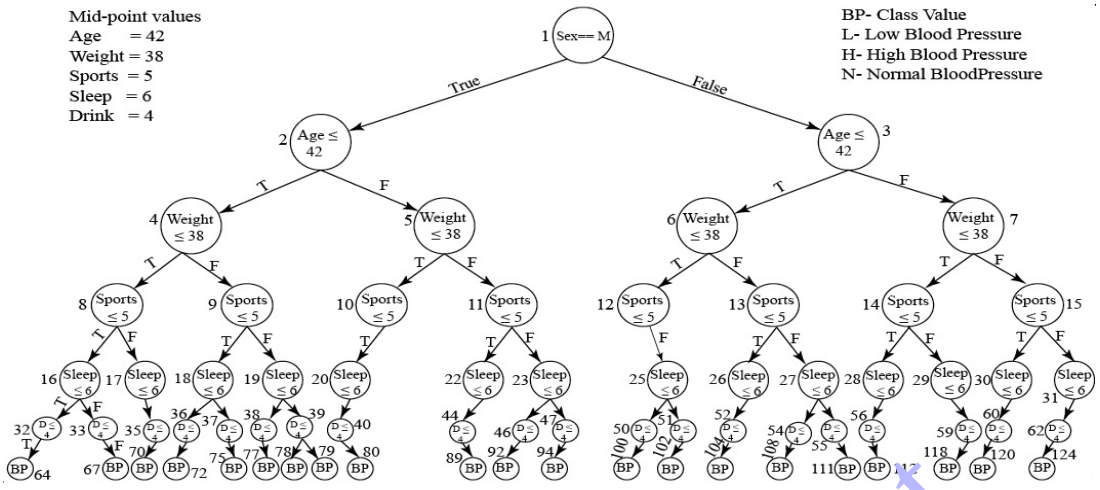

Figure 4. Classification tree in Medical database for BP

5.2.1. First distribution

All the attribute in node has travelled in $IF\,(Sex == F\ \&\ Age > 49\ \&\ Weight \le 35\ \&\ Sports \le 5\ \&\ Sleep \le 8\ \&\ Drink \le 3)\ THEN\ C$ (count the class value in BP) condition for class distribution and travel path is 1-3-7-14-28-56-112 (referred Figure 4 Classification tree). Out of One Lakh patients, the numbers of patients with high BP are106, the numbers of patients with normal BP are 106 and the numbers of patients with low BP are160. It is depicted as histogram, which shown in Figure 6.

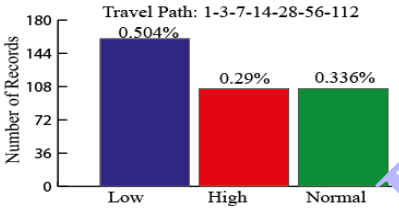

Figure 6. First distribution of the node

5.2.2. Second distribution

The all attribute in node has travelled in $IF\,(Sex == F\ \&\ Age > 49\ \&\ Weight > 35\ \&\ Sports > 5\ \&\ Sleep > 8\ \&\ Drink \le 3)\ THEN\ C$ (count the class value in BP) condition for class distribution and travel path is 1-3-7-15-30-60 (referred Figure 4 Classification tree).

Out of One Lakh patients, the numbers of patients with high BP are400, the numbers of patients with normal BP are294 and the numbers of patients with low BP are130. It is depicted as histogram, which shown in Figure 7.

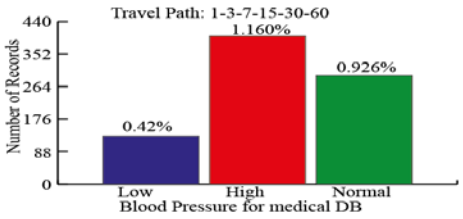

Figure 7. Second distribution of the node

5.2.3. Third distribution

The all attribute in node has travelled in $IF\,(Sex == M\ \&\ Age < 49\ \&\ Weight > 35\ \&\ Sports > 5\ \&\ Sleep \le 8\ \&\ Drink \le 3)\ THEN\ C$ (count the class value in BP) condition for class distribution and travel path is 1-2-4-9-18-36-72 (referred Figure 4 Classification tree). Out of One Lakh patients, the numbers of patients with high BP are68, the numbers of patients with normal BP are 104 and the numbers of patients with low BP are80. It is depicted as histogram which shown in Figure 8.

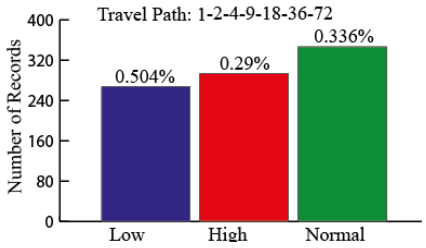

Figure 8. Thired distrubution of the node

Both Algorithms (quick sort and radix sort) are a large set of datasets which have been generated to test the precision minimum processing time by CPU and GPU computing. Figure 4 demonstrates the classification tree of the medical database which measuring the node count of the values. The classification testes were accepted with different amounts of data provided to the program extending from 10,000 to 1,00,000 and classification is completed. It is perceived that with the rise in the number of records, the estimate of accuracy in upgraded. Table 6 illustrates the estimate rules obtained from the database[6, 20].

Table 5. Processing time of SLIQ with MMDBM algorithms.

Algorithm Names	Processing time / Sec 10000	Processing time / Sec 30000	Processing time / Sec 40000	Processing time / Sec 50000	Processing time / Sec 100000
SLIQ	6.21	17.6	20.5	26.8	42.5
MMDBM Quick sort	0.432	1.069	1.076	1.467	2.106
MMDBM Merge sort	0.483	1.069	1.134	1.482	2.558
MMDBM Radix Sort	5.992	11.487	12.967	13.967	14.987

By comparing the results, given in Tables 3 and 4 with Table 5 and in the Figure 3 and Figure 9, we can conclude that GPU computes faster than CPU. Using GPU Quick sort algorithm, we can get a greater performance for all computational problems. This report is limited to one particular classification algorithm MMDBM, but it can be generalized to all other algorithms also, which is shown in Figure 9.

Table 6. Predicted Rules for Medical database in BP

N1	sex=M Node goto N2 else N3	N23	Sport<=4 Node goto N46 else N47	N45	drink<=4 Node goto N90 else N91
N2	Age<=35 Node goto N4 else N5	N24	Sport<=4 Node goto N48 else N49	N46	drink<=4 Node goto N92 else N93
N3	Age<=35 Node goto N6 else N7	N25	Sport<=4 Node goto N50 else N51	N47	drink<=4 Node goto N94 else N95
N4	Weight<=48 Node goto N8 else N9	N26	Sport<=4 Node goto N52 else N53	N48	drink<=4 Node goto N96 else N97
N5	Weight<=48 Node goto N10 else N11	N27	Sport<=4 Node goto N54 else N55	N49	drink<=4 Node goto N98 else N99
N6	Weight<=48 Node goto N12 else N13	N28	Sport<=4 Node goto N56 else N57	N50	drink<=4 Node goto N100 else N101
N7	Weight<=48 Node goto N14 else N15	N29	Sport<=4 Node goto N58 else N59	N51	drink<=4 Node goto N102 else N103
N8	Sport<=4 Node goto N16 else N17	N30	Sport<=4 Node goto N60 else N61	N52	drink<=4 Node goto N104 else N105
N9	Sport<=4 Node goto N18 else N19	N31	Sport<=4 Node goto N62 else N63	N53	drink<=4 Node goto N106 else N107
N10	Sport<=4 Node goto N20 else N21	N32	drink<=4 Node goto N64 else N65	N54	drink<=4 Node goto N108 else N109
N11	Sport<=4 Node goto N22 else N23	N33	drink<=4 Node goto N66 else N67	N55	drink<=4 Node goto N110 else N111
N12	Sport<=4 Node goto N24 else N25	N34	drink<=4 Node goto N68 else N69	N56	drink<=4 Node goto N112 else N113
N13	Sport<=4 Node goto N26 else N27	N35	drink<=4 Node goto N70 else N71	N57	drink<=4 Node goto N114 else N115
N14	Sport<=4 Node goto N28else N29	N36	drink<=4 Node goto N72 else N73	N58	drink<=4 Node goto N116 else N117
N15	Sport<=4 Node goto N30 else N31	N37	drink<=4 Node goto N74else N75	N59	drink<=4 Node goto N118 else N119
N16	Sport<=4 Node goto N32 else N33	N38	drink<=4 Node goto N76 else N77	N60	drink<=4 Node goto N120 else N121
N17	Sport<=4 Node goto N34 else N35	N39	drink<=4 Node goto N78 else N79	N61	drink<=4 Node goto N122 else N123
N18	Sport<=4 Node goto N36 else N37	N40	drink<=4 Node goto N80 else N81	N62	drink<=4 Node goto N124 else N125
N19	Sport<=4 Node goto N38 else N39	N41	drink<=4 Node goto N82 else N83	N63	drink<=4 Node goto N126 else N127
N20	Sport<=4 Node goto N40 else N41	N42	drink<=4 Node goto N84 else N85	N64	Terminated 100% with H
N21	Sport<=4 Node goto N42 else N43	N43	drink<=4 Node goto N86 else N87	N65	Terminated 100% with L
N22	Sport<=4 Node goto N44 else N45	N44	drink<=4 Node goto N88 else N89	N66	Terminated 100% with H

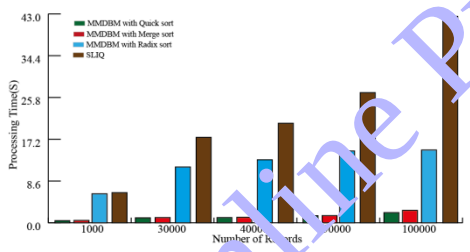

Figure 9. Scalability of SLIQ with MMDBM algorithms

CUDA is not a new program, but Extension of C programming language. CUDA code must be compiled using nvcc (nvidia cuda c compiler) Compiler. It works on modern nvidia cards (Quadro, GeForce, Tesla). We have used "Geforce GT525M" card for this research.

This test was carried out on Microsoft windows 7 together with CUDA version 5.0. The hardware platform consists of a Intel core i5-245M CPU 2.50 GHz and 6 GB RAM. Table 7 summarizes GPU characteristic used in experiments. The NVIDIA GT525M card used in GPU is with 4095M.

Table 7. Characteristic of Geforce GT525M card

Property	Value
CUDA Core	96
Graphics clock	475 MHz
Process clock	950 MHz
Memory clock	900 MHz
Memory Interface	128- it
Total available graphics	4095 MB
Dedicated video memory	2048 MB
Shared system memory	DDR3
	2047 MB

6. Conclusion

Classification is one of the major tasks in data mining. A new classifier called MMDBM has been programmed in CPU computing (Java) with quick sort and radix sort algorithms and has been tested using Medical database and also same two algorithms have been programmed in GPU computing. The algorithm can handle huge amount of datasets with large amount of Attributes. GPUs quick sort and radix sort algorithm provides exceptional scalability with the Medical data sets that has been taken for analysis and testing. The main Results have been taken into consideration and verified for accuracy and the program code is provided for CPU and GPU computing. We have discussed an

efficient parallel quick sort and radix sort algorithms in GPU and results from the Comparison of computational acceleration ratio (speed-up) and efficiency of processing time of CPU and GPU computing in MMDBM Classifier. The main results are used to compare the classifier with an existing CPU- quick sort and radix sort for the MMDBM classifier and GPU- quick sort and radix sort algorithms provide rapid and exact results with minimum execution time and supports real time applications.

References

[1] AI-hegami A., "Pruning Based Interestingness of Mined Classification Patterns," *International Arab Journal of Information Technology*, Vol. 6, No. 4, pp. 336-343, 2009.

[2] Alnihoud J., and Mansi R., **"**An Enhancement of Major Sorting Algorithms," *International Arab Journal of Information Technology*, Vol. 7, No. 1, pp. 55-62, 2010.

[3] Cederman D., and Tsigas P., "GPU-Quicksort: A Practical Quicksort Algorithm for Graphics Processors," *ACM Journal of Experimental Algorithmics*, Vol. 14, No. 4, 2009.

[4] Chiu Chun-Chieh., Luo Guo-Heng., and Yuan Shyan-Ming., " A decision tree using CUDA GPUs," *in Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, pp. 399-402, 2011.

[5] Chandra B., Paul Varghese P., **"**Moving towards efficient decision tree construction**,"** Information Sciences, 179, 1059–1069, 2009.

[6] Ganesan P., Sivakumar S., and Sundar S., "A Comparative Study on MMDBM Classifier Incorporating Various Sorting Procedure," Indian Journal of Science and Technology, Vol 8(9), pp.868–874, 2015.

[7] Mehta M., Agarwal R., and Rissanen J., "SLIQ: A Fast Scalable Classifier for Data Mining," *Int. Conference on Extending Database Technology(EDBT)*, Avignon, France, 1996.

[8] Munshi A., "OpenCL: Parallel Computing on the GPU and CPU," *in the 35st international conferenceon computer graphics and interactive techniques*, California, USA, 2008.

[9] Nasridinov A., Lee Y., and Park Young-Ho., **"**Decision tree construction on GPU: ubiquitous parallel computing approach," *Computing*, 96, pp.403-413, 2014.

[10] Nguyen H., *GPU Gems 3*, Addison-Wesley Professional, 2007.

[11] Nickolls J., Buck I., and Garland M., "Scalable Parallel Programming with CUDA," *ACM Queue*, Vol. 6, No. 2, pp. 40-53, 2008.

[12] NVIDIA Corporation., *CUDA Best Practices Guide*, 2010.

[13] NVIDIA Corporation., *CUDA C Programming Guide*, February 2014.

[14] Ömer AKGÖEK., **"**A rule induction algorithm for knowledge discovery and classification," *Turkish Journal of Electrical Engineering and Computer Science*, 21, pp.1223-1241, 2013.

[15] Panchatcharam M., Sundar S., Vetrivel V., Axel Klar., and Tiwari S., "GPU computing for meshfree particle method," *International Journal of Numerical Analysis and Modeling, Series B*, Vol. 4, pp. 394-412, 2013.

[16] Pospichal P., Jaros J., and Schwarz J., "Parallel genetic algorithm on the CUDA architecture," *Application of Evolutionary Computation*, pp. 442-451, 2010.

[17] Sanders J., and Kandrot E., *CUDA by example : An introduction to general-purpose GPU programming*, Addison-Wesley, 2011.

[18] Shafer J., Aggrawal R., and Mehta M., "SPRINT: A Scalable Parallel Classifier for Data Mining," *in Proceedings of 22nd VLDB Conference*, San Antonio, pp. 962-969, 1996.

[19] Sundar S., Srikanth D., and Shanmugam M.S., "A new predictive classifier for improved performance in data mining: object oriented design and implementation," *in Proceedings of the International Conference on Industrial Mathematics*, IIT Bombay, Narosa, pp. 491-514, 2006.

[20] Telbany M., Warda M., and Borahy M., "Mining the Classification Rules for Egyptian Rice Diseases," *International Arab Journal of Information Technology*, Vol. 3, No. 4, 2006.

Sivakumar Selvarasu obtained his Msc, Computer science from Periyar E.V.R College and M.Phil from St.Joseph's College of Bharathidasan University, India, Now he is doing Ph.D in Department of Mathematics, College of Engineering Guindy Campus, Anna University, Chennai, India. His research program focuses on Data mining and areas of interest are parallel computing (GPU), Classification and analysis, Image processing and CUDA Programming.

Ganesan Periyanagounder is working as a Professor of Emeritus in the Department of Mathematics, College of Engineering Guindy Campus, Anna University, Chennai, India. He received his Ph.D degree from Indian Institute of Technology, Mumbai, India. He has more than 30 years of experience in academic and research. His areas of interest include Computational Heat Transfer, Theoretical computer science and Object Oriented programming. He has more than 82 publications in Journals and Conferences.

Sundar Subbiah is working as a Professor in the Department of Mathematics, Indian Institute of Technology, Madras, Chennai, India. He received his Ph.D degree from Indian Institute of Technology, Chennai, India. He has 26 years of experience in academic and research. His areas of interest include Numerics for PDEs, Mathematical Modeling & Numerical Simulation, GPU Computing and Data mining. He has guided over 12 Ph.D students and 75 M.Tech / MSc students. He has more than 80 publications in Journals and conducted 11 International Conferences and Workshops.