

# Hardware Supported Rough Sets Based Rules Generation for Big Datasets

Maciej Kopczynski, Tomasz Grzes, Jaroslaw Stepaniuk

► **To cite this version:**

Maciej Kopczynski, Tomasz Grzes, Jaroslaw Stepaniuk. Hardware Supported Rough Sets Based Rules Generation for Big Datasets. 15th IFIP International Conference on Computer Information Systems and Industrial Management (CISIM), Sep 2016, Vilnius, Lithuania. pp.91-102, 10.1007/978-3-319-45378-1\_9. hal-01637509

**HAL Id: hal-01637509**

**<https://hal.inria.fr/hal-01637509>**

Submitted on 17 Nov 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



# Hardware Supported Rough Sets Based Rules Generation for Big Datasets

Maciej Kopczynski, Tomasz Grzes, and Jaroslaw Stepaniuk

Faculty of Computer Science  
Bialystok University of Technology  
Wiejska 45A, 15-351 Bialystok, Poland  
{m.kopczynski,t.grzes,j.stepaniuk}@pb.edu.pl  
<http://www.wi.pb.edu.pl>

**Abstract.** In this paper we propose a combination of capabilities of the Field Programmable Gate Arrays based device and PC computer for rough sets based data processing resulting in generation of decision rules. Solution is focused on big datasets. Presented architecture has been tested in programmable unit on real datasets. Obtained results confirm the significant acceleration of the computation time using hardware supporting rough set operations in comparison to software implementation.

**Keywords:** Rough sets, FPGA, hardware, rules

## 1 Introduction

The rough sets' theory was developed in the eighties of the twentieth century by Prof. Z. Pawlak and is an useful tool for data analysis. A lot of rough sets algorithms were implemented in scientific and commercial tools for data processing. Data processing efficiency problem is arising with increase of the amount of data. Software limitations leads to the search for new possibilities.

Field Programmable Gate Arrays (FPGAs) are the digital integrated circuits which function is not determined during the manufacturing process, but can be programmed by engineer any time. One of the main features of FPGAs is the possibility of evaluating any boolean function. That's why they can be used for supporting rough sets calculations.

At the moment there are some hardware implementation of specific rough set methods. The idea of sample processor generating decision rules from decision tables was described in [11]. In [8] authors presented architecture of rough set processor based on cellular networks described in [10]. In [3] a concept of hardware device capable of minimizing the large logic functions created on the basis of discernibility matrix was developed. More detailed summary of the existing ideas and hardware implementations of rough set methods can be found in [4].

None of the above solutions is complete, i.e. creates a system making it possible to solve each problem from wider class of basic problems related to rough sets. Our aim is to create such a system. Authors are working on fully operational System-on-Chip (SoC) including central processing unit based on

Altera NIOS II core implemented in Stratix III FPGA and co-processor for rough sets calculations. More details on previous authors' work can be found in [1, 5–7, 14].

## 2 Introductory Information

### 2.1 Notions

Let  $DT = (U, A \cup \{d\})$  be a decision table, where  $U$  is a set of objects,  $A$  is a set of condition attributes and  $d$  is a decision attribute.

Notions and definitions used in pseudocode shown in next section are presented below for the clarity of understanding:

- $t$  - pair  $(a, v)$ , where  $a$  is a condition attribute and  $v$  is a value for this attribute,
- $[t]$  - set of objects fulfilling single condition  $t = (a, v)$ , i.e.  $[(a, v)] = \{x \in U : a(x) = v\}$
- $[T]$  - set of objects fulfilling every condition  $t \in T$ , i.e.  $[T] = \bigcap_{t \in T} [t]$

### 2.2 Algorithm for Hardware Supported Rule Generation

This section describes pseudocode for an algorithm of rule generation based on LEM2 (**L**earning from **E**xamples **M**odule - version **2**) solution presented by J. W. Grzymala-Busse in [2]. We called this algorithm HRG2-LEM2 (**H**ardware **R**ules **G**eneration version **2** – **LEM2**). There are two main differences between HRG2-LEM2 algorithm and one described by J. W. Grzymala Busse. First is that HRG2-LEM2 is using hardware supported units for operations related to counting the elements as well as creating and checking sets of objects fulfilling a given condition. Second important difference is introduction of input dataset decomposition that allows for processing data by fixed size hardware modules. Further details on hardware part are presented in Section 3. Algorithm parts supported by hardware are prefixed with  $[H1]$  or  $[H2]$  in pseudocode.  $[H1]$  is module for calculating objects classes (unit *dtComparator*), while  $[H2]$  is module for calculating number of objects fulfilling condition  $t = (a, v)$  (unit *avCounter*). Further details are included in Section 3.2. For clarity of understanding, algorithm pseudocode is kept as close as possible to the original one. Authors avoided diving into hardware details, because pseudocode would become hardly understandable. Results generated by hardware blocks and their design is presented in Section 3.2.

#### HRG2-LEM2 Algorithm (Hardware Rules Generation version 2 – LEM2)

**INPUT:** decision table  $DT = (U, A \cup \{d\})$ , precomputed lower approximations set  $LA$

**OUTPUT:** global rules set  $GR$

1:  $GR \leftarrow \emptyset$

```

2: for  $vd \in$  every value of  $d$  do
3:    $lowerApprox \leftarrow LA(vd)$ 
4:    $G \leftarrow lowerApprox$ 
5:    $LR \leftarrow \emptyset$ 
6:   while  $G \neq \emptyset$  do
7:      $T \leftarrow \emptyset$ 
8:      $T_G \leftarrow \{t : [t] \cap G \neq \emptyset\}$ 
9:     while  $T = \emptyset$  or [H1]  $[T] \not\subseteq lowerApprox$  do
10:      [H2]  $t \leftarrow$  choose  $t \in T_G$  with maximum value of  $|[t] \cap G|$ . If more such
       $t$  exist, then choose first  $t$  with minimum value of  $|[t]|$ 
11:       $T \leftarrow T \cup \{t\}$ 
12:      [H1]  $G \leftarrow [t] \cap G$ 
13:       $T_G \leftarrow \{t : [t] \cap G \neq \emptyset\} \setminus T$ 
14:    end while
15:    for  $t \in T$  do
16:      if [H1]  $[T \setminus \{t\}] \subseteq lowerApprox$  then
17:         $T \leftarrow T \setminus \{t\}$ 
18:      end if
19:    end for
20:     $LR \leftarrow LR \cup \{T\}$ 
21:    [H1]  $G \leftarrow lowerApprox \setminus \bigcup_{T \in LR} [T]$ 
22:  end while
23:  for  $T \in LR$  do
24:    if [H1]  $\bigcup_{T' \in LR \setminus \{T\}} [T'] = lowerApprox$  then
25:       $LR \leftarrow LR \setminus \{T\}$ 
26:    end if
27:  end for
28:   $GR \leftarrow GR \cup LR$ 
29: end for

```

Input of the algorithm is decision table  $DT$  and precomputed lower approximations set  $LA$  for every decision class related to value of decision attribute  $d$ . Output is global rule set  $GR$ . Main algorithm loop in line 2 runs for every value  $vd$  of attribute  $d$  because lower approximations were precomputed for every decision class related to values of  $d$  attribute. Given lower approximation is loaded into set  $lowerApprox$  in line 3. Then  $lowerApprox$  is copied to the temporary set  $G$  in line 4. This set is recomputed during rule creation process performed by loop in lines 9 to 14. Local rules set  $LR$  (initialized as empty in line 5) stores rules generated for the given lower approximation (loop in lines 6 to 22). Set  $T$  that can be interpreted as single created consists of conditional parts (pairs  $(a, v)$ ). All possible combinations of conditional parts based on objects in set  $G$ , are created in line 8 and are stored in set  $T_G$ . This is introductory step that leads to definition of all available condition parts of rule created by loop in lines 9 to 14. Every iteration of loop leads to adding new conditional part to the rule. In line 10 best part  $t$  of the rule is chosen according to presented constraint. Line 11 adds part  $t$  to the constructed rule  $T$ . Sets  $G$  and  $T_G$  are recalculated

respectively in lines 12 and 13. Purpose of recalculation is removing objects covered by current rule and limit amount of possible conditional parts for future iterations of loop. Loop in lines 15 to 19 is for cutting the created rule. Cutting means removing redundant condition parts of created rule. Loop iterates over every part of temporary rule  $T$ . If rule  $T$  without its one part  $t$  is still subset of lower approximation *lowerApprox* (line 16), then part  $t$  is removed from rule  $T$  (line 17). New rule  $T$  is added to the local rules set  $LR$  in line 20. Set  $G$  is recalculated in line 21 by using all rules from  $LR$  set. Loop in lines 23 to 27 is used for cutting local rule set  $LR$ . Purpose of this operation is removing redundant rules covering objects that are covered by other rules. Loop iterates over every rule  $T$  stored in  $LR$ . If local rule set  $LR$  without its one part  $T$  is still subset of lower approximation *lowerApprox* (line 24), then part  $T$  is removed from rule set  $LR$  (line 25). Finally, local rule set  $LR$  is added to the global rule set  $GR$ .

Example of the execution of LEM2 algorithm can be found in [2]. Description of subsequent dataset processing performed by hardware units is shown in Section 2.3.

### 2.3 Hardware Dataset Processing in Proposed Algorithm

As it was mentioned, every line in pseudocode shown in Section 2.2 prefixed by [H1] or [H2] is processed by respective hardware unit. Input dataset is divided by control software into fixed-size parts which are subsequently processed by responsible for given operation hardware unit. General pseudocode for such processing is shown below:

#### HRG2-LEM2 Algorithm dataset processing

**INPUT:** decision table  $DT = (U, A \cup \{d\})$ , additional data dependant on type of operation

**OUTPUT:** operation result structure  $O$

- 1: initialize  $O$  structure
- 2: **for**  $cnt \leftarrow 0$  **to**  $m - 1$  **do**
- 3:    $RAM_{set} \leftarrow \{x \in U : x_{cnt \cdot n} \text{ to } x_{(cnt+1) \cdot n - 1}\}$
- 4:    $O_{part} \leftarrow$  result of calculations performed on  $RAM_{set}$  objects set
- 5:    $O \leftarrow$  combine  $O$  and  $O_{part}$  structures
- 6: **end for**

Input of the hardware control algorithm is decision table  $DT$  and, required by type of performed operation, additional data. Output is general operation result structure  $O$ . Loop in lines 2 to 6 is responsible for choosing parts of input decision table  $DT$ . Decision table is divided into  $m$  parts, where each of them have the fixed size of  $n$  objects. Selected part denoted as  $RAM_{set}$ , is loaded into internal FPGA's RAM memory (more details in Section 3.1) in line 3. Line 4 is responsible for storing partial results of given operation in general result structure  $O_{part}$ . Obtained results are combined together with previous ones using general structure  $O$ .

Brief descriptions of two hardware units as well as operation result structures  $O$  and  $O_{part}$  are given below:

- [H1] - hardware block for calculating objects classes (unit *dtComparator*). Result structure is set represented by binary values, where 1 means that given object is in output set, while 0 means that given object is not in output set. Objects and attributes are processed in parallel by checking given attribute-value pairs. One or more pairs are passed to the module, which then performs comparisons using parts of decomposed input dataset and creates binary sets as a result.
- [H2] - hardware block for calculating number of objects fulfilling conditions  $t = (a, v)$  (unit *avCounter*). Result structure is one-dimensional array containing number of objects equal and not equal for defined conditions  $t$ . Size of array corresponds to number of condition attributes. Similar to previous module, operations on objects and attributes are performed in parallel. Multiple pairs  $t = (a, v)$  are passed to the module and returned results are number of objects that are equal and not equal to given pair.

## 2.4 Data to Conduct Experimental Research

In this paper, we present the results of the conducted experiments using two datasets: Poker Hand Dataset (created by Robert Catral and Franz Oppacher) and data about children with insulin-dependent diabetes mellitus type 1 (created by Jarosław Stepaniuk).

First dataset was obtained from UCI Machine Learning Repository [9]. Each of 1 000 000 records is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. There is one decision attribute that describes the "Poker Hand". Decision attribute describes 10 possible combinations of cards in descending probability in the dataset: nothing in hand, one pair, two pairs, three of a kind, straight, flush, full house, four of a kind, straight flush, royal flush.

Diabetes mellitus is a chronic disease of the body's metabolism characterized by an inability to produce enough insulin to process carbohydrates, fat, and protein efficiently. Twelve conditional (physical examination results) and one decision attribute (microalbuminuria) describe the database. The database consisting of 107 objects is shown at the end of the paper [12]. An analysis can be found in chapter 6 of the book [13].

The Poker Hand database was used for creating smaller datasets consisting of 1 000 to 1 000 000 of objects by selecting given number of first rows of original dataset. Diabetes database was used for generating bigger datasets consisting of 1 000 to 1 000 000 of objects. New datasets were created by multiplying the rows of original dataset. Numerical values were discretized and each attributes value was encoded using four bits for both datasets. Every single object was described on 44 bits for Poker Hand and 52 bits for Diabetes. To fit to memory boundaries in both cases, objects descriptions had to be extended to 64 bits words by filling unused attributes with 0's. Thus prepared hardware units doesn't have to be reconfigured for different datasets until these datasets fit into configured and compiled unit.

### 3 System Architecture and Hardware Realization

Startix III FPGA contains processor control unit implemented as NIOS II embedded core. Softcore processor supports hardware block responsible for rules generation. Hardware calculation blocks are synthesized together with NIOS II inside the FPGA chip. Development board provides other necessary for SoC elements like memories for storing data and programs or communication interfaces to exchange data and transmit calculation results.

#### 3.1 Softcore Control Unit

Hardware modules are controlled by software executed in softcore processor. Main goal of this implementation is:

- read and write data to hardware modules,
- prepare input dataset,
- perform operations on sets,
- control overall operation.

Initially preprocessed on PC dataset is stored on Secure Digital card in binary version. Preprocessing includes discretization and calculation of lower approximations. In the first step of operation, dataset is copied from SD card to external DDR2 RAM module. Results of subsequent operations are stored in FPGA built-in RAM memories (MLAB, M9k and M144k).

MLAB blocks are synchronous, dual-port memories with configurable organization  $32 \times 20$  or  $64 \times 10$ . Dual-port memories can be read and written simultaneously what makes operations faster. M9k and M144k are also synchronous, dual-port memory blocks with many possible configurable organizations. These blocks give a wide possibility of preparing memories capable of storing almost every type of the objects (words) – from small ones to big ones.

#### 3.2 Hardware Implementation

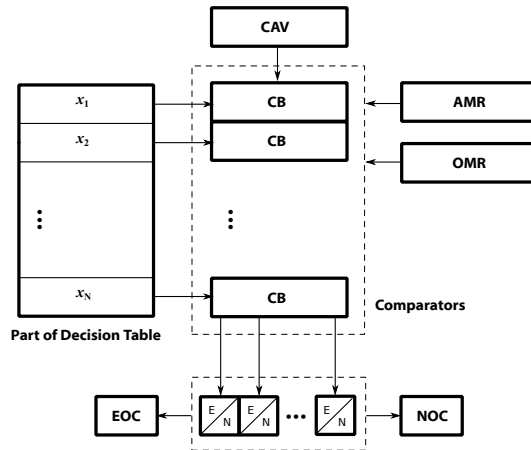
Hardware implementation, created after analysis of the algorithm HRG2-LEM2 described in Section 2.2, was focused on accelerating the operations of calculating number of objects fulfilling attribute-value pairs  $(a, v)$  stored in decision table. Another primary important operation was generating sets of objects fulfilling conditions given as  $(a, v)$  pairs. Hardware blocks were implemented as combinational units, what means that all calculations are performed in one clock cycle. Nature of performed operation gives possibility of using them for parallel computing systems.

Two hardware modules were prepared. First of them is *avCounter* used for parallel counting number of objects occurrences. Second one is *dtComparator*, which in parallel generates sets of objects fulfilling given conditions. Below are the descriptions of prepared modules.

Diagram of *avCounter* module is shown on Fig. 1. Inputs of this module are:

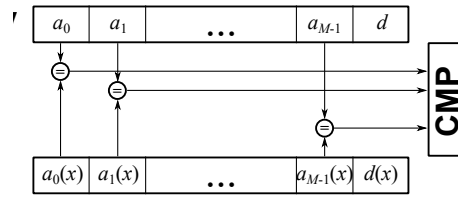
- DATR - decision table containing data for processing,
- CAV - set of conditional attributes' values represented by  $(a, v)$ ,
- AMR - attributes mask register for disabling attributes which are not taken into consideration,
- OMR - object mask register for choosing objects which are processed.

Outputs of the module return the number of objects' occurrences fulfilling each of  $(a, v)$  pair. Returned result represents both values (EOC and NOC) which are calculated in line 10 in algorithm HRG2-LEM2 described in Section 2.2.



**Fig. 1.** *avCounter* module, where  $N$  is a number of objects in part of decision table

Comparator block (CB) shown on Fig. 2 is used to compare the values from CAV register and values of given objects' attributes.



**Fig. 2.** *avCounter* primary building block, where  $M$  is a number of condition attributes

Diagram of *dtComparator* module is shown on Fig. 3. Inputs of this module are:

- DATR - decision table containing data for processing,



- CPDR - defines if conditional part of the rule (or pair  $(a, v)$ ) is defined on input,
- CPVR - contains conditional parts of the rule,
- OMR - object mask register for choosing objects which are processed.

Output of the module (OER) returns the set of objects which fulfills each conditional part of the rule.

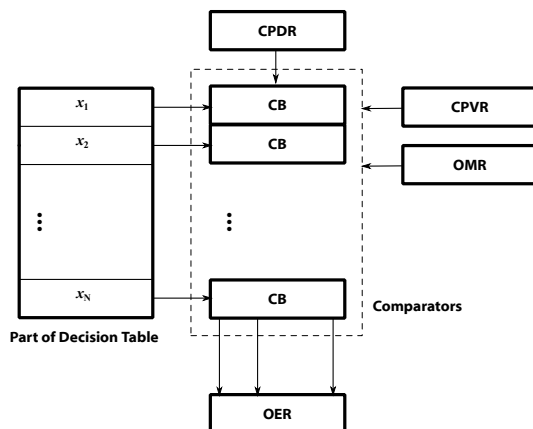


Fig. 3. *dtComparator* module

Single comparator block for *dtComparator* module is similar in principles to the one used by *avCounter*. This block is used to compare the values from CPVR register with objects from decision table.

*avCounter* and *dtComparator* are designed as a combinational circuits and thus do not need a clock signal for proper work. Amount of time needed to obtain correct results depends only on propagation time of logic blocks inside the FPGA. This property allows to significantly increase the speed of calculations because the time of propagation in contemporary FPGAs usually do not exceed 10ns. However, for the proper cooperation with external control blocks, as well as to perform other parts of the HRG2-LEM2 algorithm, both hardware modules must be controlled by the clock.

Main design principle of presented solution assumes, that each of described modules process fixed in size part of dataset. Results of calculations are stored using software implemented inside NIOSII softcore processor. Biggest impact on time of calculation is due to the parallel processing of many objects and all attributes in single clock cycle. Both modules are configured to process 64 objects described by maximum 16 attributes (condition and decision). Extending possibilities of these modules needs simple reconfiguration in VHDL source code and recompilation of hardware units. The same applies to control software.

## 4 Experimental Results

Software implementation on PC was prepared in C language and the source code was compiled using the GNU GCC 4.8.1 compiler. Results were obtained using a PC equipped with an 8 GB RAM and 4-core Intel Core i7 3632QM with maximum 3.2 GHz in Turbo mode clock speed running Windows 7 Professional operational system. Software for NIOS II softcore processor was implemented in C language using NIOS II Software Build Tools for Eclipse IDE.

Quartus II 13.1 was used for design and implementation of the hardware using VHDL language. Synthesized hardware blocks were tested on TeraSIC DE-3 development board equipped with Stratix III EP3SL150F1152C2N FPGA chip. FPGA clock running at 50 MHz for the sequential parts of the project was derived from development board oscillator.

Timing results were obtained using LeCroy waveSurfer 104MXs-B (1 GHz bandwidth, 10 GS/s) oscilloscope for small datasets. Hardware time counter was introduced for bigger datasets.

It should be noticed, that PCs clock is  $\frac{clk_{PC}}{clk_{FPGA}} = 64$  times faster than development boards clock source.

Algorithm HRG2-LEM2 described in Section 2.2 was used for hardware implementation. Software implementation used HRG-LEM2 algorithm (described in paper that is reviewed), which differs from above in lack of dividing data into parts - all data is stored in PC memory. In current version of hardware implementation, authors used data which was preprocessed on PC in terms of discretization and calculation of lower approximations. Time needed for these operations was not taken into consideration in tests related to both types of implementation. Presented results show the times for generating global rule set using pure software implementation ( $t_S$ ) and hardware supported rule generation ( $t_H$ ). Results are shown in Table 1 for Diabetes and Poker Hand datasets. Last two columns in table describe the speed-up factor without ( $C$ ) and with ( $C_{clk}$ ) taking clock speed difference between PC and FPGA into consideration.  $k$  denotes thousands and  $M$  stands for millions.

In this case, one size of hardware execution unit was used, which consumed 15 679 of 113 600 Logical Elements (LEs) total available. This number includes also resources used by NIOS II softcore processor.

Fig. 4 presents a graph showing the relationship between the number of objects and execution time for hardware and software solution for both datasets. Both axes use logarithmic scale.

Presented results show big increase in the speed of data processing. Hardware module execution time compared to the software implementation is 5 to more than 7 500 times faster. If we take clock speed difference between PC and FPGA under consideration, these results are much better - speed-up factor is up to 485 000 for Poker Hand and Diabetes datasets.

Hardware modules speed-up factors for both datasets are similar. It is worth to notice, that it doesn't matter what is the width in bits of single object from dataset, unless it fits in assumed memory boundary. Hardware processing unit takes the same time to finish the calculation for every object size, because it

**Table 1.** Comparison of execution time between hardware and software implementation for Diabetes and Poker Hand datasets using HRG2-LEM2 algorithm

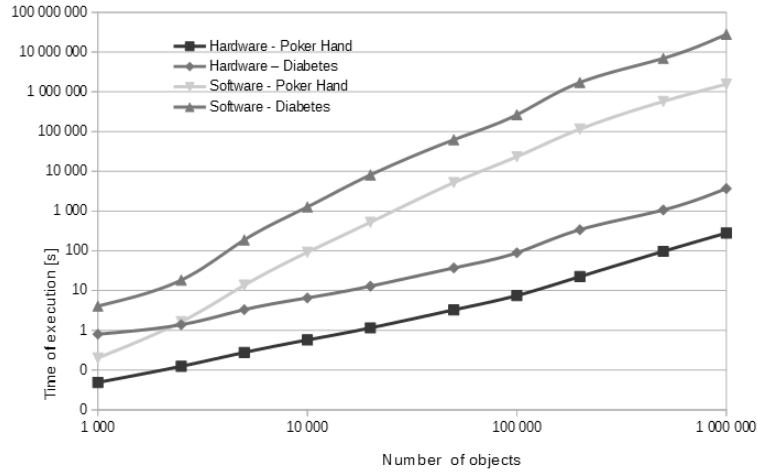
Objects	Software - $t_S$	Hardware - $t_H$	$C = \frac{t_S}{t_H}$	$C_{clk} = 64 \frac{t_S}{t_H}$
—	[s]	[s]	—	—
Diabetes dataset				
1k	4.057	0.798	5.084	325.373
2k	18.304	1.385	13.216	845.817
5k	187.784	3.308	56.767	3 633.064
10k	1 265.410	6.537	193.577	12 388.900
20k	8 101.699	12.886	628.721	40 238.145
50k	62 002.117	37.008	1 675.371	107 223.721
100k	264 164.271	89.297	2 958.261	189 328.678
200k	1 712 942.929	339.162	5 050.520	323 233.298
500k	6 935 214.679	1 065.438	6 509.260	416 592.618
1M	27 908 410.629	3 679.876	7 584.063	485 380.044
Poker Hand dataset				
1k	0.361	0.069	5.244	335.646
2k	1.635	0.124	13.148	841.452
5k	16.620	0.277	60.051	3 843.241
10k	111.835	0.571	195.866	12 535.449
20k	722.089	1.148	629.042	40 258.669
50k	5 186.988	3.261	1 590.498	101 791.898
100k	23 220.455	7.469	3 109.115	198 983.372
200k	145 131.541	29.408	4 935.072	315 844.590
500k	572 771.477	96.699	5 923.252	379 088.126
1M	2 352 553.246	325.329	7 231.314	462 804.119

always performs the same type of operation. Differences between hardware solutions comes from the nature of data and number of loops iterations.

For software execution time comparison of Poker Hand and Diabetes datasets shows, that number of attributes and characteristic of the dataset have big impact on computation time. For hardware implementation characteristic of the dataset has biggest impact.

Let comparison of attributes' values between two objects or iterating over dynamic list of elements be an elementary operation.  $k$  denotes number of conditional attributes and  $n$  is the number of objects in decision table. Computational complexity of software implementation of the rules generation is  $\Theta(kn^4)$  according to algorithm HRG2-LEM2 shown in Section 2.2. Using hardware implementation, complexity of rules generation is  $\Theta(n^4)$ . The  $k$  is missing, because our solution performs comparison between all attributes in  $\Theta(1)$  - all attributes' values between two objects are compared in single clock cycle. Additionally, rule generation module performs comparisons between many objects at time.

Many real datasets are built of tens or hundreds of attributes, so it is impossible to create a single hardware structure capacious enough to process all at-



**Fig. 4.** Relation between number of objects and calculation time for hardware and software implementation using HRG2-LEM2 algorithm for both datasets

tributes at once. In such case decomposition must be done in terms of attributes, thus the computational complexity of software and hardware implementation will be almost the same, but in terms of time needed for data processing, hardware implementation will be still much faster than software implementation. The reason for this is that most comparison and counting operations are performed by the hardware block in parallel in terms of objects and attributes.

## Conclusions

The hardware implementation is the main direction of using scalable rough sets methods in real time solutions. As it was presented, performing rule generation using hardware implementations gives us a big acceleration in comparison to software solution, especially in case of bigger datasets. It can be noticed, that speed-up factor is increasing with growing datasets.

Hardware supported rule generation calculation unit was not optimized for performance in this paper. Processing time can be substantially reduced by increasing FPGA clock frequency, modifying control unit and introducing triggering on both edges of clock signal.

Future research will be focused on optimization of presented solution: different sizes of hardware rule generation unit will be checked, as well as results related to performing the calculations in parallel by multiplying hardware modules. Using FPGA-based solutions it is relatively easy, because multiplication of execution modules needs only few changes in VHDL source code. Most time-consuming part will be design and implementation of parallel execution control unit.

## Acknowledgements

The research is supported by the Polish National Science Centre under the grant 2012/07/B/ST6/01504.

## References

1. T. Grześ, M. Kopczyński, J. Stepaniuk: FPGA in rough set based core and reduct computation, Lecture Notes in Computer Science Vol.8171: Lecture Notes in Artificial Intelligence, Rough sets and knowledge technology: 8th International Conference: RSKT2013, Berlin, Springer-Verlag, 2013, pp. 263-270.
2. J. W. Grzymala-Busse: Rule induction, Data Mining and Knowledge Discovery Handbook, Springer US, 2010, pp. 249-265.
3. A. Kanasugi, A. Yokoyama, A basic design for rough set processor, In The 15th Annual Conference of Japanese Society for Artificial Intelligence, 2001.
4. M. Kopczyński, J. Stepaniuk: Hardware Implementations of Rough Set Methods in Programmable Logic Devices, Rough Sets and Intelligent Systems - Professor Zdzisław Pawlak in Memoriam, Intelligent Systems Reference Library 43, Heidelberg, Springer, 2013, pp. 309-321
5. M. Kopczyński, T. Grześ, J. Stepaniuk: FPGA in Rough-Granular Computing : Reduct Generation, WI 2014 : The 2014 IEEE/WCI/ACM International Joint Conferences on Web Intelligence Vol.2, Warsaw, IEEE Computer Society, 2014, pp. 364-370.
6. M. Kopczyński, T. Grześ, J. Stepaniuk: Generating core in rough set theory : Design and implementation on FPGA, Lecture Notes in Computer Science Vol.8537: Lecture Notes in Artificial Intelligence, Rough sets and intelligent systems paradigms : Second International Conference : RSEISP2014 : Held as Part of JRS 2014, Berlin, Springer-Verlag, 2014, pp. 209-216.
7. M. Kopczyński, T. Grześ, J. Stepaniuk: Core for Large Datasets : Rough Sets on FPGA, Concurrency, Specification & Programming : 24th International Workshop : CS&P 2015 Vol.1, Rzeszow, University of Rzeszow, 2015, pp. 235-246.
8. T. Lewis, M. Perkowski, L. Jozwiak, Learning in Hardware: Architecture and Implementation of an FPGA-Based Rough Set Machine, euromicro, vol. 1, 25th Euromicro Conference (EUROMICRO '99)-Volume 1, 1999, pp. 1326.
9. M. Lichman, UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]. Irvine, CA: University of California, School of Information and Computer Science (2013)
10. M. Muraszewicz, H. Rybiński, Towards a Parallel Rough Sets Computer In: Rough Sets, Fuzzy Sets and Knowledge Discovery, Springer-Verlag, 1994, pp. 434-443.
11. Z. Pawlak, Elementary rough set granules: Toward a rough set processor. Rough-Neurocomputing: Techniques for Computing with Words, Cognitive Technologies. Springer-Verlag, Berlin, Germany, 2004, pp. 5-14.
12. J. Stepaniuk, Knowledge discovery by application of rough set models, Rough Set Methods and Applications. New Developments in Knowledge Discovery in Information Systems, Physica-Verlag, Heidelberg, 2000, 137-233.
13. J. Stepaniuk, Rough-Granular Computing in Knowledge Discovery and Data Mining, Springer, 2008.
14. J. Stepaniuk, M. Kopczyński, T. Grześ: The First Step Toward Processor for Rough Set Methods, Fundamenta Informaticae Vol. 127, 2013, pp. 429-443.