FPGA based accelerator for parallel DBSCAN algorithm

Shaobo Shi, Qi Yue, Qin Wang*

School of Computer and communication engineering, University of Science and Technology Beijing

Received 1 January 2014, www.tsi.lv

Abstract

Data mining is playing a vital role in various application fields. One important issue in data mining is clustering, which is a process of grouping data with high similarity. Density-based clustering is an effective method that can find clusters in arbitrary shapes in feature space, and DBSCAN (Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise) is a basic one. With the tremendous increase of data sizes, the processing time taken by clustering algorithms can be several hours or more. In recent years, FPGA has provided a notable accelerating performance in data mining applications. In this paper, we study parallel DBSCAN algorithm and map it to FPGA based on the task-level and data-level parallelism architecture. Experimental results show that this accelerator can provide up to 86x speedup over a software implementation on general-purpose processor and 2.9x over a software implementation on graphic processor.

Keywords: Data mining, Clustering, Parallel DBSCAN, FPGA, Hardware Accelerator

1 Introduction

Clustering is an effective approach of retrieving useful patterns from raw data sets. The process of clustering is to group data into different clusters so that objects in the same cluster have high similarity. And clustering is an important data mining tool that has applied in many areas. DBSCAN is an effective density-based clustering method, which is proposed by Martin [1]. Compared with other clustering algorithms, DBSCAN has some obvious advantages such as requiring minimal domain knowledge, being able to discover clusters in arbitrary shapes, being robust in removing noise and outliers, and having a good efficiency on large databases.

However, performing DBSCAN algorithm in practice is limited by the fact that the performance of general processors is improving at slower rate comparing to rapidly growing data set size. For this reason, a preferable method of accelerating executing speed is to make an algorithm parallelized. Some researchers proposed parallel DBSCAN algorithm and mapped it to distributed parallel computing platform, and achieve a near-linear speedup performance [2, 5]. However, this solution is not energy efficient since distributed parallel computing platform requires high power consumption.

Field-programmable gate arrays (FPGAs) are used as user-customized computing engines for accelerating a wide range of applications. The high-end FPGAs are characterized with enormous amount of logic gates, abundant on-chip memory and large capacity external storage, flexible programmability and lower power consumption. With these features, users can utilize multiple operation-levels and high memory access bandwidth for specific applications. Thus, we take FPGA as hardware platform and study the mapping from parallel DBSCAN algorithms to FPGA to get a higher speedup performance in this paper.

Mapping a parallel DBSCAN algorithm to FPGA should consider two issues. The first one is how to fully exploit parallel DBSCAN algorithm with the high flexibility of FPGA structure. In addition, the second is how to eliminate data dependencies existing in parallel algorithm. The contributions of this paper are summarized as follows:

- ✓ We propose a hardware architecture based on task-level and data-level parallelism, which fully exploit the bit-level parallelism provided by FPGA.
- ✓ We design a data reused pipeline structure to eliminate the extra memory access caused by the data dependencies in parallel algorithm.
- ✓ Based on previous work, we propose a more robust parallel algorithm. which can avoid wrong clustering results in some special conditions.
- ✓ To the best of our knowledge, it is the first work of implementing a parallel DBSCAN algorithm on reconfigurable hardware. Compared to the sequential software implementation on Intel general processor, our accelerator can achieve80x speedup. Besides, we can get a 2.9x speedup over the similar parallel implementation on Nvidia graphic processor.

The rest of this paper is organized as follows. In section2, we review the previous work on parallel DBSCAN implementations and hardware accelerations for the data mining applications. We introduce the

* Corresponding author - E-mail:337816437@qq.com

original DBSCAN algorithm and propose complete robust parallel DBSCAN algorithm in section3. In section 4, the accelerator hardware architecture and the specific design are presented. The performance model and experimental result are discussed in section 5. Section 6 is conclusion.

2 Related Work

Parallel DBSCAN implementation: Several works for parallelizing DBSCAN algorithm are proposed. PDBSCAN [2], whose parallel partitions are based on regionalism, is an implementation on the master-slave mode computer cluster [3] maps the sequential kernel of DBSCAN to a higher level parallel programming environment [4] is another implementation focus on parallelization of DBSCAN through simple distance function. In [5], author presented an improved work to [1] that mapped parallel algorithm to the MapReduce framework. All of these researches are based on the region partitions, and use shared-nothing architecture to run parallel algorithm. Meanwhile, the parallelism of these methods is too low, so they are not suitable for being implemented on FPGA.

CUDA-DClust is implemented with GPU [6], which is a fine-grained parallel DBSCAN algorithm and it is different from the above mentioned works. Moreover, this work can avoid the boundary processing and load unbalance issues caused by region partitions. Due to these advantages, CUDA-DClust is a good reference and comparable object for our work. However, due to the long communication delay between different Streaming Multiprocessors in GPU, the data dependency in parallel DBSCAN algorithm will increase the memory access times and thus influence the performance when it mapped on the GPU. The experiments in [7, 8] point out a fact that implementation of a customized data path in FPGA can provide a superior performance over GPU in the presence of data dependency. Therefore, the optimized and customized hardware architecture for parallel DBSCAN appears very necessary. Compared with the software implementation of DBSCAN in parallel computers system and GPU, our approach is far better than these methods in both performance and power consumption.

Hardware accelerator for data mining applications: There have been many prior researches on hardware implementation of data mining algorithms. In [9], K-means clustering is implemented as a reconfigurable accelerator, which simplified the distance calculation. However, K-means is essentially different from DBSCAN since it is a partitioned algorithm. In [10], the kernel of HOP algorithm is implemented on FPGA platform. Although HOP is a kind of density-based clustering algorithm, the final goal of this algorithm is to find the nearest densest neighbours rather than the transitive closure computation in DBSCAN. In addition, those works did not work on the parallelism from the view point of algorithm. In [11], hardware architecture for Decision Tree Classification (DTC) algorithm is described.

Other hardware implemented clustering algorithms are summarized as follows. The Apriori algorithm, a popular association rule mining algorithm, is accelerated by systolic array architecture in [12]. And its improved work with bitmapped CAM is proposed in [13]. The HAPPI architecture is proposed in [14] with the pipeline and hashing methodology to resolve the bottleneck of Apriori. In [15] the FP-Growth algorithm is firstly mapped to a systolic tree structure by mimicking the internal memory layout of software algorithm.

3 Sequential DBSCAN algorithm and Parallel model

3.1 SEQUENTIAL *DBSCAN* ALGORITHM

The original DBSCAN algorithm is a sequential clustering algorithm [1]. The key idea of DBSCAN is that the data density within a small area in the feature space must exceed a given threshold, i.e., the neighbourhood of each point in a cluster must contain a minimum number of points. To make a clear presentation of the proposed method in this paper, we introduce some basic definitions of DBSCAN as follows:

1. $N_{Eps}(p)$ *is a set* : $\{q \in D \mid dist(p,q) \le Eps\}$

2. *Directly density – reachable* : *A point* p *is directly density – reachable from a point* q *in the set of point* D *if* $p \in N_{Eps(q)}$ *and* $Numb(N_{Eps(q)}) \ge MinPts$

3. *Density – reachable* : *A point* p *is density – reachable from a point* q *in the set of point* D *if there is a chain of points* $p_1,...,p_n, p_1 = q, p_n = p$ *such that* $p_i \in D$ *and* p_{i+1} *is directly density – reachable from* p_i

4. *Density – connected* : *A point* p *is density – connected to a point* q $o \in D$ *such that both* p *and* q *are density – reachable from* o *w.r.t Eps and MinPts in* D

5. *Cluster* : *A cluster* C *is a non – empty subset of* D *iff* :
Maximality :
$\forall p,q \in D$: *if* $p \in C$ *and* $q > D_p$ *then also* $q \in C$.
Connectivity :
$\forall p,q \in C$: p *is density – connected to* q *in* D.

6. *Core point* : *o is a Core point if* $N_{Eps(o)} \ge MinPts$.

7. *Border point* : *p is a Border point if* $N_{Eps(p)} \le MinPts$ *and p is Directly density – reachable from a Core point*.

8. *Nosie point* : *p is a Nosie point if* $N_{Eps(p)} \le MinPts$ *and p is not Directly density – reachable from any Core point*.

With these definitions, we show the simple sequential proceeds of algorithm as follows:

S1: For arbitrary unclassified point P ∈ D,

S2: Retrieve(P), If P is core point, mark with ClusterID;

S3:For the unclassified or noise point Q $\in N_{EPS}(P)$, store them into SeedStack, mark with ClusterID;

S4: While SeedStack not empty, for the top element P _top of SeedStack
S5: Repeat S2-S4
S6: If D have unclassified point, goto S2

The main task of DBSCAN is retrieve function (step S2) for find the neighbourhood of each point and finds the transitive closure relationship. The complexity of DBSCAN is O(N2) without index structure, and is O(Nlog(N)) if with a multidimensional index structure. We run the sequential non-index algorithm in Intel Vtune Performance Analyzer Tool, the result shows that CPI rate is 0.761 and the Data Reference per instruction retired (Load/Store) is 0.742, according to the MineBench in [16]; DBSCAN is compute-intensive and memory-intensive. Our hardware architecture is designed for non-index DBSCAN, but it can be transplanted to fit the DBSCAN with index structure easily.

3.2 PARALLEL MODEL BASED ON THE TempCluster

From sequential algorithm, DBSCAN can start at an arbitrary point, which is unclassified to find the other points that fulfil the maximum density connectivity. Intuitively, we can unroll this loop so that DBSCAN can start at multiple unclassified points. This is the basic parallel concept that different clusters with respective ClsterIDs can be clustered simultaneously. These clusters are identical with the concept of 'Chain' proposed in [6]. Compared to the definition of cluster, a chain is a set of data object belong to a common density-based cluster that do not have to meet maximalist. In other words, a chain can be considered as a tentative cluster with a tentative clusterID. Different chains may have collisions that mean these chains belong to the same cluster, so the collision check mechanism is necessary.

Our hardware parallel model is based on chain, but we have a robust collision check mechanism that is described later. In order to distinguish 'Chain' from each other, we call it as TempCluster (TC). The definition of TempCluster is:

$$\begin{aligned} &TempCluster\ is\ a\ subset\ C \subseteq D, if\ \ and\ only\ if\ :\\ &\forall p,q \subseteq C : p\ is\ Density\text{-}connected\ \ from\ \ q\ each\ \ other. \end{aligned}$$

$$\begin{aligned} &Clusters \subseteq D : \{C_1 \bigcup C_2 \bigcup C_3 \bigcup ..... \bigcup C_n\},\\ &C_i \subseteq D : \{TC_{i1} \bigcup TC_{i2} \bigcup TC_{i3} \bigcup ..... \bigcup TC_{in}\} \end{aligned}$$

Thus, the clustering results of D are composed of one or several C_i and each C_i is made up of one or several TC_{ij}. Therefore, we can devise the customized data path for the parallel generation of multiple TC_{ij} with different start points.

In [6], the valid condition of collision is the point in an arbitrary TC should be marked already has a different TempClusterID (TCID). This means that this collision point has been marked by other TCs (step S2), so this point should not be store into SeedStack of local TC (step S3). However, the collision check criteria proposed in [6] may generate wrong clustering results under special condition. As shown in Figure1(a), the black point is this special collision point. This point is not a core point for its neighbour points are less than Minpts. TC1 and TC2 should not be merged into one cluster, because they cannot satisfy the density connected condition according to definition4, and. If the TC1 and TC2 are merged into one cluster, the clustering effectiveness of DBSCAN will be impacted seriously. We propose a robust collision check criterion as follows:

$$\begin{aligned} &Let\ TempCluster1\ and\ TempCluster2\ be\\ &found\ in\ D, TempCluster1 \cup TempCluster2\\ &belong\ to\ same\ cluster\ if\ \ and\ only\ if\ :\\ &\exists o \subseteq D, o \in TC1 \cap TC2, and\ o\\ &1, o\ is\ border\ \ point\ \ in\ \ TC1, and\ \ o\ is\ core\ \ point\\ &\ \ in\ \ TC2;\\ &2, o\ is\ border\ \ point\ \ in\ \ TC2, and\ \ o\ is\ core\ \ point\\ &\ \ in\ \ TC1;\\ &3, o\ is\ core\ \ point\ \ in\ \ both\ \ \mathrm{TC1}\ \ and\ TC2. \end{aligned}$$

This check method utilizes the concept of border property and core property, and can exclude the special situation that two clusters just share with a border point. As shown in fiugre1(b), the black point is a valid collision point. Therefore, our collision check criteria can avoid occurrence of special phenomenon described above. In our later implementation, the collision check procedure must check the border value and core value in each point data. A detailed description of our check method is presented in alogrithm2 of Section4.

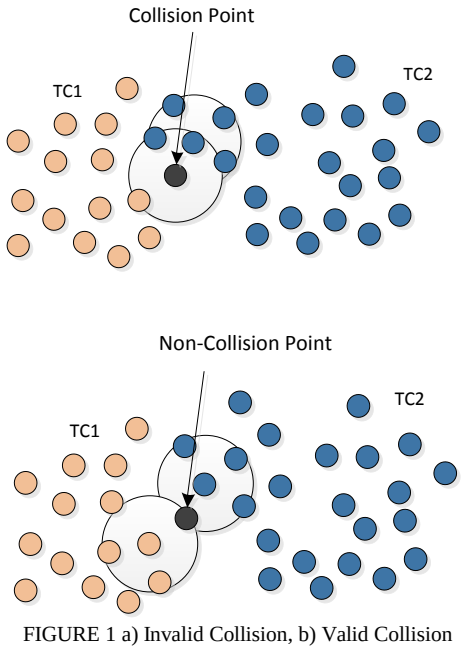

FIGURE 1 a) Invalid Collision, b) Valid Collision

4 Hardware architecture

As Figure 2 show, DBSCAN algorithm accelerator consists of FPGA chip(XC6VLX240T) and external memory which includes DDR3 SDRAM. The accelerator loads data from host PC to SDRAM through PCIe in DMA mode, and user can read the clustered data when the algorithm done from accelerator in the same way. We mapped parallel hardware architecture to FPGA. Some key module such as Processing Elements Array (PE Array) is responsible for perform the parallel TempCluster expansion. PE Array control unit is used as manages the control signal, and update the candidate data in PE Array.

And other modules such as memory interface (PCI-I interface, DDR3 SDRAM interface) and I/O FIFO are responsible for memory access and data buffer.

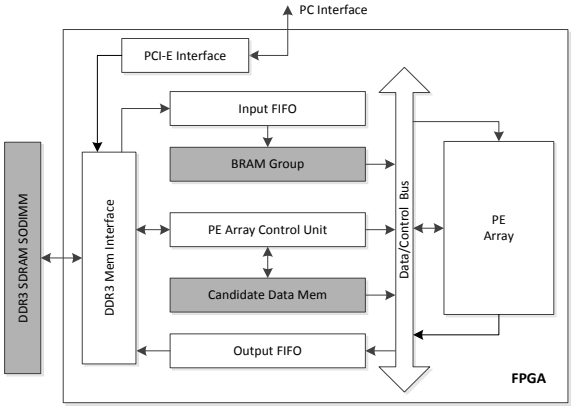

FIGURE 2 Architecture of Parallel DBSCAN accelerator

The raw data store in the SDRAM we call it Spatial Data (SPData), for example, one 2-dimision data occupy 64bits and is composed of several properties data. The format of this data is shown in the figure 3. The ID data has 20 bits that can support the representation of 10^6 data amount. Core point and neighbour point properties use 1 bit respectively, and ClusterID use 9 bits. The stack property also use 1 bit. We also call the high-order 32bits data as data property. Each coordinate value is a 16 bit fixed point data. For the high dimension data, we only need expend the data property.

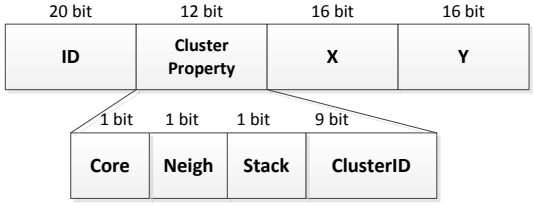

FIGURE 3 Data format in SDRAM

4.1 OPTIMIZED PE ARRAY DESIGN STRATEGY FOR COLLISION CHECK

We adopt multiple PEs to perform multiple TempCluster calculation in parallel pattern, and single PE is just responsible for one TempCluster. In GPU architecture, it is a similar processing method for parallel algorithm. However, data dependencies caused by collision check can decrease the performance when parallel DBSCAN executes on GPU. Left part of figure 4 demonstrates how data dependency affects GPU parallel implementation. In GPU, all of SMPs (StreamingMultiProcessors) share data through the Global Memory (GDDR SDRAM), and the SPData are stored in the Global Memory. Each SMP is responsible for a TC. For a simple example, SPData2 is read by SMP1, and meet the ClusterID marking condition through calculation. Thus the SPData2 should return its latest ID property to SMP1 in order to check whether it has been marked by other SMPs. And in this procedure is performed by an atomic operation provided by GPU software, other SMPs cannot access SPData2 before read operation of SMP1complete. Since the situation that all SMPs have a large latency to Global Memory and memory access conflicts caused by data dependency, parallel DBSCAN algorithm is not so efficient when mapped to GPU architecture.

Aimed at this problem, a solution in hardware design that minimizes the overhead of data dependency is to make collision check between PEs in local memory and avoid memory access conflicts. And the abundant register resource and design flexibility of FPGA make this solution possible. As right part of Figure4 shows, the latest ID property SPData0 can be transmitted from PE1 to PE2 directly through the buffer. If SPData0 in PE2 should be checked, we can obtain its latest state from buffer without reading SDRAM. And this buffer can only be written by PE1 and read by PE2, so we can avoid memory access conflicts. The more detailed operation-level design will be described later.

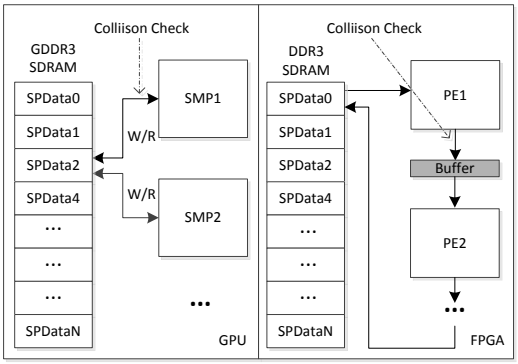

FIGURE 4 Difference between GPU and FPGA in Memory allocation

4.2 PE ARRAY DESIGN BASED ON DATA REUSES AND TASK-LEVEL PARALLELISM

We propose a pipeline PE array based on the data reuse, to eliminate the extra memory access caused by collision check. As the Figure 5 show, This PE array can work in a fully task-level parallel mode. In the seed expansion, PE will calculate distance with every SPData in Dataset (with the index structure, is the all of the region data that fulfil the query condition). Thus, this is make the data reuse possible. We design register files in each PE, called SpatiData Reg. And the SPData from SDRAM will the transfer to every PE cycle by cycle in the form of data streaming. PE#1 is responsible for the load data from SDRAM, other PEs load data from previous PE's SpatiData Reg. Lastly, PE#N will write clustered data back to SDRAM.

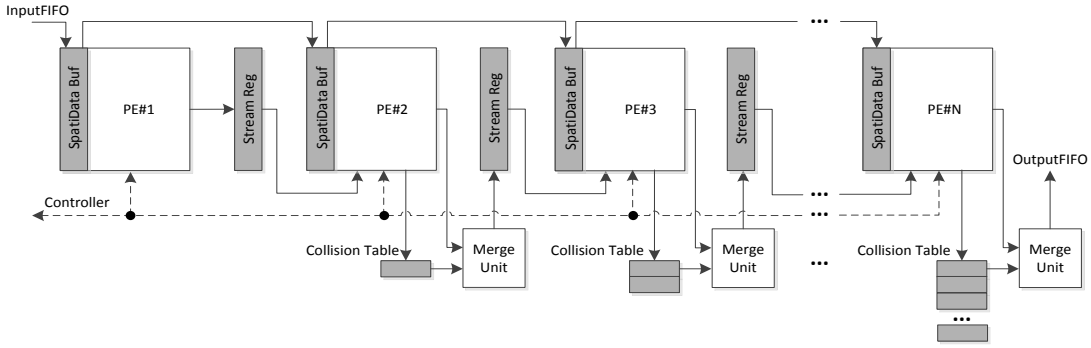

FIGURE 5 Architecture of PE array

The output of PE is the property data which has been compared with Eps, and its high-order 32-bits in SPData. We insert register files between each PE to buffer these output data, called Stream Reg. For the fact that PE#i+1 always fall behind PE#i one cycle, so buffering output one cycle can play an synchronous role when each PE perform collision check. The LableCheck Unit of PE#i+1 can get the output of PE#i simultaneously, and make collision check according to the criteria mentioned before. The algorithm executed on the PE Array showed in Algorithm 1.

Algorithm1. Parallel Algorithm for PE Array

```
Parallel DBSCAN Algorithm for PE array in one Candidate corepoint Calc:
Input: SpatialDataset D,ε,Minpts
Output: Clusterd SpatialDataSet ,CollisionSignal
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Initi: Generate Seed[N], TCID[N]
Dirtribute Dataset to PE#i one by one
For i=1:N PE#i Parallelly Do:
  ExpandTempCluster(Seed[i],TCID[i],ε,Minpts,Dataset,ClusterDataset);
  receive clusterd Dataset from Pre StreamReg;
  send clusterd Dataset to Next StreamReg;
  If PE#i return false
    Generate new seed to PEi;
    mark candidate point as Nosie;
  Else PE#i has Collision PE# X
    send TCID[i] and TCID[X] to Controller unit
   Update Collision Matrix;
  End IF
End For
```

This collision check method make PE obtained the latest data, and should generate collision signal without pipeline break off. From this figure we can see that the PE2 to PE N are likely to generate the collision signal, and the generation is simultaneous. And the i-th PE has a potential collision signal with anyone among the first PE and (i-1)-th PE. Therefore, each PE must have a structure to record the collision signal. Due to the different amount of potential collision signal in each PE and the small storage amount, we adopt the register file to implement the function of collision signal record, we called it Collision Table. Besides the collision signal record unit, we also need a unit to process these collision signals, and it is called Merge Unit. The function of Merge Unit is to update the cluster property of every point with the new ClusterID according to the collision table.

4.3 PE DESIGN BASED ON DATA-LEVEL PARALLELISM

Since the high bandwidth of SDRAM, each PE can obtain several SPData in one clock, and PE can perform data-level computation. For the data which arrive at successive clock have no data dependency, so we can map sequential process procedure in pipeline. The DCU Array performs parallel Euclidean distance calculation between seed data and SPData, the distance formula is: $D_i = \sum_{j=1}^{K} (CD_j - SD_j)^2$, where CD=*CandiData*, SD=*SPData*, K – *dimension.*

The Figure 6 shows the PE structure. The parallelism degree depends on the data width and the SDRAM bandwidth. DCU array are composed with k subtractions (adder logic), k multipliers and one adder to get a distance.

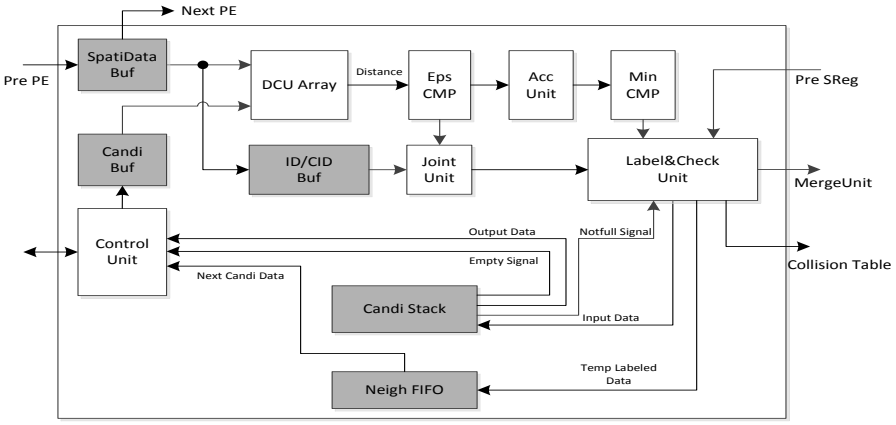

FIGURE 6 Architecture of PE

This is a three stages pipeline structure that can produce distance every cycle, and the multiplier is used IP core provided by Xilinx. For other module, Eps comp module, Acc Unit, MinComp, LabelCheck Unit are respectively complete the function of S22 to S23 in Algorithm 2.

Algorithm2. PE Array Expand Algorithm in each PE

```
Parallel ExpandChain in each PE:
Input: Seed ,PointSet ,ε ,Minpts
Output: Clusterd PointSet ,CollisionSignal
-------------------------------------------------------
ExpandKernel(Seed,PointSet,TCID)
Initi phase:
S1:P=PointSet[][];NeighborCount =0;
Calc phase:
S2:For i=0 to sizeof(PointSet)  step DCUArraySize Do:
    S21:DCUArray  parallelly Do:
          Q=PointSet[i];
          d=Distance[seed][Q];
    S22:EpsCMP and AccUnit parallelly Do:
          if d ≤ ε then
             h=atomicAcc(NeighborCount);
          end if
    S23:if h ≥ Minpts then
            if(Stack not full)
               store the ID of Q to CandiStack;
            else
                mark the stackbit of point with 1;
            end if
             CheckandMark(Q,TCID);
        else
             store the ID of Q to NeighborStack;
    end if
  end for
  S24:if h ≥ Minpts then
         mark seed as  CorePoint;
       else
         mark seed as Noise;
S3:while(CandiStack not empty) do:
         newseed = pop(CandiStack);
         send newseed to controllel unit;
         repeat S1-S2;
    end while
```

ID/CID Buf is to buffer the property data before put them into LabelCheck Unit in pipeline. Candi Stack store the effective point, but limited to the capacity of BRAM on FPGA, the stack bit in overflow data must be marked as 1 to indicate this point is belong to these PE, and will be write to the DDR3 SDRAM. The detail executed process of PE is shown in Algorithm 2.

5 Experiment and Evaluation

5.1 PERFORMANCE MODEL

The total run time is composed of data transfer between host pc and accelerator, clustering time on FPGA and the time of merging TC into complete cluster through collision signal. And run time of clustering on FPGA is the most consuming time part.

Performance Model:

Number of PE: p
Frequency of FPGA: f MHz
Data amount: M points
Data width per each point: w Byte
Input Bandwidth for each PE: *bandwidth* Byte/s
Pipeline Depth of PE:D

$$t_{actual\_comp} = (floor(\frac{M}{p}) * (\frac{2*m*w}{bandwidth} + D + 2*P)) * \frac{1}{f}$$

This is an ideal model, P is the number of PE, and M is the amount of SPData. Data width for each SPData is w, and B is input bandwidth of PE#1. And f is frequency of FPGA. D is the depth in single PE. The DBSCAN hardware architecture was implemented on a Xilinx ML605 board, which includes a Virtex-6 LX240T FPGA, a 512MB DDR3 SDRAM. The peak bandwidth of SDRAM can achieve 6.4GB/s.

5.2 COMPARISON WITH PC

In order to evaluate the DBSCAN accelerator performance, we run sequential DBSCAN algorithm on Intel Corei7 920 quad-core CPU, 2.7GHz and 4GB RAM. The test data set is from SpatialDataGenerator[17]. We generate 10 groups 2 dimension data set from 100k to 1000k points, each data set contains 10 clusters, Minpts is 4, Eps is 0.02. The software implementation is written in C. For the simple control of SDRAM access, frequency

of FPGA is set to 200MHz (maxim speed shows in Table1).

TABLE 1 Resource utilization

Data Dim	Number of PE	Slices	BRAM	Frequency
2	40	32445(85%)	14400Kb(96%)	260.586MHz
8	20	30300(80%)	14400Kb(96%)	197.570MHz

Thus, PE#1 can receive 256bits in one cycle since single 2-dimension SPData is 64bits. LX240T can support 40 PEs for 2-dimension SPData as Table1 show. Runtime comparison as Figure7 show and the average speed up over CoreI7 is 86x to 72x with 40 PEs by our accelerator.

5.3 COMPARISON WITH GPU

Although the bandwidth of GDDR3 is ten times of DDR3, and the frequency of GPU is higher than FPGA, we also compared the parallel DBSCAN performance between GPU and FPGA. The test data set is provided by the author of [6], which is 8-dimension data and single SPData is 256 bits. Frequency of FPGA is set to 100MHz in order to get 2 SPData from SDRAM in one cycle. Runtime is shown in Figure 8, the average speed up over GPU is 2.9x to 1.5x with 20 PEs. Meanwhile, the power consumption of GTX280 is 236w, and the Xilinx ML605 FPGA board is 15w. So our accelerator has a far better performance per Watt.

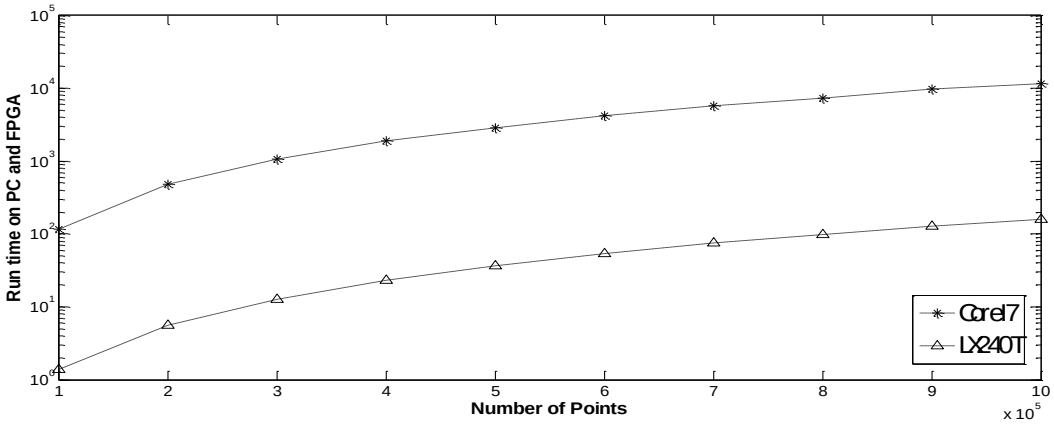

FIGURE 7 Runtime Comparison with Corel7

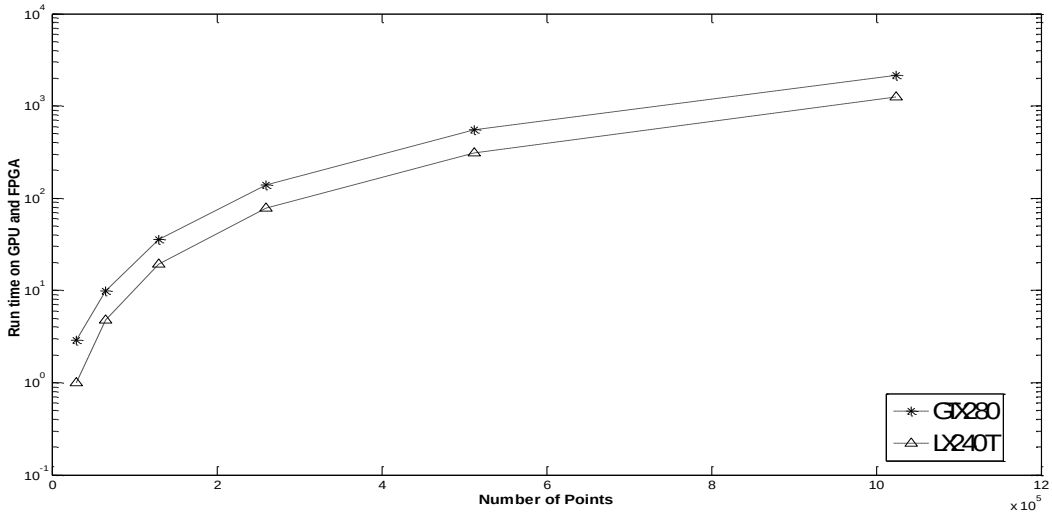

FIGURE 8 Runtime comparison with GTX280

6 Conclusions

In this paper, we have designed the hardware architecture for accelerating DBSCAN, which is a commonly used data mining algorithm. According to the deficiency of previous work, we propose a more robust collision check mechanism in parallel DBSCAN algorithm. Hardware architecture we designed is based on the task-level and data-level parallelism. We have mapped this architecture to a FPGA-based accelerator. When compared with the PC and GPU implementation, experiment results show that our architecture can yield up to 86x and 2.9x speedup respectively.

References

[1] Ester M, Kriegel H P, Sander J, Xu X 1996 A densitybased algorithm for discovering clusters in large spatial databases *Knowledge Discovery and Data Mining (KDD-96)* http://dns2.icar.cnr.it/manco/Teaching/2005/datamining/articoli/KDD-96.final.frame.pdf
Xu X, Jager J, Kriegel H P 2002A fast parallel clustering algorithm for large spatial databases *High Performance Data Mining* 263-290 http://dl.acm.org/citation.cfm?id=383194

[2] Arlia D, Coppola M 2001 Experiments in parallel clustering with dbscan *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, ser. Euro-Par '01. London* UK: Springer-Verlag 326–31

[3] Brecheisen S, Kriegel H-P, Pfeifle M 2006 Parallel Density-Based Clustering of Complex Objects *Advances in Knowledge Discovery and Data Mining, Lecture Notes in Computer Science* **3918** 179-188

[4] He Y, Tan H, Luo W, Mao H, Ma D, Feng S, Fan J 2011 MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce *The 17th IEEE International Conference on Parallel and Distributed Systems (ICPADS)* 473-80

[5] Bohm C, Noll R, Plant C, Wackersreuther B 2009 Density-based Clustering using Graphics Processors *Proceedings of the 18th ACM conference on Information and knowledge management* 661-70

[6] Cope Band Cheung P Y K, Luk W, Howes L 2010 *IEEE Transactions on Computers* **59**(4) 433-48

[7] Asano S, Maruyama T, Yamaguchi Y 2009 Performance comparison of FPGA, GPU and CPU in image processing *IEEE International Conference on Field Programmable Logic and Applications 2009 (FPL'2009)* 126-31

[8] Estlick M, Leeser M, Szymanski J, Theiler J 2001 Algorithmic Transformations in the Implementation of K-means Clustering on Reconfigurable Hardware *Proceedings of the Ninth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2001 (FCCM '01)*

[9] Choudhary P, Choudhary A 2005 Design of a hardware accelerator for density based clustering applications. *Proceedings of the International Conference on Application-specific Systems, Architectures and Processors (ASAP), July 2005*

[10] Narayanan R, Honbo D, Zambreno J, Memik G, and Choudhary A 2007 An FPGA Implementation of Decision Tree Classification *Proceedings of the IEEE International Conference on Design, Automation and Test in Europe (DATE), April 2007*

[11] Baker Z K, Prasanna V K Efficient 2005 Hardware Data Mining with the Apriori Algorithm on FPGAs *Proceedings of the Thirteenth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2005 (FCCM '05), 2005*

[12] Baker Z, Prasanna V 2006 An architecture for efficient hardware data mining using reconfigurable computing system *The 14th Annual IEEE Symposium on Field Programmable Custom Computing Machines 2006 (FCCM'06) 2006*

[13] Wen Y-H, Huang J-W, Chen M-S 2008*IEEE Trans. Knowledge and Data Eng.* **20**(6) 784-95

[14] Sun S, Zambreno J 2011 *IEEE Transactions on Parallel and Distributed Systems* **22**(9) 1497-505

[15] Choudhary A N, Honbo D, Kumar P, Ozisikyilmaz B, Misra S, Memik G 2011 Accelerating data mining workloads: current approaches and future challenges in system architecture design. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1(1) 41-54

[16] http://www.rtreeportal.org/ Oct 2013

Authors

Shaobo Shi , 08 1985, Beijing, China

Current position, grades: Ph.D
University studies: University of Science and Technology Beijing
Scientific interest: Micro architecture, VLSI design, Data mining, FPGA accelerator design
Publications: Shaobo Shi, Yue Qi, Qin Wang. Accelerating Intersection Computation in Frequent Itemset Mining with FPGA. The 15th IEEE International Conference on High Performance Computing and Communications, 2013.

Yue Qi, 11 1975, Beijing, China

Current position, grades: Ph.D, Lecturer
University studies: University of Science and Technology Beijing
Scientific interest: Micro architecture, VLSI design, Wireless Sensor Networks

Qin Wang, 01 1961, Beijing, China

Current position, grades: Ph.D, Professor
University studies: University of Science and Technology Beijing
Scientific interest: Micro architecture, VLSI design, Wireless Sensor Networks