ARC 2014: A Multi-Dimensional FPGA-Based Parallel DBSCAN Architecture

NEIL SCICLUNA AND CHRISTOS-SAVVAS BOUGANIS, Imperial College London

Clustering large numbers of data points is a very computationally demanding task that often needs to be accelerated in order to be useful in practical applications. This work focuses on the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm, which is one of the state-of-the-art clustering algorithms, targeting its acceleration using an FPGA device. The paper presents an, optimised, scalable and parameterisable architecture that takes advantage of the internal memory structure of modern FPGAs in order to deliver a high performance clustering system. Post-synthesis simulation results show that the developed system can obtain mean speed-ups of 31x in real-world tests and 202x in synthetic tests when compared to state-of-the-art software counterparts running on a quad-core 3.4 GHz Intel i7-2600k. Additionally, this implementation is also capable of clustering data with any number of dimensions without impacting the performance.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Algorithms implemented in hardware*; *Gate arrays*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Clustering, DBSCAN, FPGA, Parallel Hardware Architectures

ACM Reference Format:
Neil Scicluna and Christos-Savvas Bouganis, 2014. ARC 2014: A Multi-Dimensional FPGA-Based Parallel DBSCAN Architecture. *ACM Trans. Reconfig. Technol. Syst.* 1936, 7406, Article ARTICLENO (January 2015), 15 pages.
DOI:http://dx.doi.org/10.1145/2724722

1. INTRODUCTION

Clustering is the task of intelligently grouping data points into groups or clusters, where the grouping of the points is based on a particular criterion, such as distance. Clustering has many applications including data mining, statistical data analysis, pattern recognition and image analysis [Martin Ester et al. 1996; Daszykowski et al. 2001; Thapa et al. 2010]. Various clustering algorithms have been developed so far, usually targeting a specific domain of applications by defining the notion of a cluster accordingly. With high complexity and long computation times, sometimes even taking hours for large datasets [He et al. 2011], the need to perform clustering as fast as possible is becoming more and more prevalent.

The most widely used clustering algorithms are K-Means [Hartigan and Wong 1979], Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [Martin Ester et al. 1996] and Ordering Points To Identify the Clustering Structure (OPTICS) [Ankerst et al. 1999]. While K-Means provides a fast solution to the clustering problem, it has been shown to have certain limitations. These include its inability to identify and reject noise in the data, as well as its failure to take into account the spatial density of the clustered data points [Martin Ester et al. 1996]. Additionally, the result

Author's addresses: N. Scicluna and C.-S. Bouganis, Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, United Kingdom.

1936-7406/2015/MonthOfPublication-ARTICLENUMBER
DOI:http://dx.doi.org/10.1145/2724722

of K-Means is heavily dependent on its initialisation and on the number of clusters provided by the user [Hartigan and Wong 1979]. The DBSCAN and OPTICS clustering algorithms address those limitations in exchange for higher complexity.

These algorithms perform clustering using the spatial density of the data. However, OPTICS does not actually perform clustering, but instead provides insight on how to do this, thus requiring additional processing for clustering to be achieved [Ankerst et al. 1999]. This makes the whole process more computationally demanding. On the other hand, DBSCAN is faster than OPTICS [Martin Ester et al. 1996; Ankerst et al. 1999] and serves as a good middle-ground, thereby making it one of the most popular and heavily cited clustering algorithms [Microsoft 2014].

With increasing needs to perform clustering on large datasets as fast as possible, running these on generic processors is proving to be inadequate and specialised hardware is often utilised. FPGA-based implementations of clustering algorithms have been developed to tackle these problems and show very promising results. Some examples of this are the real-time K-Means implementation proposed in [Maruyama 2006] and [Hussain et al. 2011], where these proved to be both faster and more power efficient than GPU implementations. Similarly [Winterstein and Constantinides 2013] and [Annovi and Beretta 2010] describe two approaches which involve using kd-trees and a sliding window consecutively to exploit the parallelism available on FPGAs.

Very few hardware based implementations of the more complex and powerful, density-based clustering methods have been developed thus far. This field therefore, is not very mature and investigating and providing alternate ways to perform density-based clustering in real-time could open a vast array of possibilities. The two algorithms which satisfy this criterion are DBSCAN and OPTICS where as previously mentioned, the latter has disadvantages which make it sub-optimal for real-time applications.

In this paper, an FPGA-based hardware implementation of the DBSCAN algorithm is described. The proposed design takes advantage of the dynamic and massively parallel nature of an FPGA device by performing only certain stages of the algorithm in parallel. This way, performance gains are still achieved, but the on-chip resources required are minimised. The proposed architecture is highly scalable to a degree that the performance gains are not limited by dataset size, but only by the resources available on the FPGA device utilised. Furthermore, the system is designed as a fully parameterisable IP core where aspects such as the size and dimensions of the input data, internal precision, pipeline depths and the level of parallelism, can all be modified by simply altering the parameters and re-synthesising. Finally, the system is also FPGA target independent, with the only requirement being that the chip used has sufficient amounts of Block Random Access Memory (BRAM). All these aspects make this the most flexible hardware implementation of DBSCAN yet.

The research presented in this paper extends the previous work in [Scicluna and Bouganis 2014]. We include more in depth coverage on the various research performed in this field, a more detailed background of the DBSCAN algorithm itself and more detailed explanations on the design choices made. Additionally, the challenges and methodologies used to model, implement and simulate this design in VHDL are also covered extensively. Furthermore, a thorough analysis of the performance of the system when clustering datasets of more than two dimensions is performed. Finally, the implementation of another FPGA-based DBSCAN algorithm [Shaobo Shi and Wang 2014] is investigated and some other performance and resources considerations are discussed.

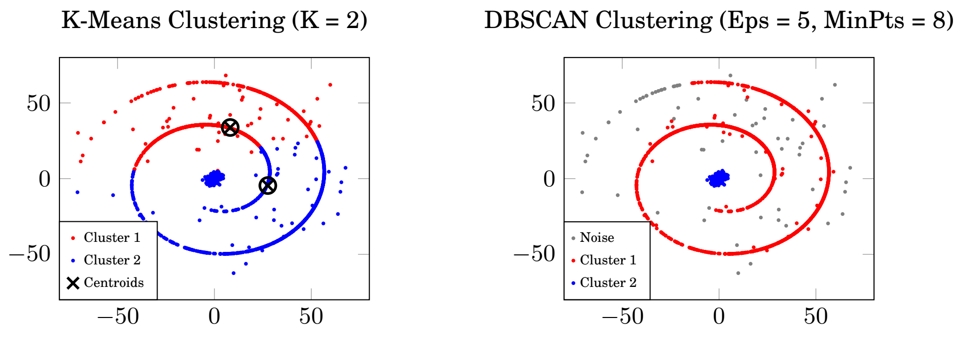

Fig. 1. Comparison between clustering results of K-Means and DBSCAN

2. BACKGROUND

2.1. DBSCAN Algorithm

The DBSCAN algorithm performs clustering based on the spatial density of the data points. This approach to clustering is intuitive, as the definition of a cluster simply refers to a region where there is a typical density of points which is considerably higher than the outside region of the cluster. Additionally, the density in areas where points can be considered as noise, is lower than those of clusters.

The key concept is that in order to form a cluster, there must exist at least $MinPts$ data points that are all within the Eps radius of each other. The $MinPts$ and Eps are user specified parameters. Data points which contain at least as many points in their Eps neighbourhood as $MinPts$, are considered as core points. If a point contains fewer points than $MinPts$ in its neighbourhood, but contains at least one core point, it is considered as a border point. In [Martin Ester et al. 1996], this point is said to be *directly density-reachable* from a core point, but not the other way around. The cluster is then expanded by grouping all the *directly density-reachable* core points and the respective border points. This is referred to as *density reachability* and essentially means that there is a chain of *directly density-reachable* points connecting two particular points. Finally, points which are neither *directly density-reachable*, nor contain at least as many points as $MinPts$ in their neighbourhood, are considered as noise.

Figure 1 highlights the advantages that DBSCAN has over standard K-Means algorithm when attempting to find non-linearly separable clusters. Additionally, with K-Means, the number of clusters is ideally known a-priori as the result might be otherwise unsatisfactory. In order to circumvent this issue various techniques are adopted. One could either determine the number of clusters using Silhouettes beforehand as shown in [Llet et al. 2004], or alternatively, the algorithm can be run a number of times to minimise a unit of error. This however, proves to be quite expensive in terms of complexity and processing [Vattani 2011]. On the other hand, DBSCAN does not require these extra steps, as the clusters are determined by density. In this particular example, DBSCAN correctly identifies that the dataset has 2 clusters and that some points can be treated as noise. The inherent noise rejection in DBSCAN is based on the information provided by the parameters $MinPts$ and Eps.

The algorithm itself works as follows. The first step, is to retrieve all the *directly density-reachable* points with respect to Eps for each point. If there are less points than $MinPts$, the algorithm moves to the next point, otherwise, the points are assigned to the current cluster (as defined by the cluster identification number). The points obtained in this initial step are referred to as the *immediate neighbourhood points*. The next step is to expand the cluster by pushing all the points retrieved onto a queue.

On each iteration, a point is dequeued and all the *density-reachable* points with respect to Eps from that point are retrieved. If the number of points is larger than or equal to $MinPts$, then these points are added to the cluster and pushed onto the queue. These are referred to as the *extended neighbourhood points*. Subsequently, as more *density-reachable* points are found, they are added to the queue to find other points which form part of the cluster. This is repeated until the queue is empty, which signifies that the cluster has been formed completely. The cluster identification number is then incremented and a new point is loaded to start compiling a new cluster. This whole process is repeated until all the points in the dataset have been checked.

It should also be noted that DBSCAN also works for multiple dimensions without changes to the core algorithm. This is because the only operation performed on the data is distance measurement, which can be adapted to multiple dimensions. Furthermore, interchanging distance functions such as Euclidean distance and Manhattan distance is also possible. Such changes impact the shape and radius of points considered in the neighbourhood.

2.2. Related Work

The time complexity of the standard DBSCAN algorithm is $O(n^2)$ (where n is the number of points in the dataset) since a range query, which is done by calculating and checking the distance to all the other points, needs to be performed for each point in the dataset. To improve this, tree data structures such as the R*-Tree [Beckmann et al. 1990] used in [Martin Ester et al. 1996], are adopted to accelerate region queries, thereby reducing the time complexity to $O(n * log(n))$. This however adds the requirement of constructing the tree for the dataset, where the insertion strategy is $O(n * log(n))$. Moreover, spatial accesses using an R*-Tree are not always efficient [Chen et al. 2010].

The Parallel-DBSCAN (P-DBSCAN) algorithm described in [Chen et al. 2010], adopts a different spatial index called the Priority R-Tree (PR-Tree). Here a form of parallelism is introduced where the database is first separated into several parts and then, the computational nodes build their own PR-Tree and carry out the clustering independently. Each node in this system is a desktop PC. Finally, the results are aggregated. An alternative approach to parallelism, but on the same platform, is taken in MapReduce-DBSCAN (MR-DBSCAN) [He et al. 2011] and Hierarchical-Based DBSCAN (HDBSCAN) [Li and Xi 2011], where a map-reduce structure is implemented to spread the computation across multiple nodes that can work in parallel using the Hadoop platform [White 2009]. These implementations all aim to solve the problem of very large and multi-dimensional data clustering. Even though significant performance increases over standard implementations are achieved for datasets with hundreds of thousands of points and more, this is not true for smaller datasets due to the overhead introduced. As a result, these methods are suitable only for certain cases and are still dependent on how fast each individual node can perform the clustering.

Thapa et al. [Thapa et al. 2010] propose a Graphics Processing Unit (GPU) implementation of the DBSCAN algorithm that takes advantage of the large amounts of memory and processor cores available on modern GPUs. Two different approaches are explored in attempt to accelerate this algorithm through parallelism. The first involves computing the region query for each point by comparing it to all the other points in the database in parallel and subsequently, storing all the results in memory. The second approach involves computing the range queries of all the points in parallel and once again storing the results in memory. A different approach is proposed in [Andrade et al. 2013] called G-DBSCAN. This system manages to extract a very significant amount of parallelism by indexing the data using graphs. These are constructed in parallel and subsequently, a breadth-first search is performed to identify the clusters in parallel as well. The fastest known implementation thus far, is the dedicated hardware Very-Large-Scale Integration (VLSI) architecture proposed in [Shimada et al. 2013]. This design however can only perform 2D clustering and is therefore very application specific. The hardware architecture is designed in such a way that there is a processing element for each pixel and thus full pixel-parallel processing is achieved. This results in fast clustering speeds, but requires a significant amount of area per pixel and is therefore infeasible, even for moderately large datasets, particularly when interconnect requirements are considered.

To the best of the authors' knowledge, there has been only one other FPGA implementation of DBSCAN developed since [Scicluna and Bouganis 2014]. This is the work proposed in [Shaobo Shi and Wang 2014], which also proposes an FPGA-based parallel architecture, but using a notably different parallelisation strategy. Shi et al. divide the dataset into smaller datasets depending on the number of available parallel elements (PEs) synthesised. Their architecture then performs clustering on these individual chunks and checks in real-time for any collisions. A collision would mean that points being clustered by two separate PEs actually form part of the same cluster. Both the input data and results are stored in SDRAM, whilst internal first in, first out (FIFO) memory blocks are used for interfacing with the external memory and for storing the immediate and extended neighbourhood points. The FIFOs used for clustering therefore exist within the PEs and are replicated accordingly for parallelism. In this architecture, Euclidean distance is used for determining the size of the clusters and therefore, this requires the use of both multipliers and adders in each PE. Since each PE needs to not only calculate the distance and temporarily store the points, but also check the collision table, merge if necessary and control this whole process, a significant amount of complexity per PE is introduced, particularly in terms of resources used. The implementation was then compared to a Core i7 920 CPU and an Nvidia GTX280 GPU where speed-ups of up to 86x and 2.9x respectively, were measured when clustering synthetic datasets.

In this work, we aim to achieve the performance benefits available through parallelism and hardware implementation, while also maintaining a great level of flexibility. The key contributions are, the analysis and development of a novel parallelisation strategy for the DBSCAN algorithm and the design of a high performance, parameterisable and multi-dimensional, FPGA-based implementation of said algorithm.

3. CONCEPT AND ARCHITECTURE

The major contributor to the time complexity of the algorithm is the range query process that needs to be performed for every point in the dataset. This was also confirmed experimentally through extensive profiling of a custom DBSCAN MATLAB implementation based on [Daszykowski et al. 2001]. While using R*-Trees does indeed improve the time complexity of the algorithm, simulations performed using Elki [Achtert et al. 2013] show that having to reconstruct the tree for each new dataset poses significant performance impact for real-time applications. Furthermore, due to their complexity and highly dynamic nature, R-Trees (which are very similar to R*-Trees), are shown to be costly in terms of resources to implement efficiently in hardware [Xiao et al. 2008]. Thus the proposed architecture utilises standard indexing instead of a tree based data structure.

For DBSCAN to perform the clustering, two sets of range queries are performed, the first obtains the immediate neighbourhood of the particular point where the second set performs the range queries in order to obtain the extended neighbourhood of points for that cluster. In most cases, this second batch of range queries takes the longest portion of execution time. This was corroborated by performing profiling tests in MATLAB with multiple datasets and also varying parameters. The datasets used are data points

Table I. DBSCAN Range query bias analysis

Dataset No.	Size	Eps	MinPts	Imm. Neighb. Range Queries	Ext. Neighb. Range Queries
1	19504	25	10	238	19342
1	19504	25	80	2657	16995
1	19504	55	220	1903	17731
2	2015	25	8	346	1710
3	6472	70	100	193	6310
4	2237	100	90	176	2094
5	2927	80	100	96	2832
6	2003	100	60	2	2003

obtained from corner detection in an image using a Harris Corner Detector [Harris and Stephens 1988] and were originally used in a Foveated Vision Processing application. These datasets are of varying spatial densities and consist of coordinates of local features detected in a variety of images. These results are shown in Table I.

These extended neighbourhood range queries have no data dependencies and thus can be performed in parallel. Furthermore, the algorithm performs these queries by essentially having a queue of points on which these range queries need to be performed. Throughout runtime, this is constantly appended with new points and thus, further parallelisation can be extracted. This is achieved by loading all the points in the queue at each iteration and subsequently, performing all the range queries for these points concurrently. This reduces the computation time significantly.

Figure 2 shows a hardware architecture diagram for the design outlining all the key modules. The input data is assumed to be stored in the Input Memory element which is not internal to the DBSCAN IP Core. Having the input data stored externally allows for maximum flexibility, but in turn, some specifications are enforced with regards to how the input data is supplied to the DBSCAN IP Core. It is assumed that all the dimensions of the currently addressed data point are available simultaneously. Using an FPGA device, this can be achieved by appropriately configuring the BRAM to certain widths or by using multiple BRAM blocks and spreading the data across these blocks. If external SDRAM is used for the input memory, care has to be taken that the available bandwidth satisfies this criterion. Furthermore, it is assumed that the memory subsystem can provide a data point every clock cycle, which is possible through pipelining or using SRAMs. In the case that SDRAM is used for the input memory, pipelining will most probably be required in order to maintain high clock speeds. In the diagram, a multiplexer which is external to the DBSCAN IP Core, is also shown. This makes it possible to shift the control of the Input Memory address from the DBSCAN block to the external system in order to output the results. This would only be required if the Input Memory block is single-port as if this were dual-port, this could be done directly. The architecture is designed to accommodate both cases.

The other two RAM elements in the architecture are the Cluster ID Memory and the Visited Flag Memory, which are internal to the IP Core and are implemented in BRAM. The former stores the cluster identification number for each data point, while the latter marks whether a point has been visited.

The immediate neighbourhood range query results are stored in a FIFO memory element which is implemented using the available BRAM/FIFO resources on the FPGA. This serves as the point queue on which the extended neighbourhood range queries are performed. However, this does not store the actual data points, but just their addresses in main memory.

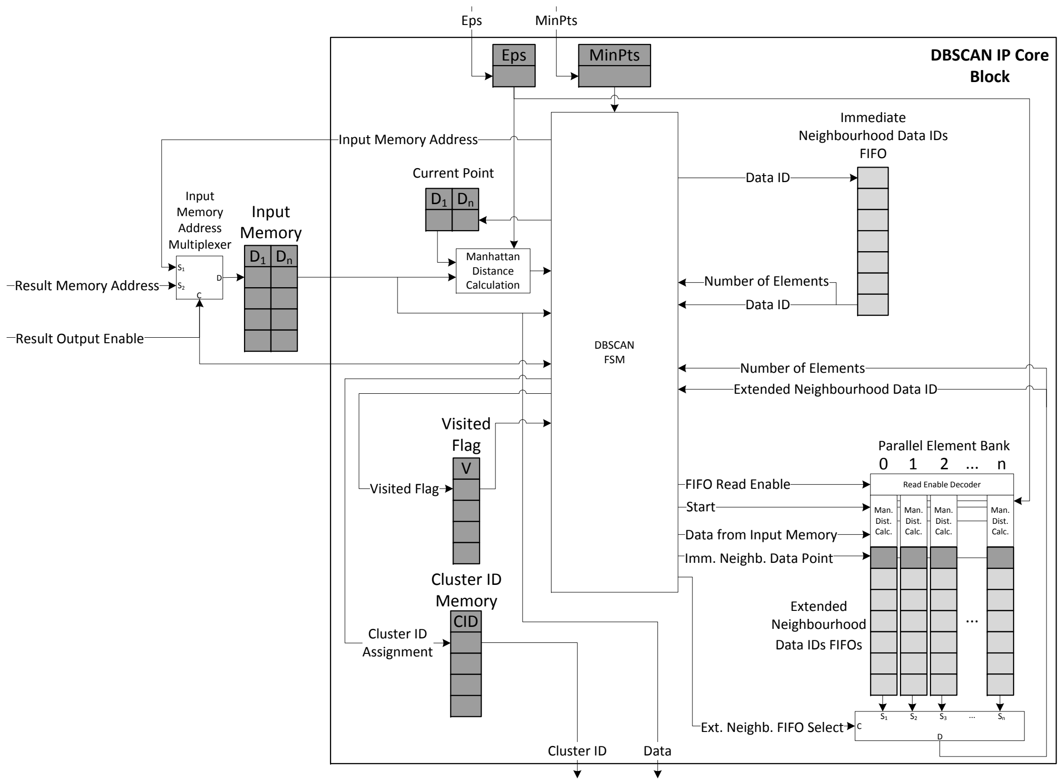

Fig. 2. FPGA DBSCAN Hardware architecture

The main parallelised aspect of the design is the computation of the extended neighbourhood range queries. This is done by performing multiple distance measurements simultaneously by using multiple blocks of the Manhattan Distance Calculation Datapath. This element is automatically generated based on the number of dimensions set and is also pipelined for high clock frequency. It should be noted as well that increasing the data precision and the number of dimensions, does not have any impact on the latency of the design, as the hardware is reconfigured to perform the calculations in parallel. This component is designed to work as a black box, in the sense that, given two data points and the Eps parameter value, it provides an output signal signifying whether or not the distance between the two points is smaller than or equal to Eps. Since the data point should only be stored into the queue if that condition is satisfied, this output signal is then used as a write enable for the FIFO that stores the result of the extended neighbourhood range query. The combination of a distance calculation block and a FIFO is referred to as a PE. As shown in the diagram, multiple PEs can be instantiated depending on the available resources and the target performance. Once again, each parallel element FIFO stores only the addresses of the coordinates and not the actual data point itself. This is key as it makes the system very scalable, since the amount of BRAM resources required, does not change with the number of dimensions or bit precision. The elements that need to be regenerated when the number of dimensions is increased are; the Input Memory pipeline FIFO, the Manhattan Distance Calculation Datapath and the registers that store the current points that are currently being checked.

While the results of multiple range queries are obtained simultaneously with these PEs, the cluster identification number and visited flag must be updated serially and therefore, there needs to be a way of selecting between the separate parallel elements. This selection is controlled by the Finite State Machine (FSM) and it is crucial that the read enable signal is high for only one FIFO at a time. Correspondingly, the outputs of the FIFOs also need to be considered individually. This is achieved through the use of a decoder and a multiplexer. If the number of parallel elements is fairly large, the multiplexer would have a very long critical path, resulting in lower clock speeds. To remedy this, the multiplexer is pipelined.

To perform the clustering, the DBSCAN FSM iterates through the points in the data memory and if the data point is not marked as visited in the visited flag memory, a range query is performed by checking the distance of that point to all the other points in the dataset. All the points that are within the Eps neighbourhood are then stored in the immediate neighbourhood data ID FIFO. If the number of elements is greater than or equal to $MinPts$, these points are assigned to the current cluster and the FSM then loads into the register bank all available points in the immediate neighbourhood FIFO, or as limited by the number of available PEs. Subsequently this batch of range queries is performed concurrently by loading a new value from the input coordinate memory on each clock cycle and measuring the distance between this point and all the points associated with each PE. In the case that the points satisfy the range query criteria, they are then stored in the respective FIFO. The element count for each FIFO is then checked against $MinPts$ and if the condition is satisfied, these points are added to the immediate neighbourhood queue to continue expanding the cluster. This whole process is repeated until there are no points left in the queue, at which point the next memory element is checked to start forming a new cluster.

With regards to number representation, it was determined that using fixed-point representation was the most appropriate for this implementation. This is due to the massively parallel nature of the architecture, particularly when it comes to the Manhattan Distance Calculation units. Using fixed-point instead of floating-point representation allows these units to be as small and as fast as possible.

4. EXPERIMENTAL RESULTS

A fully parameterisable IP core was developed using VHSIC Hardware Description Language (VHDL) and was designed and synthesised using the Xilinx ISE 14.1 Suite. Designing the system to be fully parameterisable proved challenging. Great care was taken to provide a means to control almost all aspects of the hardware architecture which on synthesis, generates automatically with minimal user intervention. The proposed architecture was synthesised for the Xilinx Virtex 7 XC7VX690T-3 FPGA and subsequently, its performance was evaluated through post-synthesis simulation using the variety of datasets described in Section 3.

Since most of the current works have focused on 2D point clustering, the same approach is followed in this performance evaluation. Figure 3 shows a visualisation of this dataset, with the points to be clustered marked in red. Being pixel coordinates, these datasets are therefore two dimensional and each dimension is stored with 16 bit precision. Maximum clock speeds reported by the synthesis tool for this design with 2D input data and 1–710 PEs, range from 350–410 MHz. The post-synthesis simulation results are then compared to a range of software methods run on a desktop computer with an Intel Core i7 2600K 3.4 GHz Sandy Bridge processor and 16 GB of DDR3-1066 MHz memory. The evaluated design instance targets an image processing application, however, the proposed architecture can be configured to handle multi-dimensional data of any word length without any impact on the latency of the system, provided the target FPGA is large enough. In the current implementation, it

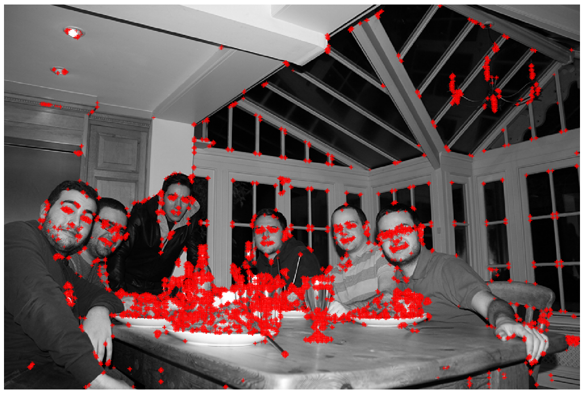

Fig. 3. Dataset 1 points visualised in red.

Table II. Effects of varying number of PEs (Synthesis Results)

No. of PEs	Max Clock Speed (MHz)	LUT Utilisation (%)	BRAM/FIFO Utilisation (%)	Total Power (W)
1	409	0.1	1	0.57
25	413	1	5	0.84
50	393	2	8	1.03
100	395	4	15	1.75
150	391	6	22	2.28
300	378	12	42	3.90
500	372	20	69	6.92
710	353	34	98	8.97

was assumed that the data points exist already in the FPGA. The proposed system instantiated in the target FPGA can accommodate a number of points which is more than that required by most applications.

Table II shows the resource utilisation along with the maximum clock speed and respective power consumption for varying numbers of PEs. The power consumption is estimated using the Xilinx XPower Analyzer tool and includes both the dynamic and static power. From this table we can determine that with 16 bit precision and 2D data, the design occupies a minimum of approximately 0.1%, with each PE taking up 0.042% of slice and Look-Up Table (LUT) area. BRAM utilisation is solely dependent on the configuration for that particular application and whether or not the input memory is stored in BRAM. The array of parameters available provide full control over the sizes of each element and therefore the amount of BRAM resources used can be calculated.

The proposed system was compared against three software implementations. The first of these is the MATLAB implementation, while the second and third were measured by ELKI [Achtert et al. 2013], which is a software clustering algorithm performance analysis tool written by the developers of DBSCAN. One of these measurements is with standard indexing, whereas the other was done using R*-Tree indexing. To provide a fair comparison, the R*-Tree timing includes both the time taken for the generation of the tree data structure and the execution of the DBSCAN algorithm accelerated by that data structure. To provide a worst-case comparison, the software execution time shown for each case is the shortest one between three software implementations. The results for the tests performed, along with the respective chosen parameters are shown in Table III. The Eps and $MinPts$ parameters in these tests were primarily chosen to provide meaningful results for the targeted image processing

Table III. FPGA DBSCAN Implementation performance analysis results

Dataset No.	No. of Points	Eps	MinPts	Parallel Elements (PEs)	Clock Speed (MHz)	Software Time (s)	FPGA Time (ms)	Speed-Up
1	19504	25	80	300	377.98	7.16	211.88	33.77
1	19504	55	220	710	353.15	11.4	371.51	30.68
2	2015	25	8	85	405.48	0.13	2.77	46.21
3	6472	70	100	150	391.11	1.74	49.56	35.11
4	2237	100	90	150	391.11	0.28	9.76	28.69
5	2927	80	100	150	391.11	0.58	22.34	26.14
6	2003	100	60	150	391.11	0.19	8.64	22.45

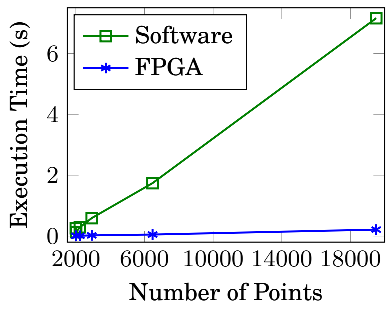

Fig. 4. Graph showing performance scalability with increasing numbers of points

application. The geometric mean of the speed-ups achieved is 31x. Similarly, Figure 4 shows how the performance of the fastest software version and the proposed FPGA design scale with increasing dataset sizes.

It should be noted that the results achieved with the proposed FPGA implementation for the datasets tested, were identical in terms of quality when compared to the software counterparts. The two main reasons for this are that, no changes were made to how the algorithm processes the data and that the bit precision used for the data and internal core components was sufficient to have no deviations in the final result.

The conducted experiments show that there is an upper limit as to how much parallelism can be extracted from a dataset and this is dependent on the combination of the spatial density of the data and the parameters used. This is due to the fact that for each point checked, there is only a limited number of points returned with the immediate neighbourhood range query. Increasing the number of PEs beyond this number would not provide additional performance benefits. In spite of this, the results show that the proposed parallelisation strategy proves significantly beneficial. The amount of parallelism available also increases with dataset size. The choices of the numbers of PEs used in the tests shown in Table III were made to ensure that the number of range queries performed in parallel are maximised for each dataset. This is based on a MATLAB model developed to analyse the performance effects of varying numbers of PEs on the system. This MATLAB model essentially runs the algorithm and keeps track of the number of equivalent cycles that the proposed design would require to perform the same task. The number of PEs is a parameter which can be specified such that the neighbourhood range queries performed in parallel are not counted. Additionally, the model also calculates the number of PEs required to take full advantage of the parallelism available in a particular dataset.

The results show that for the range of datasets and parameters tested, which provide a wide range of test cases with varying spatial density, significant performance can be

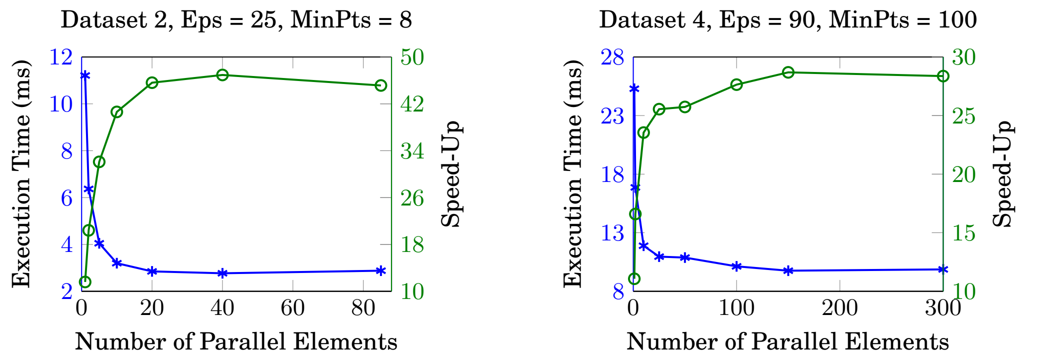

Fig. 5. Execution time (marked by $*$) with varying number of parallel elements and respective speed-up (marked by o)

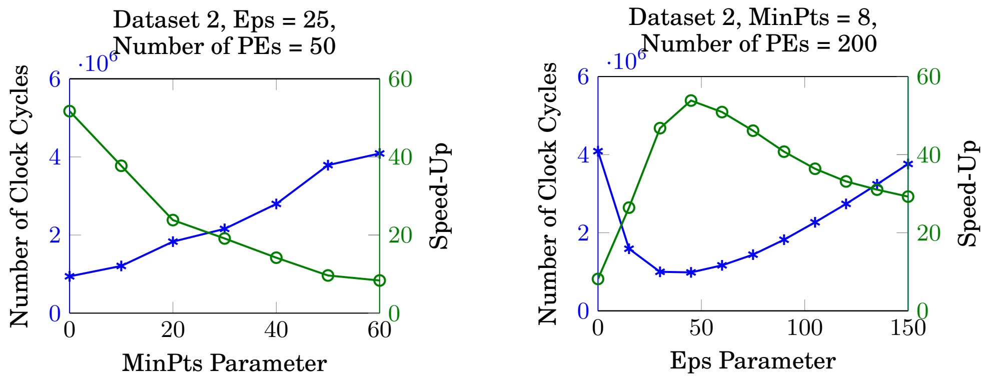

Fig. 6. Number of clock cycles taken (marked by $*$) to compute clustering with varying $MinPts$ (a) and Eps (b) and respective speed-up (marked by o) over software implementation

exploited even with a small number of PEs. Figure 5 shows how this applies to two particular test cases. As shown in Table III, the maximum clock speed that the design can be run at, varies with the number of PEs. This is mainly due to the longer critical paths introduced and partly also due to the fact that the synthesis tool cannot optimise as effectively when more resources are used. The FPGA times reported in the results were measured with the design running at the maximum clock speed attainable with the respective number of PEs.

Figure 6(a) shows that the time taken to cluster the dataset increases approximately linearly with the $MinPts$ parameter. This is due to more parallelism potentially becoming available in the data as this parameter gets smaller. In this case, when $MinPts = 0$ maximum parallelism is achieved, with the result being a single large cluster. On the other hand, when $MinPts = 60$, no clusters are formed and therefore, no parallelism can be extracted. Figure 6(b) shows that the relationship is not linear for the Eps parameter. When the Eps parameter is small we encounter the same issue where only a few small clusters are formed and therefore little performance gain can be obtained through parallelism. As Eps becomes larger, the number of PEs becomes the limiting factor and therefore, more are required to take advantage of the parallelism available in the data. Additionally, as occurs with the standard DBSCAN implementations, with larger Eps the algorithm takes longer to compute.

Table IV. DBSCAN FPGA and GPU performance comparison

No. of Points	Software Time (s)	Sequential Time (s) [Thapa et al. 2010]	GPU Time (ms) [Thapa et al. 2010]	FPGA Time (ms)	Speed-Up over Software	Speed-Up over GPU
5000	0.56	1.33	340	2.6	215.55	130.87
10000	1.69	4.5	1120	8.4	201.14	133.30
15000	3.53	6.82	2490	17.41	202.76	143.02
20000	5.64	9.09	4910	29.62	190.42	165.78

Table V. FPGA DBSCAN multi-dimensional performance with variable size datasets analysis

No. of Dimensions	No. of Points	Software Time (s)	FPGA Time (ms)	Speed-Up
2	200	0.002	0.08	25.14
3	1000	0.067	1.15	58.36
4	5000	1.184	16.19	73.15
5	25000	17.657	204.78	86.22

Furthermore, the proposed system was tested using the datasets that were used in [Thapa et al. 2010], which the authors have kindly provided. It should be noted that unlike the datasets tested previously, these are synthetic datasets and have constant spatial density throughout, with the points ordered by cluster. Table IV shows the execution times of the various implementations with the parameters set to $Eps = 1.5$ and $MinPts = 4$ as used in [Thapa et al. 2010]. The number of PEs was set to 50 as this extracts the maximum amount of parallelism available in the data and allows for a clock speed of 393 MHz. The geometric mean of the speed-ups achieved in synthetic dataset tests is 202x when compared to the software implementations and 143x when compared to the GPU implementation.

To analyse the performance of the design when clustering datasets with an increasing number of dimensions, synthetic datasets similar to the ones used in [Thapa et al. 2010] were generated. These were generated such that they have constant spatial density throughout and organised to form eight clusters with the same Eps and $MinPts$ parameters. Each cluster is made up of five points in every dimension. Consequently, all other effects on performance, except for the number of points, are minimised. Table V shows the simulation results when clustering data in a range from two to five dimensions. The Eps and $MinPts$ parameters were both set to 5 and the number of PEs was set to 50. This results in an estimated maximum clock speed of 395 MHz.

As can be seen from Table V, the execution time of the proposed FPGA implementation is significantly less when compared to the software counterpart. To provide another perspective an experiment was performed to see how the performance changes when the number of points is kept constant and the number of dimensions increase. Table VI shows the results of these simulations. The slight differences in the FPGA times are due to the fact that the cluster sizes differ from one dataset to another and therefore, more parallelism can be extracted in some cases. These results also show that while the number of dimensions does not affect the execution time of FPGA implementation, the execution time of the software implementation increases significantly. Figure 7 show the execution times and speed-ups of both dataset groups. The reason why significant performance benefits are seen with increasing number of dimensions, is all due to the distance calculations being performed in parallel and therefore, not impacting the performance of the algorithm.

Table VI. FPGA DBSCAN multi-dimensional performance with same size datasets analysis

No. of Dimensions	No. of Points	Software Time (s)	FPGA Time (ms)	Speed-Up
2	5000	0.665	3.55	187.08
3	5000	0.826	3.18	260.11
4	5000	1.117	3.00	372.0
5	5000	1.323	3.31	399.65

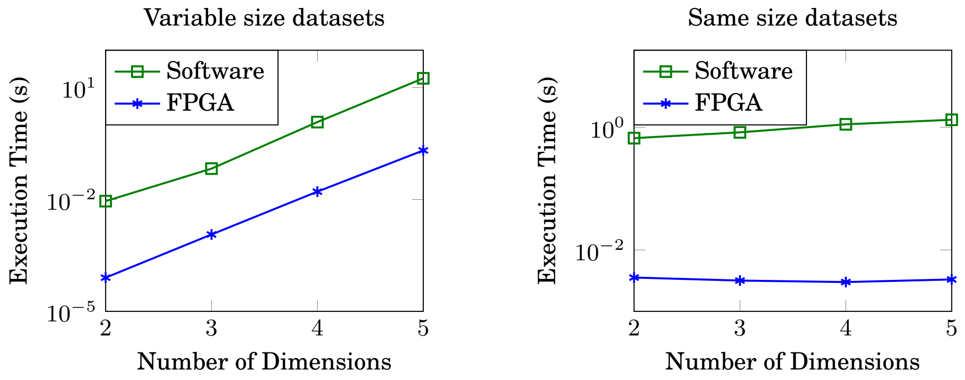

Fig. 7. Performance scalability with number of dimensions

5. PERFORMANCE AND RESOURCE CONSIDERATIONS

As discussed, all experimental results shown are based on post-synthesis simulations of the design. Furthermore, the results were obtained with the assumption that the input data is already in the FPGA itself. Consequently, the execution times shown do not include the time required to transfer the data from a host to the FPGA device as might be required in a real application. This assumption is valid for datasets which fit in the target FPGA BRAM, because as discussed in [Shaobo Shi and Wang 2014], the time taken to cluster the data with DBSCAN is much more time consuming than transferring the data.

On the other hand, when the dataset is significantly larger, transferring all the information to BRAM would not be the best approach. As shown in [Shaobo Shi and Wang 2014] even with a small Xilinx ML605 board, which has 512MB of DDR3 SDRAM, a data transfer rate of 256 bits per cycle can be achieved from external memory. In these cases, keeping the input memory external to the FPGA and transferring one data point with N dimensions per clock cycle as the design expects, would be feasible for most standard applications. For example, this would provide the ability to cluster datasets with 8 dimensions, each of 32-bit precision. Furthermore, with the ability to configure the memory read pipeline depth as a synthesis parameter, this is even less of a problem. Since the input memory is always read sequentially for each range query performed, the latency penalty introduced is very small. Therefore, the performance impact is negligible, especially when considering that the design can be run at a higher frequency once the input memory is pipelined. Simulations of the proposed design show that increasing the pipeline depth from 1 to 2 only increases the computation time by 0.26 ms as operating frequency can be also be increased from 353.47 MHz to 383.09 MHz. This only accounts for a 9.03% increase in computation time when clustering Dataset 1 which has 19504 points.

Other BRAM based blocks in the design are not affected by the data precision or dimensionality of the input data and since they are accessed in a non-sequential manner, it is important that these are internal to the FPGA chip. These elements however are relatively small, as they either store the cluster IDs, a single bit to mark the data point as visited, or store memory addresses. The parameters affecting how many resources these elements consume are the number of data points being clustered and the number of PEs synthesised.

6. FUTURE WORK AND CONCLUSIONS

This paper presents a novel parallelisation strategy for the DBSCAN algorithm and a hardware architecture suitable for FPGAs which based on that strategy. The proposed design utilises key aspects of the FPGA fabric, such as the parallelism and reconfigurability, in order to accelerate the targeted algorithm. Additionally all aspects of the design are highly parameterisable, including the data precision and dimensionality.

Currently the limiting factor of the design is the size of the dataset that can be clustered, as this is directly limited by the resources, particularly the amount of BRAM/FIFOs available on the particular FPGA device used. However, adapting a similar map-reduce structure to the one proposed in [He et al. 2011], where very large datasets are clustered by spreading the computation across multiple computer nodes, could theoretically be adapted for this design. Using this scheme, multiple interconnected FPGAs, each with an adapted version of this proposed implementation, can be used as a single node in a cluster. This would in turn allow for even more parallelism, along with the ability to cluster very large databases. This approach however should be investigated thoroughly as great care needs to be taken to minimise communication overheads as much as possible, which could lead to severe performance degradation.

In conclusion, when compared to established software methods, simulations show that the proposed design achieves considerable performance benefits, which are higher than those obtained using the GPU implementations in [Thapa et al. 2010] and [Andrade et al. 2013]. Additionally, performance gains are achieved even with small datasets, as there is very little overhead with the proposed parallelisation strategy. With regards to the VLSI implementation, while the proposed system cannot match its performance, it is significantly more flexible and allows for clustering of larger datasets with more dimensions, while still providing substantially high performance.

REFERENCES

Elke Achtert, Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. 2013. Interactive Data Mining with 3D-parallel-coordinate-trees. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, New York, NY, USA, 1009–1012. DOI:http://dx.doi.org/10.1145/2463676.2463696

Guilherme Andrade, Gabriel Ramos, Daniel Madeira, Rafael Sachetto, Renato Ferreira, and Leonardo Rocha. 2013. G-DBSCAN: A {GPU} Accelerated Algorithm for Density-based Clustering. *Procedia Computer Science* 18, 0 (2013), 369–378. DOI:http://dx.doi.org/10.1016/j.procs.2013.05.200 2013 International Conference on Computational Science.

Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jrg Sander. 1999. OPTICS: Ordering Points To Identify the Clustering Structure. ACM Press, 49–60.

A. Annovi and M. Beretta. 2010. A fast general-purpose clustering algorithm based on FPGAs for high-throughput data processing. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 617, 13 (2010), 254–257. DOI:http://dx.doi.org/10.1016/j.nima.2009.10.046 11th Pisa Meeting on Advanced Detectors Proceedings of the 11th Pisa Meeting on Advanced Detectors.

Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. ACM, 322–331.

Min Chen, Xuedong Gao, and HuiFei Li. 2010. Parallel DBSCAN with Priority R-tree. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*. 508–511. DOI:http://dx.doi.org/10.1109/ICIME.2010.5477926

M Daszykowski, B Walczak, and D.L Massart. 2001. Looking for natural patterns in data: Part 1. Density-based approach. *Chemometrics and Intelligent Laboratory Systems* 56, 2 (2001), 83–92. DOI:http://dx.doi.org/10.1016/S0169-7439(01)00111-3

Chris Harris and Mike Stephens. 1988. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*. 147–151.

J. A. Hartigan and M. A. Wong. 1979. A K-Means Clustering Algorithm. *Applied Statistics* 28 (1979), 100–108.

Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. 2011. MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*. 473–480. DOI:http://dx.doi.org/10.1109/ICPADS.2011.83

Hanaa M. Hussain, Khaled Benkrid, Ahmet T. Erdogan, and Huseyin Seker. 2011. Highly Parameterized K-means Clustering on FPGAs: Comparative Results with GPPs and GPUs.. In *ReConFig*, Peter M. Athanas, Jrgen Becker, and Ren Cumplido (Eds.). IEEE Computer Society, 475–480. http://dblp.uni-trier.de/db/conf/reconfig/reconfig2011.html#HussainBES11

Lingjuan Li and Yang Xi. 2011. Research on Clustering Algorithm and Its Parallelization Strategy. *2012 Fourth International Conference on Computational and Information Sciences* 0 (2011), 325–328. DOI:http://dx.doi.org/10.1109/ICCIS.2011.223

R. Llet, M.C. Ortiz, L.A. Sarabia, and M.S. Snchez. 2004. Selecting variables for k-means cluster analysis by using a genetic algorithm that optimises the silhouettes. *Analytica Chimica Acta* 515, 1 (2004), 87–100. DOI:http://dx.doi.org/10.1016/j.aca.2003.12.020 Papers presented at the 5th {COLLOQUIUM} {CHEMIOMETRICUM} {MEDITERRANEUM}.

Hans-peter Kriegel Martin Ester, Jrg S, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 226–231.

Tsutomu Maruyama. 2006. Real-time K-Means Clustering for Color Images on Reconfigurable Hardware.. In *ICPR (2)* (2006-09-25). IEEE Computer Society, 816–819. http://dblp.uni-trier.de/db/conf/icpr/icpr2006-2.html#Maruyama06

Microsoft. 2014. Most Cited Data Mining Articles on Microsoft Academic Search. (June 2014). http://academic.research.microsoft.com/RankList?entitytype=1&topDomainID=2&subDomainID=7&last=0&start=1&end=100

Neil Scicluna and Christos-Savvas Bouganis. 2014. FPGA-Based Parallel DBSCAN Architecture. In *Reconfigurable Computing: Architectures, Tools, and Applications*, Diana Goehringer, MarcoDomenico Santambrogio, JooM.P. Cardoso, and Koen Bertels (Eds.). Lecture Notes in Computer Science, Vol. 8405. Springer International Publishing, 1–12. DOI:http://dx.doi.org/10.1007/978-3-319-05960-0_1

Qi Yue Shaobo Shi and Qin Wang. 2014. FPGA based accelerator for parallel DBSCAN algorithm. *COMPUTER MODELLING & NEW TECHNOLOGIES* 18, 2 (2014), 135–142. http://www.tsi.lv/sites/default/files/editor/science/Research_journals/Computer/2014/V2/art20_cmnt1802-45.pdf

A. Shimada, Hongbo Zhu, and T. Shibata. 2013. A VLSI DBSCAN processor composed as an array of micro agents having self-growing interconnects. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*. 2062–2065. DOI:http://dx.doi.org/10.1109/ISCAS.2013.6572278

R.J. Thapa, C. Trefftz, and G. Wolffe. 2010. Memory-efficient implementation of a graphics processor-based cluster detection algorithm for large spatial databases. In *Electro/Information Technology (EIT), 2010 IEEE International Conference on*. 1–5. DOI:http://dx.doi.org/10.1109/EIT.2010.5612134

Andrea Vattani. 2011. k-means Requires Exponentially Many Iterations Even in the Plane. *Discrete & Computational Geometry* 45, 4 (2011), 596–616. DOI:http://dx.doi.org/10.1007/s00454-011-9340-1

Tom White. 2009. *Hadoop: The Definitive Guide* (1st ed.). O'Reilly Media, Inc.

Bayliss S. Winterstein, F. and G.A. Constantinides. 2013. FPGA-based K-means clustering using tree-based data structures. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*. 1–6. DOI:http://dx.doi.org/10.1109/FPL.2013.6645501

Xiang Xiao, Tuo Shi, Pranav Vaidya, and Jaehwan John Lee. 2008. R-tree: A Hardware Implementation.. In *CDES* (2009-12-05), Hamid R. Arabnia (Ed.). CSREA Press, 3–9. http://dblp.uni-trier.de/db/conf/cdes/cdes2008.html#XiaoSVL08