A Hardware Accelerator for Data Classification within the Sensing Infrastructure

Mario Barbareschi, Ermanno Battista, Nicola Mazzocca
Department of Electrical Engineering and Information Technology
University of Naples Federico II, Naples, NA 80125, Italy
name.surname@unina.it

Sridhar Venkatesan
Center for Secure Information Systems
George Mason University, Fairfax, VA 22030, USA
svenkate@masonlive.gmu.edu

Abstract

Cyber Physical Systems are typically deployed using simple sensing nodes and communicate with a complex elaboration and management infrastructure through the internet. The new trend in the design of such systems is to implement significant part of the data elaboration within the sensing infrastructure. Due to the scarce computing capabilities of the nodes and tight performance constrains, it is necessary to equip the nodes with special purpose hardware accelerators. In particular, we discuss a Decision Support System implementation in which special nodes are able to autonomously perform the data classification task.

In this paper, we present a node architecture equipped with a special purpose co-processors to perform data classification trough decision tree visiting algorithm, and we discuss its suitability for the WSN domain.

1. Introduction

In monitoring applications, the status of the environment under surveillance is determined by analyzing the data collected by the sensors. Originally, the raw data from the sensors were communicated to a higher capacity node (cluster head or base station) to process and analyze the data. Communicating large volumes of data depletes sensor's battery quickly. Moreover, with the manifestation of cyber physical security to enable cross-domain interaction [15], the overhead due to communication will further strain a node's longevity. Hence, local data processing emerged as an attractive solution to reduce the communication overhead. However, performance of software implementations of the algorithms degraded with the increase in data to be processed. To efficiently support local data processing, hardware implementation of the algorithms were explored, [13].

In this paper, we propose a special Von Neumann architecture called *Tree Visiting Processing Unit* (TVPU) which follows a tree visiting algorithm to implement the decision tree classifier. To improve performance, TVPU facilitates pipelining and possesses a separate branch predictor unit which leverages the pipelining to predict the flow of execution.

In the following section, we briefly introduce Decision Support System and, in particular, the Decision Tree classifier which is commonly adopted for classifying sensed data.

2. Decision Support Systems

In the infrastructure monitoring domain, the Decision Support System (DSS) plays a crucial role in aiding automatic decision making process. It applies a set of rules to suggest or to enforce pre-defined actions. Typically, the DSS core is implemented by means of data mining algorithms. These data mining algorithms encompass pattern recognition, machine learning and statistical tools to extract useful information/gain insights from the data. One such data mining approach is the Decision Tree (DT). It captures all the conditions in the rules of a DSS and represent them as a tree.

In a DT, nodes are conditions. When a node is evaluated, a choice is made and the tree is traversed in accordance with the choice. Each evaluation is performed on the basis of a model variable, called feature. The leaves of the tree are decision classes. A path begins at the root of the tree and terminates at a leaf, where the class of input data is determined. The DSS will match the class with a corresponding action, according to the business logic. In this article, we refer to *binary* DTs and *binary* visiting algorithm. This is not a restrictive assumption as any n-ary tree can be converted to an equivalent binary tree.

Formally, we indicate a condition as a boolean value $D^k_\rho(f) = \rho(f, k)$: ρ is a binary comparison ($\rho \in \{<, \leq, ==, \neq, >, \geq\}$), k is a constant, f is a formal input parameter, the so called feature. For instance, the representation of the condition $t_{node38} \leq 27.5\,^\circ\mathrm{C}$ is given

IEEE IRI 2014, August 13-15, 2014, San Francisco, California, USA
978-1-4799-5880-1/14/$31.00 ©2014 IEEE

by: $D_{\leq}^{27.5\,^{\circ}\mathrm{C}}(t_{node38}) =\leq (t_{node38},\ 27.5\,^{\circ}\mathrm{C})$.

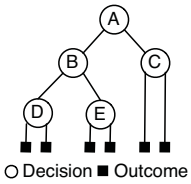

Figure 1: Decision Tree and Branch Prediction

Figure 1 depicts a simple binary DT example with five internal nodes. The intrinsic binariness of the tree is due to the fact that each node has 2 descendants and they will be visited by means of a binary decision. For instance, if the A condition is true, the algorithm continues in evaluating the B condition, otherwise the algorithm will evaluate the C condition and so on.

In the WSN domain, DSSs based on DTs are well suited [1]. As for the others domain where DSSs are adopted, the heterogeneity of the data sources is dealt with *semantic process*. This technique involves the adoption of a RDF schema, which can be used to gather and encode the measured data. A data model can be automatically generated using a learning algorithm (such as ID3 or C4.5 [8]), which takes as input a pre-labeled set (the learning set) and outputs a data model. The resulting data model is used by a prediction algorithm (e.g. tree visiting) to classify an input data. Decision trees also provide the flexibility to tune the DSS; as new conditions surface, the data model can be refined and updated by executing the learning phase.

As mentioned earlier, as data volume increases or when the data model becomes complex, a software implementation may not be able to meet the application requirements. In such cases, moving to a hardware implementation maybe necessary to handle the data volume. Although some approaches are discussed in the literature [13], to the best of our knowledge, this is the first work that proposes a hardware accelerator specifically designed to facilitate data classification in the WSN domain.

In the next section, we discuss a hardware implementation of a tree visiting algorithm that is suited for WSN sensing applications.

3 Implementing a Tree Visiting Processor

In a previous work [2], we proposed a speculative way to design a massively parallel hardware accelerator for visiting the decision tree. In this section, we introduce the implementation of a special purpose processor for tree visiting. The architecture we propose is called *Tree Visiting Processing Unit* (TVPU), and is designed to have a small hardware footprint. It is a special Von Neumann architecture capable of performing only *compare and branch* operation and *store* operation. This instruction set is the minimum amount of operation required to perform a tree visiting algorithm and returning a classification.

The tree visiting algorithm is computed starting from the root of a decision tree and returns a classification of the input data. At each step a condition, described in the node, has to be evaluated through a compare and branch operation. If the condition is true, the algorithm proceeds on the left sub-tree, else on the right one.

Algorithm 1 treeVisiting(features[], DT)

Require: A feature vector **features[]**; a classification **model** DT
Ensure: The predicted class

```
1: node ← model.root
2: while !node.isLeaf() do
3:     if node.ρ(features[node.f], node.k) then
4:         node ← node.leftChild
5:     else
6:         node ← node.rightChild
7:     end if
8: end while
9: return node.label
```

As illustrated in Algorithm 1, the tree visiting requires only *comparisons and branches*. Once the algorithm is evaluated, the returned value is needed to be saved in memory through a *store* operation. To build a hardware special machine that executes this algorithm, we defined two instruction types: (i) the node instruction and (ii) the leaf instruction. The first (Algo 1, lines 3:6) evaluates the branch condition, the latter (Algo 1, line 9) stores the result of the algorithm (the class represented by the node label).

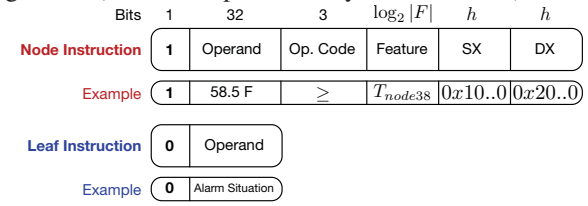

Figure 2: Tree Visiting Processing Unit Instruction Set

The format for the two instruction types is given in Figure 2. The first bit in the instructions defines the *instruction type*. The following 32 bits are the value of an *Operand*: in case of a Node instruction, this field contains a constant value (a single precision floating point) which is required to compute an operation in combination with the feature field; in case of a Leaf instruction, the Operand contains the classification result. The *Operation Code* field is expressed on 3 bits and is used to specify one of the seven operation supported by the architecture: (i) No-Operation, (ii) $>$, (iii) $<$, (iv)$\geq$, (v) $\leq$, (vi) $==$ and (vii) $\neq$. The *Feature* field represents the actual feature that is being evaluated in the current operation. The length of this field depends on the number of feature taken into account in the data model. If F is the set of all features, then the *Feature* field will be $\log_2|F|$ bits long. The last two fields in the Node instruction set are the

address to the left (*SX*) and right child (*DX*) of node. To point to any node in the tree, we need $h = log_2(|N| + |L|)$ bits, where N is the set of internal nodes and L is the set of classes.

The TVPU special purpose processor loops through three phases similar to the general purpose Von-Neuman processor:

Instruction Fetch: an instruction is retrieved from the instruction memory; the address is kept in the Program Counter (PC) register.

Instruction Decode: the fetched instruction is decoded; if the instruction is of node type, the actual operands have to be loaded from the feature memory.

Instruction Execute: in the case of node instruction, the specified operation is evaluated over the operands. The result indicates which child address (the left or the right) needs to be loaded in the PC. In the case of a leaf instruction, the operand (or class) is stored in a predefined memory location.

To increase the throughput, the sequential execution of the Von Neumann instruction cycle can be parallelized through pipelining which is discussed the following section.

3.1 Pipelined Execution and Branch Prediction

Pipelining is a technique used to increase the number of operations executed in a time unit. In pipelining, the processing stages can work independent of each other on different instructions. However, the shortcoming of adopting a pipelined implementation is the introduction of hazardous conditions on the branches. For instance, to feed the pipeline, it is required to load subsequent instructions before evaluating the branch. To overcome this hazard, a famous technique, known as Branch Prediction (BP), has been incorporated in our architecture.

A BP is a special unit whose aim is to predict the outcome of a branch condition before evaluating it [11]. Its purpose is to feed the instruction pipeline with instructions that are likely to be executed. Consider the case of an if-then-else block. The processor pipeline needs to be fed continuously; in this case, before evaluating the if-then-else block, either an instruction inside the *if* block or in the *else* block is legitimate to be executed. The branch prediction unit decides which of the block (*if* or *else*) will be pushed into the pipeline. The execution derived by the prediction is in a speculative state: if the BP prediction will turn out correct, the speculation will be resolved with a commit; otherwise the speculation fails with a roll-back operation that puts the processor in a non-speculative state, executing the right instruction. A good Branch Predictor reduces the number of wastefully executed operations, i.e., the case of executing the *if* block but the branch evaluation results in the *else* case. To this end, some predictor history table techniques can be implemented in order to bias the predicted branch on the basis of previous branch evaluations [5].

The BP effectiveness is guaranteed by stationary condition on the branches: a condition is stationary if it changes with a low frequency. If the branch refers to a stationary condition, the BP will predict the next instruction to be executed with high accuracy. In the case of high variable condition, the BP will be less accurate leading to great number of mispredictions and negatively affecting the performance.

3.2 Effectiveness of the BP Unit in WSN

A sensor node typically monitors physical phenomena, such as temperature, humidity, pressure, etc. Sampling a physical process results in values which are slower in changing than the system working frequency. Since the BP efficacy is directly related to the lack of change, the branch predictions on these values will be accurate. A branch misprediction will occur only when the new value assumed by the sample has to be classified differently from a previous sampling. This is the case when the monitored phenomenon crosses a threshold which separates two classification classes. These thresholds are defined by the learning algorithm and are stored in the data model.

To better understand, consider figure 3 in which we represented a simple data model. It has two features Λ_0 and Λ_1 and two classes, the dark and light gray classes. The graph indicates that a generic point (a, b) belongs to the dark gray class if $(a \leq \lambda_0 \wedge b \leq \lambda_1) \vee (a > \lambda_0 \wedge b > \lambda_1)$, in the other cases it belongs to the light gray class. The curve reported on the graph is a representation of the value assumed by the samples. As one can notice, the data model associates to the samples different classes over time. In particular, when a threshold is crossed, the data model classifies the sample with a different class. During transitions, the branch prediction is expected to fail (Algorithm 1, line 3). The misprediction is due to the fact that the previous value assumed by the sampled data leads to a different decision path in the tree, so the algorithm has to change its execution flow. The red circles in Figure 3 highlight the misprediction events around the thresholds.

Since the misprediction occurs when at least one sampled feature crosses a threshold, the frequency of the misprediction event is directly proportional to the phenomenon variation frequencies around the thresholds. Data models are built in order to define a n-dimensional region (among the features space) to isolate data which are highly correlated. So that, it is really unlikely that a well defined data model will identify regions where data are positioned near the borders of a classification region instead of its center.

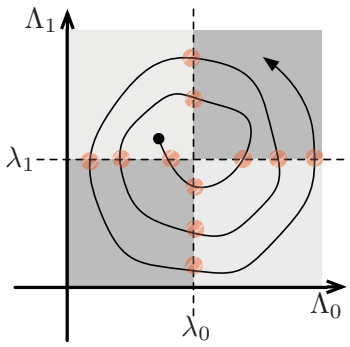

Figure 3: Branch prediction effects on the tree visiting process

3.3 Tree Visiting Processor Architecture

The Tree Visiting Processing Unit has been implemented in VHDL language. Figure 4 depicts a detailed Register Transfer-Level (RTL) schema of the architecture. The design of the TVPU is a five-stage pipeline with a Control Unit which manages the execution. The registers BP/IF, IF/ID, ID/EX1, EX1/EX2 isolate each pipeline stage from the others, keeping the information strictly required by the next stages. The first stage comprises of the Program Counter (PC) and the Branch Prediction Unit (BPU). The PC points to the instruction address that is to be executed and the BPU feeds the PC with the instruction address that is more likely to be executed in the next cycle. In the second stage, the instruction is fetched from the memory address pointed by the PC. This stage is followed by instruction decode. Here, the *isLeaf* flag represents the instruction type and controls the execution stage and the selection of the output. If *isLeaf* flag is asserted, then it is a Leaf instruction and only the operand has to be extracted. If *isLeaf* flag is not asserted, then it is a Node instruction. For a Node instruction, the instruction decode will extract the address to the left and right child, the feature index, the operation code and the operand. The feature index points to the address of the feature (used to make the current decision) in the feature memory.

In the next stage, the execution depends on the type of instruction. For a Node instruction, an operation (selected through the operation code) is applied to the the actual feature and the operand (e.g. $T_{Node_8} \leq 58.5F$). In case of a Leaf instruction, no operation is performed and the operand is directly forwarded to the next stage. The multiplexer, on the right part of this stage, handles the output selection depending on the instruction type.

The last stage completes the execution phase: in case of Leaf instruction, the output is just forwarded to the Control Unit (CU), otherwise depending on the ρ's result, the left or the right child is selected through a multiplexer.

The CU manages all the execution, taking into account the speculative state of the processor. The main three functions are: starting and stopping the processing flow; managing speculation, checking if the execution is in compliance with the BPU prediction; managing access to the instruction and feature memory. In particular, to evaluate the speculation consequence, the CU keeps all the speculative addresses that the BPU loads in the PC. After the initial latency (in this case five clock cycles), at every EX2 node instruction completion, the CU compares the outcome address with the address that the BPU loaded into the PC five clock cycles ago. If they are equal, then CU continues to evaluate the subsequent instructions. If the addresses are different, it implies that BPU have chosen the wrong address to be loaded into the PC. To solve this issue, the CU makes the following operations: it flushes the pipe registers, loads in the PC the correct address and updates the BPU prediction table with the correct address. At the end, the CU waits the pipe latency before checking the outcome address again. In the case of leaf instruction completion, the CU stops to feed the pipeline, signaling the end of the algorithm and saving the outcome predicted class.

With respect to the memory requirements (BPU table, instruction and feature), total size can be computed as: $F_{Mem} = 32 \cdot |F|$ where 32 is the size of an operand in single precision floating point format; $I_{Mem} = h \cdot [1 + 32 + 3 + log_2 |F| + 2 \cdot h]$[1]; $BPU_{Mem} = 2 \cdot h \cdot |N|$.

Table 1 reports the TVPU performance in terms of clock frequency, throughput and occupied slices. The values are retrieved using Xilinx ISE 14.4, after the PAR phase, configured to synthesize on the Virtex XC5VLX110T FPGA. Our VHDL project is defined with generics port, so it is easily configurable in maximum number of nodes and in maximum number of features. Since the feature cardinality seems to not significantly affect the performance, we hold the number of features to 32. The cardinality only affects, in logarithmic fashion, the memory instruction width and with minimal extent the pipe registers. On the other hand, the number of nodes linearly increases the instruction memory. Moreover, we compute the resource efficiency, that relates the FPGA slice usage (hardware footprint) to the architecture effective throughput. Since we use BRAMs to implement the TVPU memories, the occupied slices refers only to the control logic and pipes registers footprint. To take into account the BRAMs in the resource efficiency metric, as generally done we consider each BRAM equivalent to 72 slices.

Table 1 shows a logarithmic trend in slice occupancy, because for each doubling in nodes number it is only required to add only one bit in pipe registers. As for the throughput, there is no direct way to obtain the equivalent through-

[1]This is the worst case scenario formula because we optimize the memory usage by considering not all the leaves, but the equivalent class they belongs to. This allows us to reduce the number of leaf instructions we need to save.

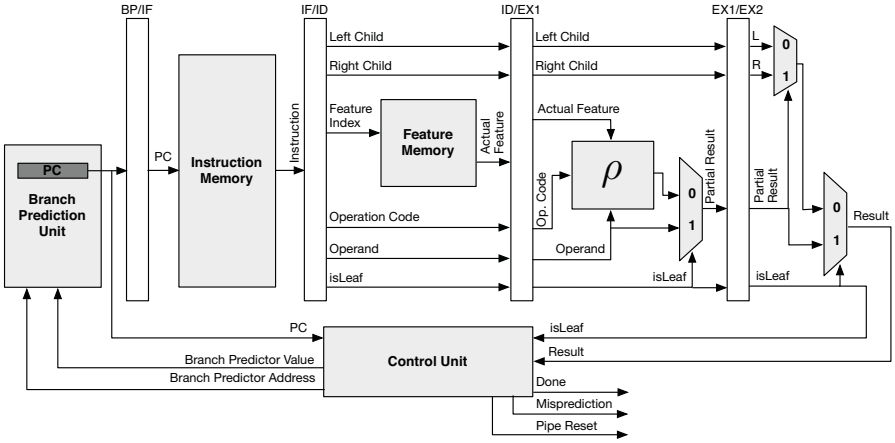

Figure 4: Architecture overview of the Tree Visiting Processing Unit

Max. #Nodes	Max. Clock Frequency (MHz)	Throughput (Gbps)	Occupied Slices	Resource Eff. (Slices/Gbps)
32	360.36	36.90	246	6.73
64	348.77	32.46	269	8.43
128	336.01	28.67	301	10.82
256	349.37	27.51	327	12.58
512	361.15	26.41	349	14.73
1024	344.78	23.53	384	19.84
2048	329.99	21.12	432	28.58
4096	310.14	18.68	468	44.01

Table 1: Performance Evaluation

put due to the speculative execution. Hence, we report the throughput evaluated in the case of a complete tree without any misprediction.

4 Experimental Results

In order to demonstrate the feasibility of the approach, we implemented the TVPU as a co-porocess on Xilinx Virtex XC5VLX110T. The core processor was implemented with a Microblaze soft-core: it controls the co-processor and feeds it with the features vector to process.

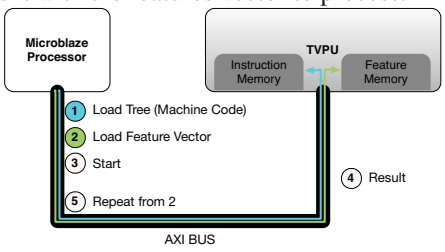

Figure 5: Core Processor and TVPU co-processor

Figure 5 depicts the architecture components and the operational scheme. The Microblaze processor loads a tree into the TVPU's Instruction Memory. The tree is converted from a Predictive Model Markup Language (PMML) model description in the TVPU machine code. Once the tree has been loaded, the Microblaze loads the feature vector into the TVPU's Feature Memory, it subsequently starts the classification process through a start command. The TVPU computes the classification and interrupts the Microblaze processor through an interrupt line and returns the classification result. The Microblaze processor will then loop through the operation starting from the feature vector loading. As for the software loaded into the TVPU, we used a training set to obtain a decision tree. The training was composed by temperature, pressure and humidity sampled by 30 node sensors pre-classified using 20 alarm levels: 0 no alarm, 19 high severity alarm. The obtained tree consists of 2348 leaves and nodes. The maximum height of the tree was 21.

Figure 6 reports the TVPU performance in terms of classification and misprediction delay. The lower the computation delay, the better are the performance: it classifies quickly. On the other hand, the higher the misprediction delay, the better the performance: it takes long time between two misclassifications. Comparing the two graphs in Figure 6, one can notice that the classification delay and misprediction delay are inversely related.

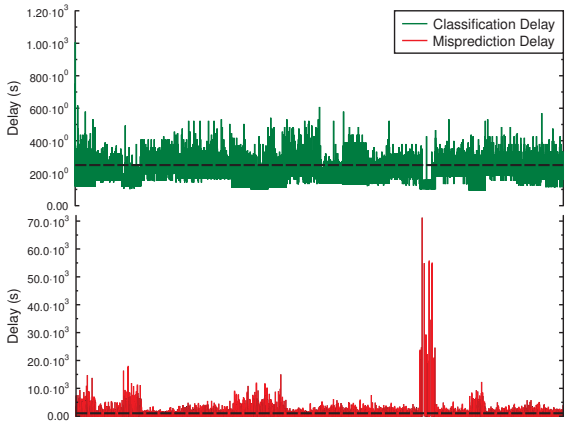

Figure 6: Classification and Misprediction Time Trends

5. Related Work

Decision Support System (DSS) are widely adopted in the WSN domain due to the introduced benefits, such as automatic decision-making, alarm generation, maintenance scheduling, and so on. In the literature, most of the well known approaches to deploy a DSS are specialized for a particular domain (agriculture, medicine, weather). The Energymon, presented in [12], is a DSS that collects data over a WSN to monitor the energy performance of buildings. In this work, the DSS is used to classify the energy consumption category of a building starting from the monitoring of environmental parameters such as temperature, heat flow and gas consumption. The authors in [4] defined a DSS with the aim of support the decision making process for a precision irrigation system. They adopted semantic processing and decision tree machine learning approach in order to define the rules and improve the accuracy of the irrigation system. In the clinical field, the adoption of a DSS improves the health-care monitoring applications. With the growth of data sources (e.g. mobile e-health sensors), new techniques, such as machine learning, and new architectures are needed to guarantee a low response time and high accuracy [6].
To meet high performance in the classification phase, hardware solution are proposed. In particular, FPGA are very suited to implement classification algorithm because DSS requires reprogramming during the tuning phase. Several FPGA-based classification architectures have been proposed in order to accelerate the classification processes [9, 10]. In [14] two FPGA architectures were proposed to classify packets traffic. Both architectures use a programmable classifier exploiting the C4.5 algorithm. Our processing unit seems to have better resource efficiency, with a high number of tree nodes, compared to the low cost architecture presented in [14]. Using NetFPGA, in [7] the authors proposed a traffic classifier by using C4.5. The main architectural characteristic is the software programmability, without loss of service.

6 Conclusion and Future Work

This work proposed a co-processor to enable local data classification by sensor nodes using the decision tree visiting algorithm. To practically realize the proposal, we show the implementation of TVPU on a FPGA which exploits a BPU to enhance the process throughput. We argued that the proposed solution offers very high performance: using a real data set, we show the proposed approach suitability in the WSN domain. The proposed approach, is not limited to the classification of WSN data, but it can be also adopted to analyze communication traffic among the nodes [3].

Furthermore, as a part of our future work, we plan to show the feasibility of the approach and evaluate its energy consumption by comparing with available software/hardware solutions.

References

[1] F. Amato, M. Barbareschi, V. Casola, and A. Mazzeo. An fpga-based smart classifier for decision support systems. In *Intelligent Distributed Computing VII*, pages 289–299. Springer, 2014.

[2] F. Amato, M. Barbareschi, V. Casola, A. Mazzeo, and S. Romano. Towards automatic generation of hardware classifiers. In *Algorithms and Architectures for Parallel Processing*, pages 125–132. Springer, 2013.

[3] M. Barbareschi, A. Mazzeo, and A. Vespoli. Network traffic analysis using android on a hybrid computing architecture. In *Algorithms and Architectures for Parallel Processing*, pages 141–148. Springer, 2013.

[4] C. Goumopoulos, B. OFlynn, and A. Kameas. Automated zone-specific irrigation with wireless sensor/actuator network and adaptable decision support. *Computers and Electronics in Agriculture*, 105:20–33, 2014.

[5] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2012.

[6] M. S. Mohktar, K. Lin, and S. J. e. a. Redmond. Design of a decision support system for a home telehealth application. *International Journal of E-Health and Medical Communications (IJEHMC)*, 4(3):68–79, 2013.

[7] A. Monemi, R. Zarei, and M. N. Marsono. Online NetFPGA Decision Tree Statistical Traffic Classifier. *Computer Communications*, 2013.

[8] J. R. Quinlan. *C4.5: programs for machine learning*. MKP Inc., San Francisco, CA, USA, 1993.

[9] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics*, 11(9):647–657, 2010.

[10] P. Skoda, B. Medved Rogina, and V. Sruk. FPGA implementations of data mining algorithms. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 362–367. IEEE, 2012.

[11] C. O. Stjernfeldt, E. W. Czeck, and D. R. Kaeli. Survey of branch prediction strategies. Technical report, Technical Report CE-TR-93, 1993.

[12] P. T. Szemes, Z. Baranyai, J. Hamar, and M. Zoltai. energymon: Development of wireless sensor network based decision support system to monitor building energy performance. In *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, pages 31–36. IEEE, 2011.

[13] B. Tavli, K. Bicakci, R. Zilan, and J. M. Barcelo-Ordinas. A survey of visual sensor network platforms. *Multimedia Tools and Applications*, 60(3):689–726, 2012.

[14] D. Tong, L. Sun, K. Matam, and V. Prasanna. High throughput and programmable online traffic classifier on FPGA. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 255–264. ACM, 2013.

[15] F.-J. Wu, Y.-F. Kao, and Y.-C. Tseng. From wireless sensor networks towards cyber physical systems. *Pervasive and Mobile Computing*, 7(4):397–413, 2011.