Hardware Decision Trees for Machine Learning on Chip

Walter Gallego Gomez

Outline

- Decision trees basics
- Unsupervised learning for decision trees
- Decision trees HW implementations

Decision Trees basics

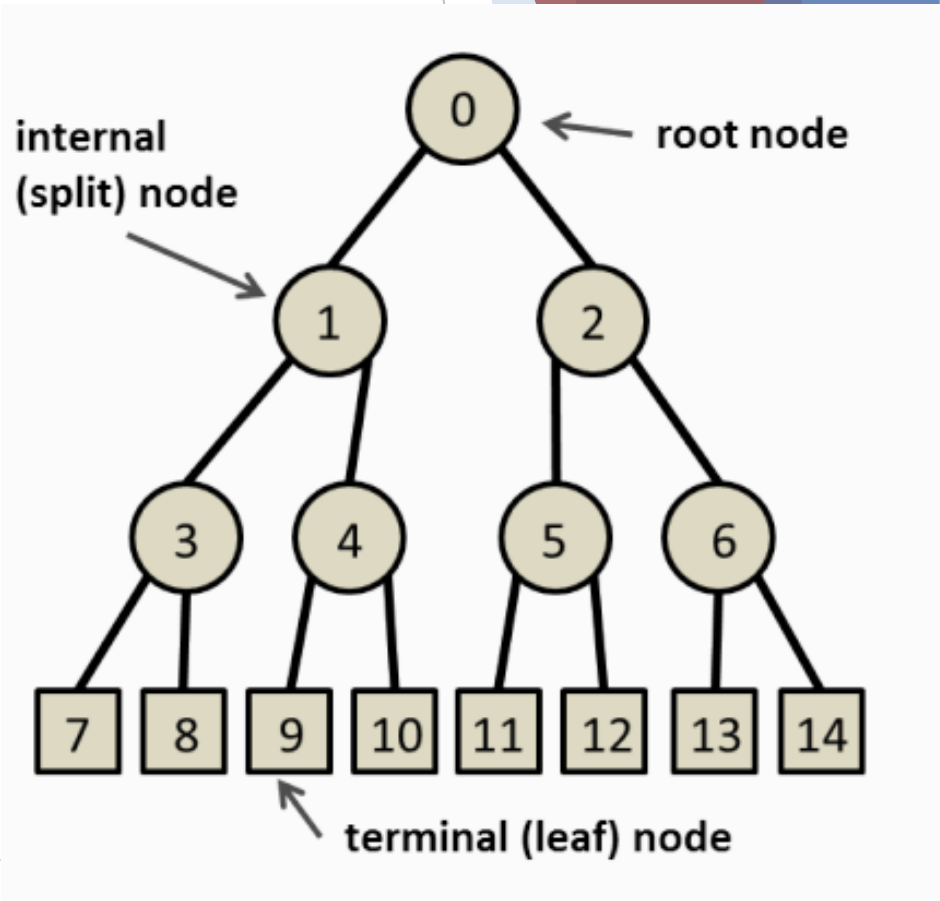
Decision Trees

Decision tool that uses a tree like graph.

Each node implements a function over the sample attributes

The branches of each node represent the different outcomes the function can take.

Final nodes (leaf) represent the outcome of the tree.



Decision Trees

Decision trees can be:

Binary: If the outcome of each node can only have two results.

Ordered: continuous/numerical attributes.

Unordered: discrete attributes.

Classification vs regression

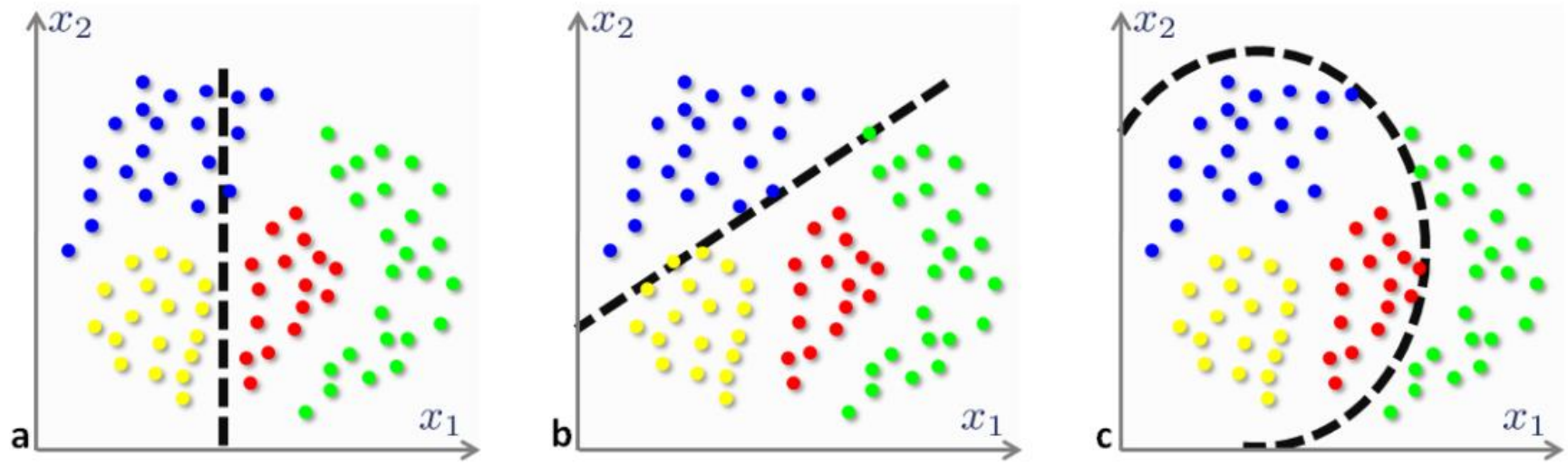
A decision tree is said to perform classification if the class labels are discrete values, and regression if the class labels are continuous.

Decision Trees

Axis-parallel(Univariate): The function implemented on each node involves only one of the attributes, and it is linear.

Oblique(Multivariate): The function at each node can involve more than one attribute, and it is linear.

Non - linear: The function is non-linear



Tree Induction (for labeled training data)

The problem of building optimal decision trees, optimal in the sense of minimizing the expected number of tests required to classify an unknown sample is NP-Complete. even the problem of identifying the root node in an optimal strategy is NP hard.

Therefore, some **sub-optimal** strategy needs to be adopted.

Tree Induction - greedy top-down

Starting with an empty tree and the entire training set, the following algorithm is applied until no more splits are possible:

For each node:

- score each one of the possible splits S, using a goodness measure.
- Choose the best split S* as the test at the current node, and create as many child nodes as there are distinct outcomes of S*.
- If all the training examples at the current node t belong to category c, create a leaf node with the class c and halt.

Goodness measure – Gini score

Gini score is a measure of how good is a split at dividing training samples. If each of the branches contains samples with the same class label, the split is good:

$$Gini_i = 1 - \sum_j \left(\frac{S_{ij}}{S_i}\right)^2$$

$$Gini = \sum_i \frac{Si}{S} \times Gini_i$$

Si: Number of samples on branch *i*

Sij: Number of samples on branch *i* that belongs to class *j*

The lower the gini score, the better

Possible splits

(Considering only axis-parallel trees)

For each attribute:

- If the attribute is discrete, only D splits are possible, where D is the number of values the attribute can take.

- If the attribute is continuous, the number of possible splits is infinite. In practice the attribute has to be discretized.

Unsupervised learning

Unsupervised learning: Clustering

As we have seen, the tree induction process requires a goodness measure for each possible split. The Gini score (and other metrics) require labeled data, so they cannot be used here.

To overcome this problem some different metrics need to be used.

Using logical decision trees for clustering (1997)

The following metrics are suggested:

Intra-branch distance: The average distance between the samples inside each branch needs to be minimum. A small distance indicates samples belong to a cluster.

Inter-branch distance: The distance between branches needs to be maximum. This indicates the split is good at separating samples belonging to different clusters.

Using logical decision trees for clustering (1997)

To calculate these metrics we need to define:

- How to measure distances between samples. For real attributes, we can consider some classical function like the **Euclidian distance** (or an approximation, like Taxicab distance). If the attributes are **enum**, Hamming distance or Levenshtein distance.

- How to find a prototype (or centroid) of a set of samples, to allow the calculation of distance between branches.

Hierarchical clustering

Hierarchical clustering algorithms are of two types:

Agglomerative: This is a "bottom up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

Divisive: This is a "top down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

Hierarchical clustering

In hierarchical clustering, the training data is divided in clusters, but maybe the case no rules are devised for classifying new test data.

Nonetheless, the algorithms are based on finding minimum/maximum **distances** between samples/groups of samples, and those ideas may be useful when constructing the decision tree.

Clustering Via Decision Tree Construction (2000)

In this paper, an alternative algorithm is described:

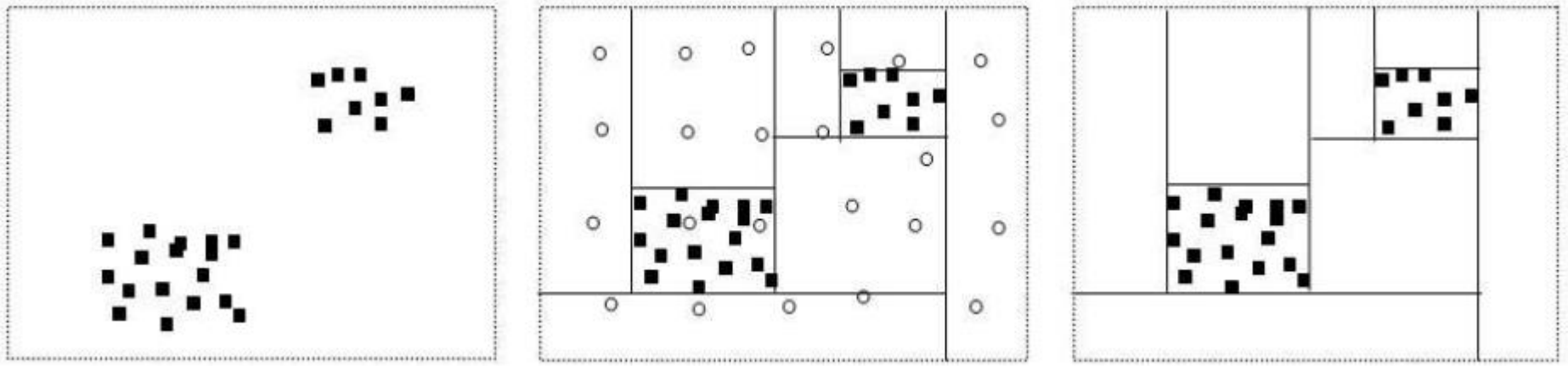
The basic idea is that to regard each data record (or point) in the dataset to have a class Y . We then add another type of points, called non-existing points. We give them the class, N.

With the N points added to the original data space, our problem of clustering regions becomes a classification problem.

Clustering Via Decision Tree Construction (2000)

The reason why this technique works is that if there are clusters in the data, the data points cannot be uniformly distributed in the entire space. We can isolate the clusters because within each cluster region there are more Y points than N points, and in the empty regions, there are more N points than Y points.

The decision tree technique is well known for this task.



Clustering Via Decision Tree Construction (2000)

With the virtual labels we can apply one of the metrics used for supervised learning, like Gini index or **information gain**.

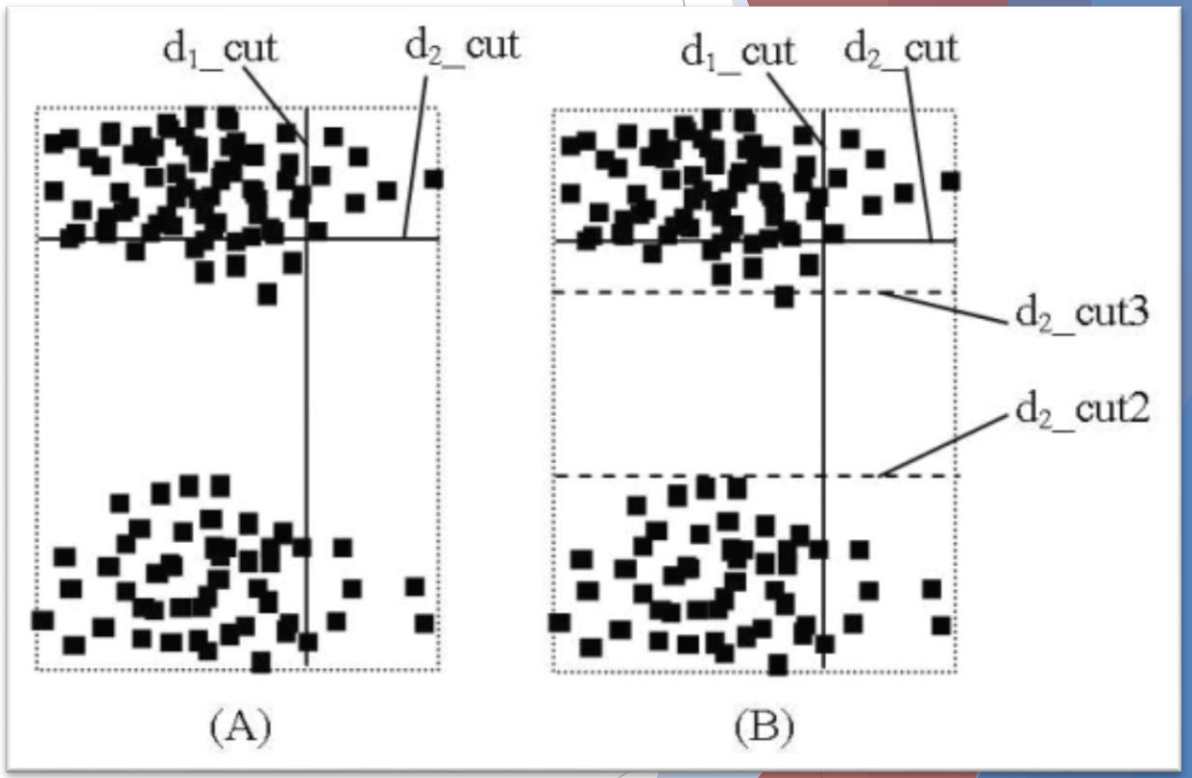
The splits chosen are going to be those that isolate the regions where only N-labeled points exist (the empty regions), which will eventually lead to clustering.

The N-labeled points are not physically added to the data set (not stored in memory). As their distribution in uniform, the number of N points in a given region can be easily calculated.

Clustering Via Decision Tree Construction (2000)

To improve the accuracy, the algorithm is extended, to do a 2-step *lookahead*.

Starting from the cut obtained from the information gain, a more suitable cut for **clustering** is searched. The regions with lower number of Y points (lower **density**) are favored, as they probably contain empty regions.



The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning

Instance based learning algorithm that can be used for:

Supervised learning

- Classification
- Regression

Unsupervised learning

- Nearest neighbor Retrieval

(It can not be used for **clustering**)

The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning

It adds data points one at a time, hence it is naturally **online**. Few instance-based algorithms have this property while being simultaneously **fast** in training and querying.

Each node in the tree is a sample.

The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning

Training (for classification):

The algorithm moves through the tree starting from the root node, and recursively compares the **distance** to the query point from the current node and from its children, moving to and recursing at the child node that is closest to the query, unless the current node is closest. In this case the algorithm stops and the sample label is **predicted** as equal to the current node.

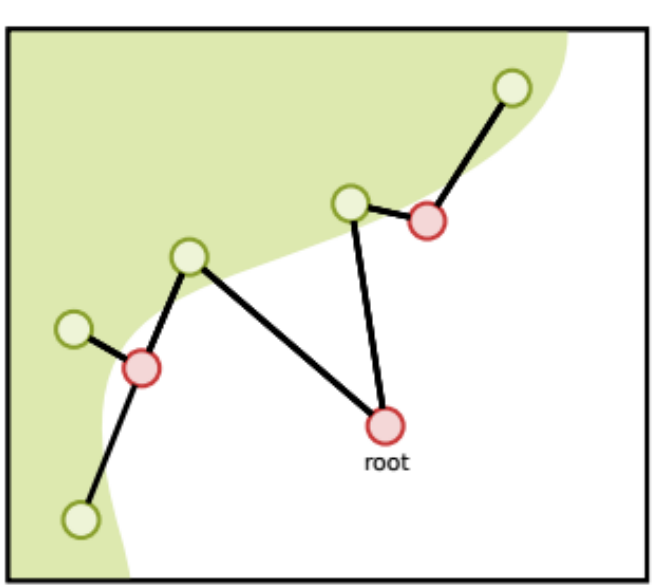
The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning

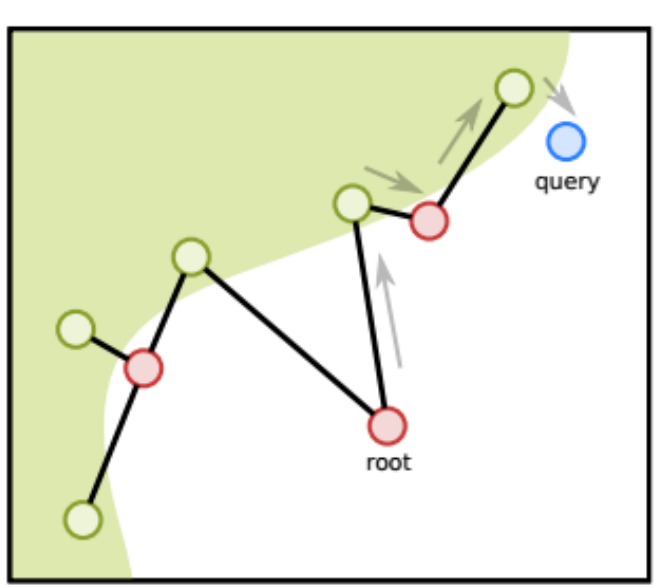
Training (for classification):

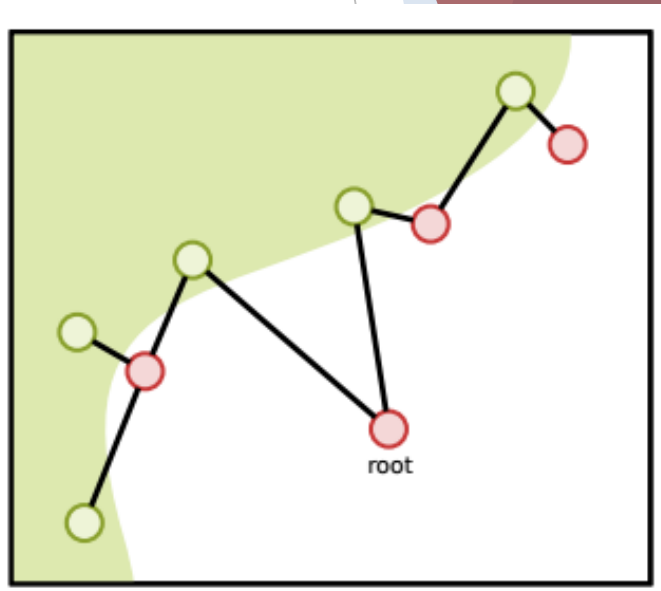
Samples are **only added** to the tree when they can provide significant new information.
If the prediction is right, there is no need for adding the new sample, because the tree can make good predictions without it.
If the prediction is wrong, the sample is added as a child of the current node.

The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning

Example: Two labels, red and green. The blue bubble is the new sample







The query is predicted as green, because the node closest to it is green. But the query is actually red, then we have to add it to the tree.

The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning

For the other two modes, the recursive process is the same, but the decision to add a new sample changes:

For regression: As the label space is continuous, the decision is not based on equality, but on a comparison: $\left|label_{predicted} - label_{real}\right| > \epsilon$

For retrieval: As there is no information about the label, all the samples in the training set are added to the tree.

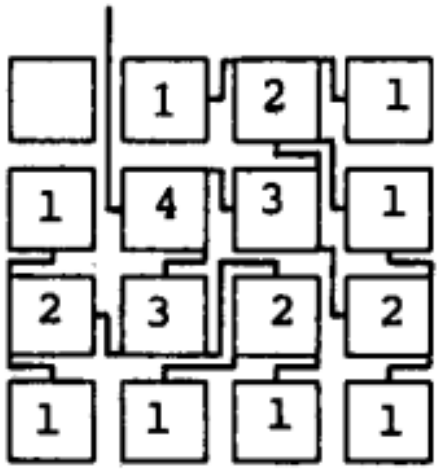
Hardware implementations

On Implementing Large Binary Tree Architectures in VLSI and WSI (1989)

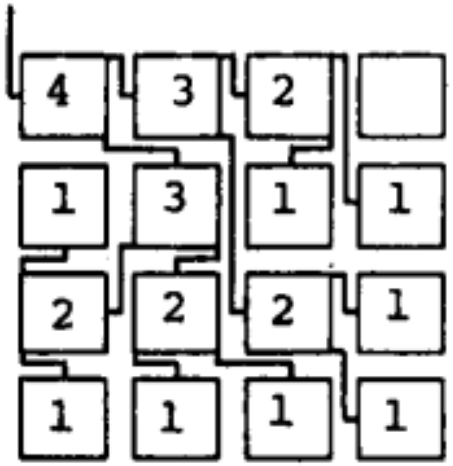
The paper shows how to map any binary tree in a rectangular shaped array.

Area utilization is maximized, as only one of the PE is left unconnected, for any tree size.

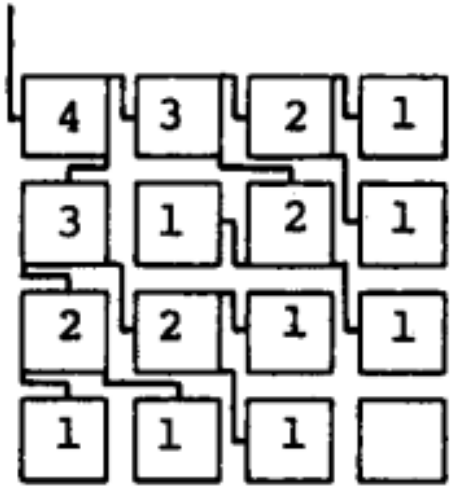
The layout is based on four types of basic modules, each implementing a 4-level tree. By combining this modules, one can describe any tree.


1 2 1
1 4 3 1
2 3 2 2
1 1 1 1

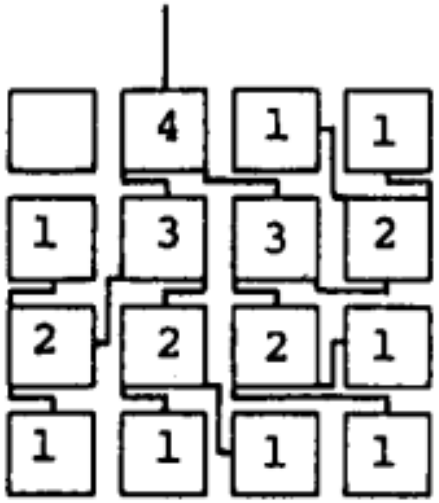
M1


4 3 2
1 3 1 1
2 2 2 1
1 1 1 1

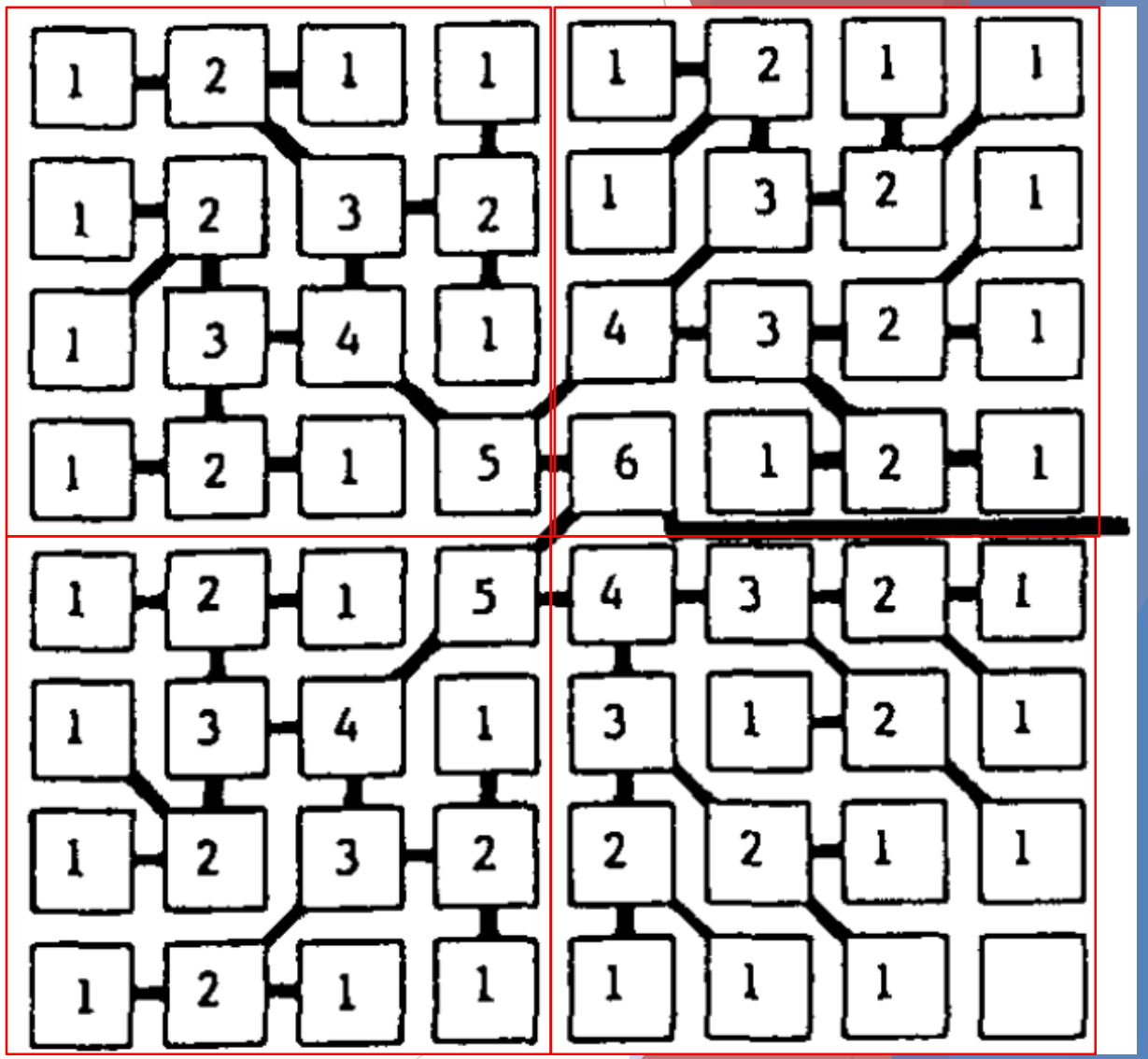
M2


4 3 2 1
3 1 2 1
2 2 1 1
1 1 1

M3


4 1 1
1 3 3 2
2 2 2 1
1 1 1 1

M4


1 2 1 1
1 2 3 2
1 3 4 1
1 2 1 5
1 2 1 1
1 3 2 1
4 3 2 1
6 1 2 1
1 2 1 5
1 3 4 1
1 2 3 2
1 2 1 1
4 3 2 1
3 1 2 1
2 2 1 1
1 1 1

An FPGA Implementation of Decision Tree Classification (2007)

The paper focus on the tree induction process. and uses the Gini score as metric for the possible splits.

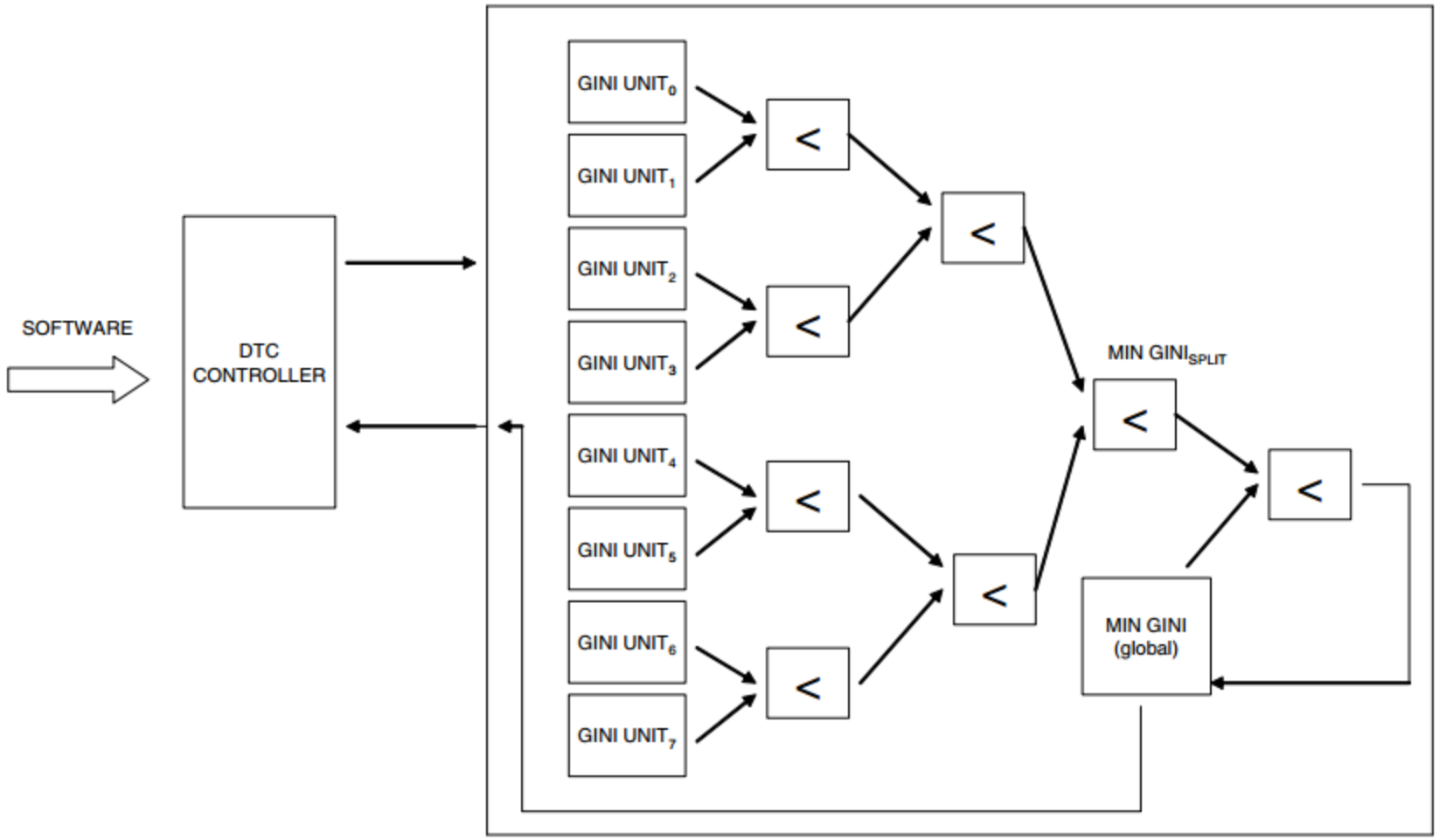
In SW, Gini score calculation can take up to 40% of the total time.

The authors describe a HW implementation of a simplified version of the Gini score calculation, to obtain higher performance and fewer resources utilization.

An FPGA Implementation of Decision Tree Classification (2007)

To find the best split, it is necessary to evaluate the gini score for all the possible splits.

As each split is defined by a comparison between an attribute and a value, and the comparisons are independent, they can be performed in parallel.



Implementing Decision Trees in Hardware (2011)

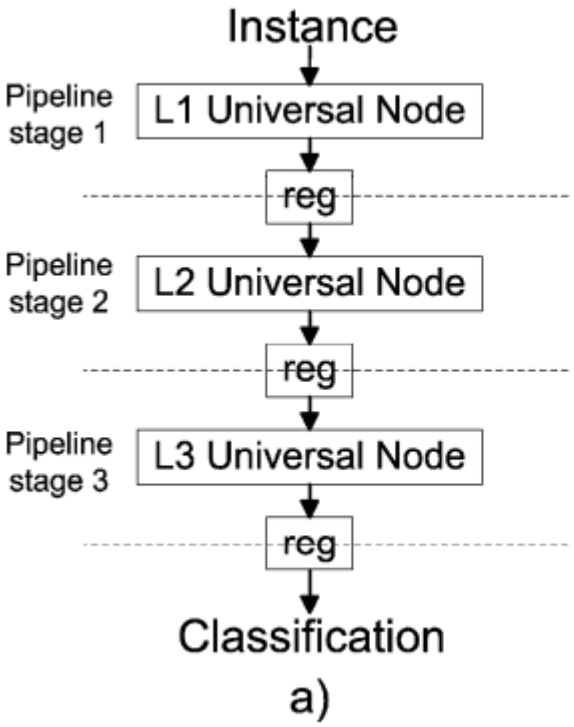
It describes some HW implementations for decision trees. They can implement **axis-parallel**, **oblique** and **non linear** DT.

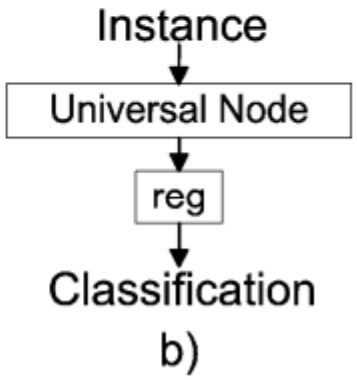
All are based on the concept of memory pointers.

There are 4 variants:

- Single module per level
- Single module per level parallel
- Universal node
- Universal node parallel.

Implementing Decision Trees in Hardware (2011)





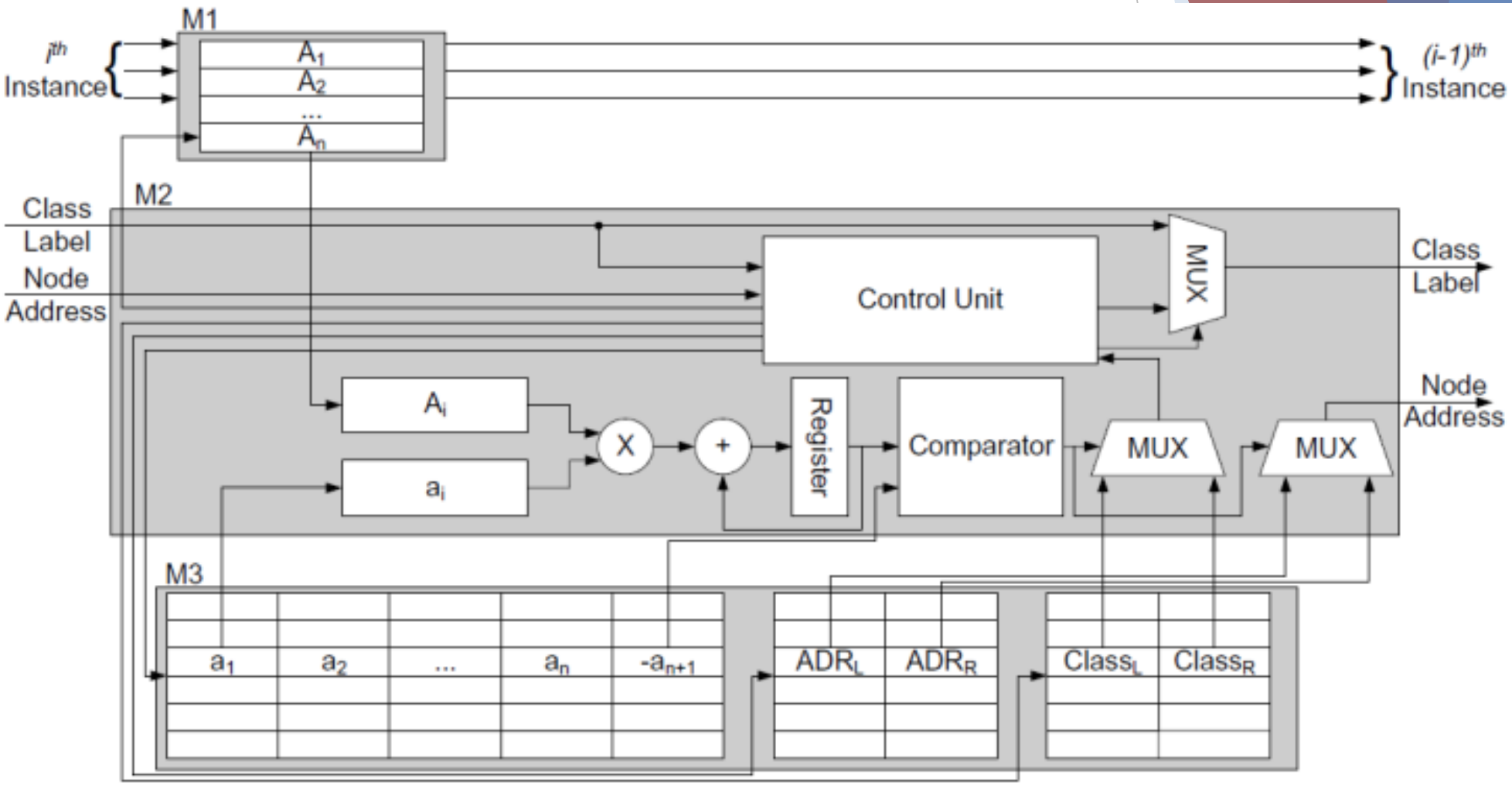
a) Basic structure of the Single Module per Level (SMpL) architecture for a DT of the depth 3.

b) Basic structure of the Universal Node (UN) architecture

Implementing Decision Trees in Hardware (2011)

sMpL Module.

According to the address received from the previous level, the node parameters are fetched from memory.

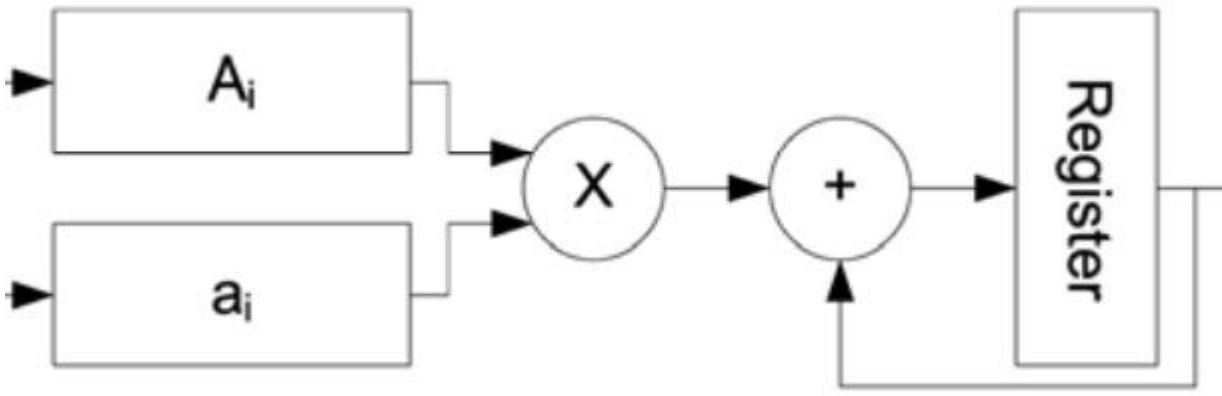


Implementing Decision Trees in Hardware (2011)

To implement oblique trees, it is necessary to perform a linear combination over some or all the attributes:

$$\sum_{i=1}^{n} a_i \times A_i + a_{n+1}$$

In HW, it can be implemented sequentially, with only one multiplier and one adder.
Or to improve performance, increasing the HW resoruces, the multiplications can be done in parallel, and a multi-input adder used.



Implementing Decision Trees in Hardware (2011)

The **SMpL** architecture works as a **pipeline**, where each stage corresponds to a level. Therefore, after the initial latency, one sample can be classified in L clock cycles, where L is the latency of one single level.

The single level latency depends on the number of attributes used in each node, and is lower for the **parallel** architectures.

The UN architecture classifies a sample after MxL cycles, where M is the number of levels the sample needs to traverse. M is not the same for all the samples.

Other memory based architectures

Other papers describe architectures similar to Universal Node and SMpL.

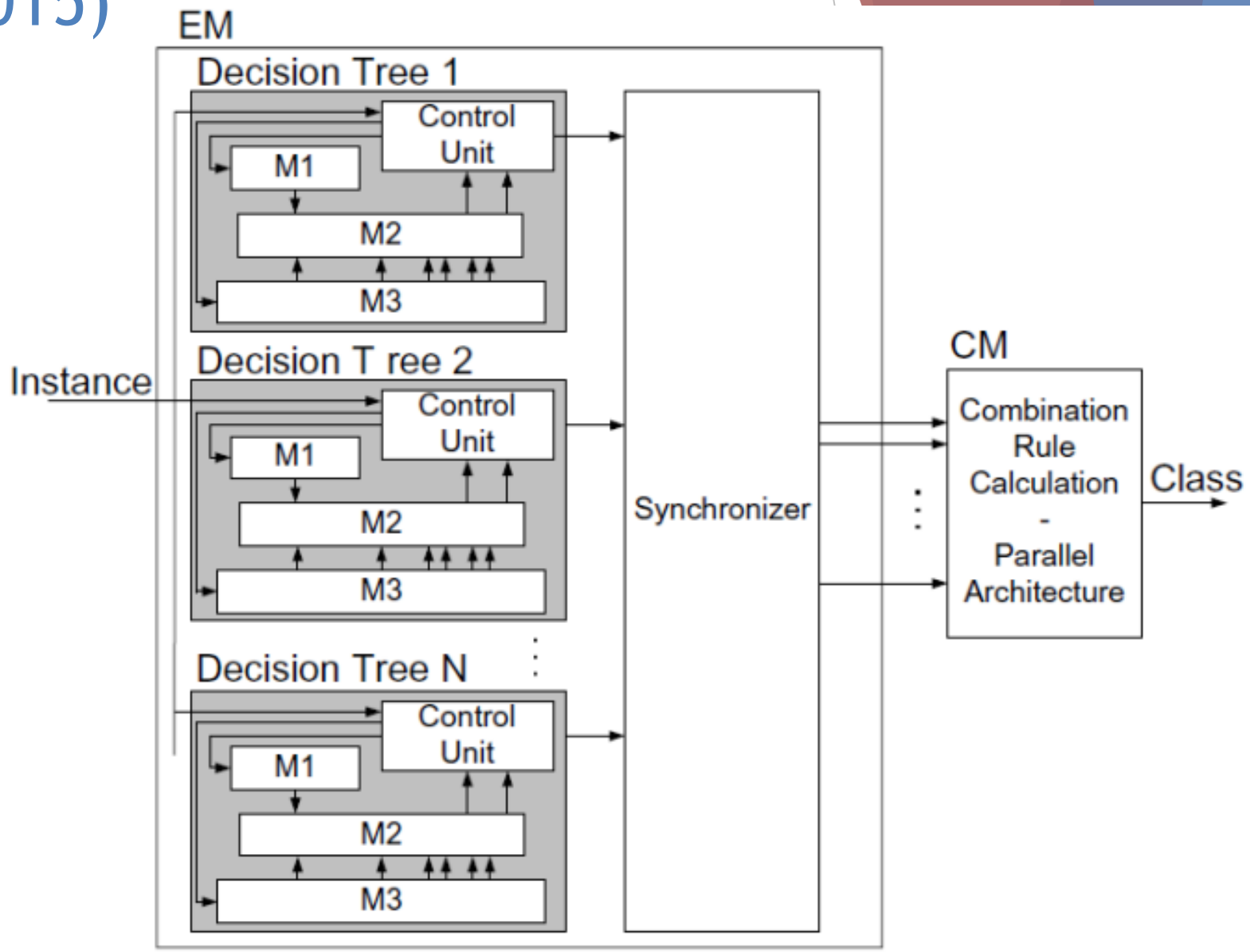
Two interesting result in *High Throughput and Programmable Online Traffic Classifier on FPGA:*
The **universal node** architecture has a better resource utilization than a **SMpL** when the tree is large.

If several instances of the same **UN** are used to classify different samples in parallel it can achieve comparable throughput to the **SMpL**

Hardware Acceleration of Decision Tree Ensemble Classifiers (2015)

To improve classification accuracy, an ensemble classifier is implemented.

The architecture consist on **N trees**, a **voter** that decides the resulting class, and a **synchronizer** to ensure all the tree results arrive at the same time to the voter.



Hardware Acceleration of Decision Tree Ensemble Classifiers (2015)

Possible architectures for the HW implementation of the voter are proposed. The general idea is to obtain the decision as fast as possible, using parallel comparisons.

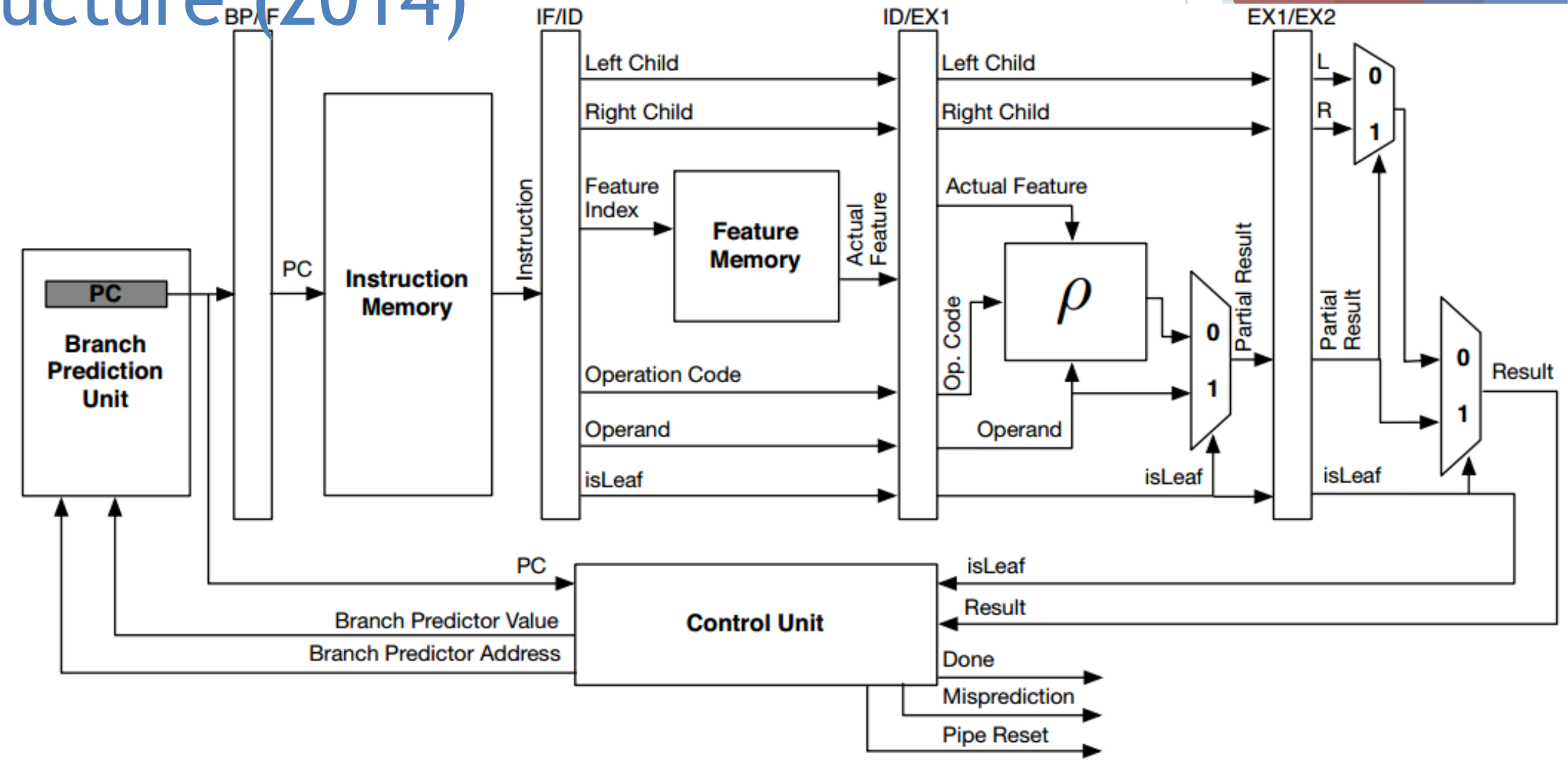
- Simple majority voting
- Unanimous voting
- Plurality voting
- Weighted voting
- Behavior knowledge space rule

A Hardware Accelerator for Data Classification within the Sensing Infrastructure (2014)

Proposes a dedicated coprocessor with the minimal necessary resources for executing a Decision tree: only **Branch**, **store** and **comparisons** instructions

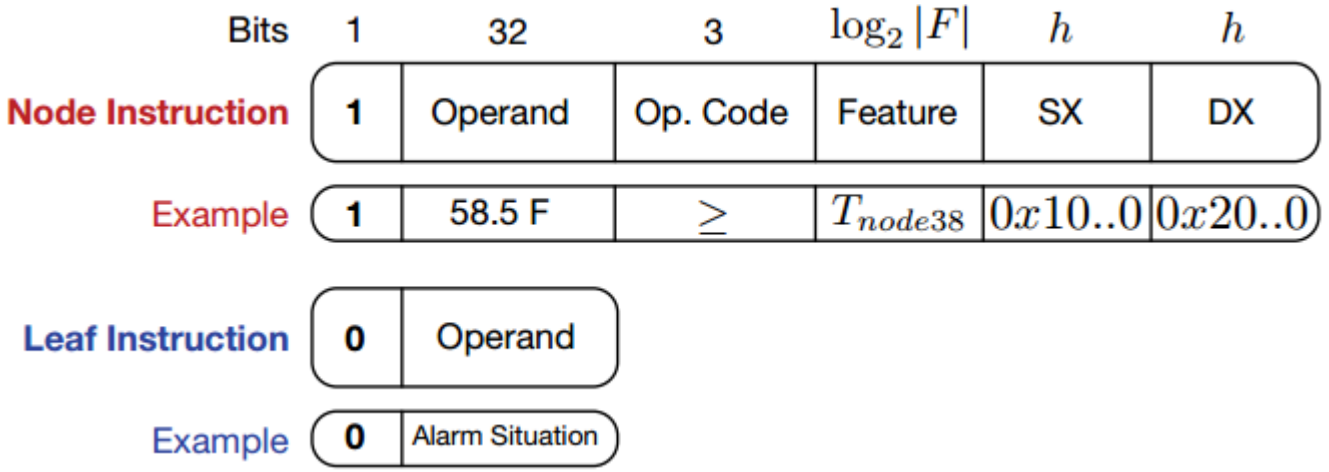
The structure of the DT is described as a sequence of instructions
The coprocessor is a pipelined processor similar to a very simple MIPS.

A Hardware Accelerator for Data Classification within the Sensing Infrastructure (2014)



A Hardware Accelerator for Data Classification within the Sensing Infrastructure (2014)

The ISA consist only on two type of instructions



A Hardware Accelerator for Data Classification within the Sensing Infrastructure (2014)

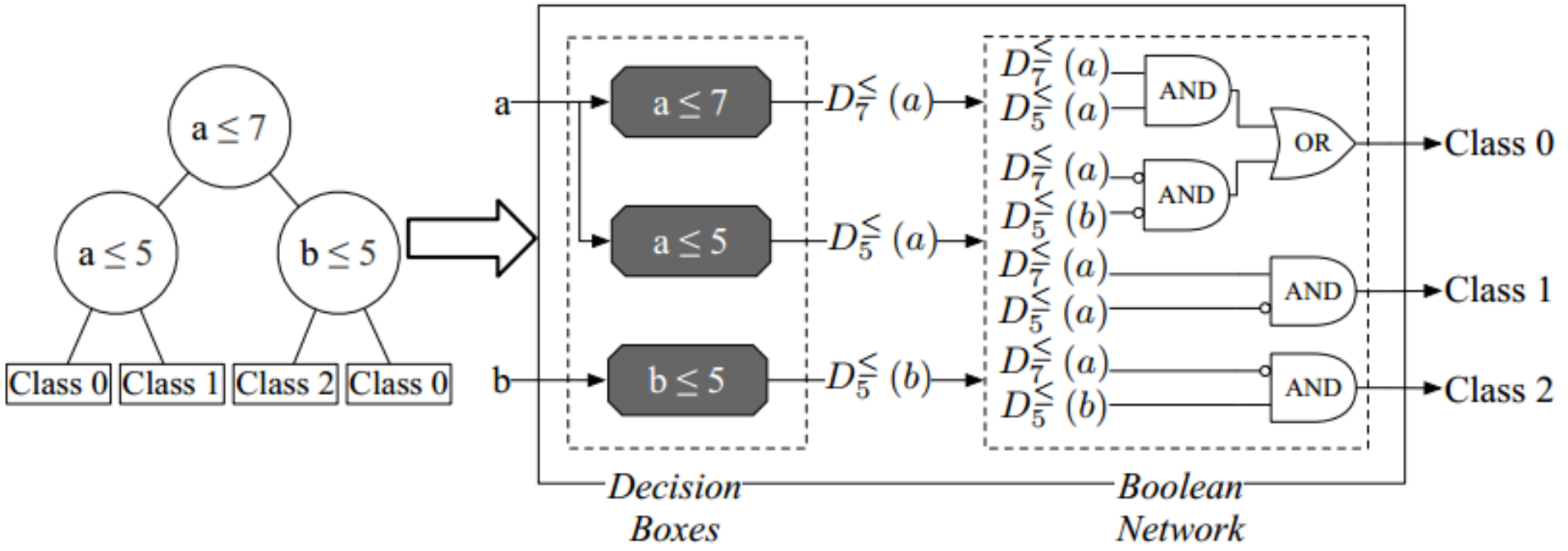
It uses **branch prediction** to increase throughput. The predictions are based on the fact that, most of the time, sequential samples belong to the same class.

Uses minimal resources as it only needs 1 module for all the tree, and can implement any binary axis-parallel tree, as the tree structure is described by instructions.

It can be considered as an hybrid btw HW and SW solutions, and it is similar to the universal node architecture

Towards Automatic Generation of Hardware Classifiers

Fully parallel approach in which each node has a dedicated HW an are all evaluated in parallel. The class output is in the form of SOP

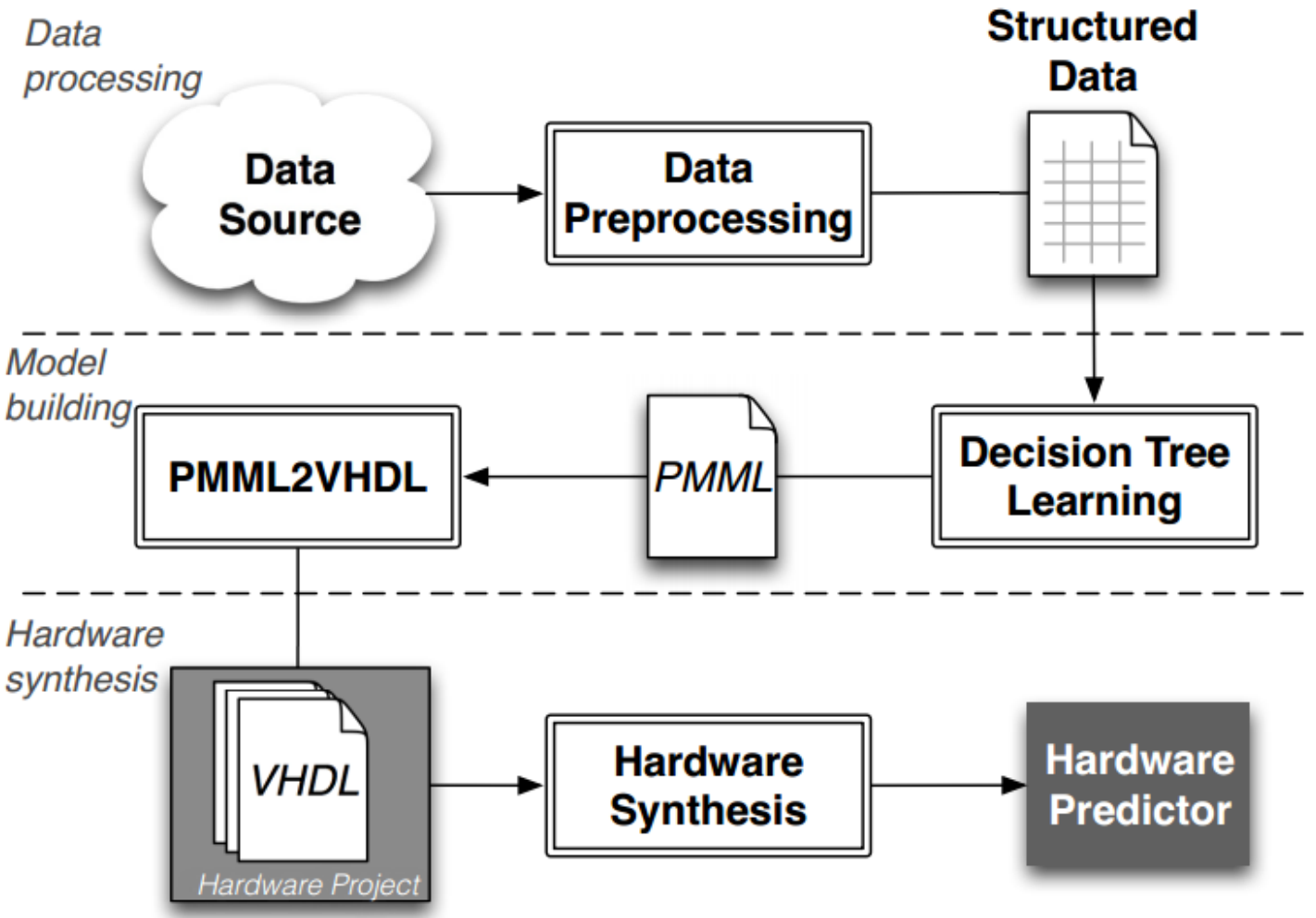


Towards Automatic Generation of Hardware Classifiers (2013)

To automate the HW tree generation, a design flow Is proposed.

Decision Tree Learning: Using the Structured Data, generates a tree using the C4.5 algorithm for induction. The result is stored in PMML format.

PMML2VHDL: Ad hoc SW to convert the PMML model to VHDL.



Hardware Decision Tree Prediction – A scalable approach (2016)

It focuses on the scalability of the previous architecture. As each node is has its own HW resource, scalability needs to be analyzed.

It is demonstrated that the number of gates needed to synthesize the fully parallel DT is at most proportional to the product of number of internal nodes and the number of features.

Design proposals

Memory based architecture + Trainer

Two HW modules:

A memory based tree. It can implement any tree as the structure of it depends on the values in the memory(ies).

- One single SMpL
- Several instances of UN

A trainer that reads the training set values and writes to the configuration memory(ies) the parameters of the obtained values (using scan chain?)

- Inter and intra distances cost function
- N virtual points approach

Boundary Tree HW implementation

Using a variation of UN or SLpM architectures. (The number of children at each node can be larger than 2)

Each node/level need to have the capability of calculating distances between samples.

May need to instantiate multiple trees, to create a forest and improve accuracy.

It works for supervised online classification, not clustering.