Novel Method for Embedded RISC Architecture to Reduce Memory Access Energy

T.Gnanasekaran[1], K.Duraiswamy[2], M. M. Arunprasath[3]

Abstract

In embedded system, reducing program size is an important goal, because storing large size codes in a smaller memory is a problem. More than one application in one embedded system has large size codes. In order to program the device efficiently, the memory size must be large enough to accommodate the large size codes. For example, PIC16F84 microcontroller has lower memory than PIC16F877 microcontroller. So the PIC16F84 must be replaced with new PIC16F877, in order to get multiple applications. But replacing the existing device with new device or component is not so easy. In this proposed system, the existing system problems must be overcome to improve the efficiency. A way to achieve this is to restrict the size of the instructions. Shorter the instructions are obtained mainly by restricting the number of bits that encode registers. This is achieved by means of compressing the repeated instructions. While executing the codes the instructions are decompressed. This method increases the efficiency of the system and also reduces the energy.

Keywords : Instruction Compression, Instruction Decompression Table, RISC Processor.

[1]Assistant Professor, Department of ECE, BannariAmman Institute of Technology, Sathyamangalam.

[2]Dean/Academic, K.S.Rangasamy College of Technology, Tiruchengode

[3]Lecturer, Aurora's Engineering College, Hyderabad
e-mail : t.gnanasekaran@gmail.com.

1. Introduction

Here a new technique is introduced for reducing the energy spent by the memory-processor interface of an embedded system during the execution of firmware code. The method is based on the idea of compressing the most commonly executed instructions i.e., the instructions used by the embedded code with the highest execution probability so as to reduce the energy dissipated during memory access. This solution allows us to fix a priori the bit width of the compressed instructions. The two-fold advantage obtained from this choice is that the size of the instruction decompression table is fixed and limited, and the instruction fetching/decompression logic has reduced complexity. Here is an architecture for instruction decompression is introduced. In this architecture, the memory bandwidth and energy required to fetch the program from memory is reduced.

Here a new technique is introduced for reducing the energy spent by the memory-processor interface of an embedded system during the execution of firmware code. The method is based on the idea of compressing the most commonly executed instructions so as to reduce the energy dissipated during memory access. A major contributor to the system power budget is the memory-processor interface [5]. For this reason, several techniques focusing on memory-processor interface power optimization have been proposed.. They can be classified into two broad classes: bus encoding techniques [3] and memory organization techniques.

These encoding schemes reduce interface power by changing the format of the information transmitted on the processor-memory bus. Memory organization methods change the way information is stored in memory and the address streams generated by the processor have relatively low transition activity. If the number of instructions used by the embedded code becomes large, it increases the number of bits of the compressed instructions. It is a major limitation. To overcome this limitation, increase the size of the instruction decompression table, this may excessively complicate the implementation of the controller that handles instruction fetching and decoding. Especially when bit-width of the compressed instructions is not compatible with the available memory addressing scheme. A solution is to compress only the instructions used by the embedded code with the highest execution probability. This solution allows us to fix a priori the bit width of the compressed instructions. The two fold advantage are size of the instruction decompression table is fixed and limited and the instruction fetching/decompression logic complexity has reduced. In this new architecture the memory bandwidth and energy required to fetch the program from memory is reduced.

2. Original Architechure

The existing system, the processor-memory architecture is shown in the figure 1. Here all the instructions are being fetched from memory. Executed with dynamic bit length k-bit size with bus architecture.

The instruction bits are being fetched by core program of the system according with instruction. The instruction code will be fetched from corresponding memory address. The instruction will be passed through desired bus architecture.

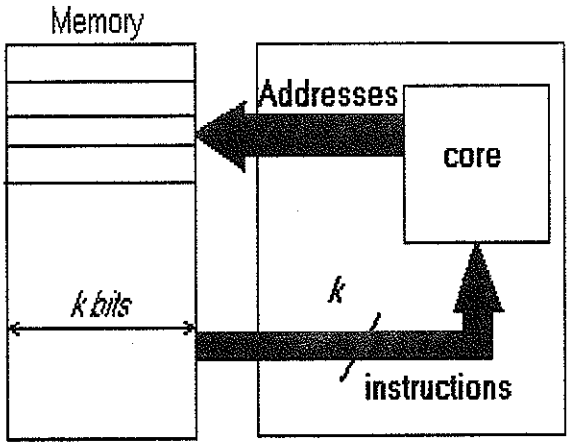

Figure 1 : Original Architechure

3. Modified Architecture

Another method which is shown in the figure 2, the system will be feed through compression system. The repeated instruction are passed through $\log_2 N$. All the instructions are decompressed inverse of $\log_2 N$. The decompression process is executed with referring Instruction Decompression Table (IDT) values. The width of uncompressed data is more than $\log_2 N$. The modified architecture [6] has the fallowing draw backs.

1. The IDT may become very large so the area required is also more to store it.
2. The bit width of the compressed instructions is comparable to bit width of original instructions, thus making negligible reduction in memory bandwidth.
3. Two bytes are used to store compressed Values of $\log_2 N$. It is not efficient.

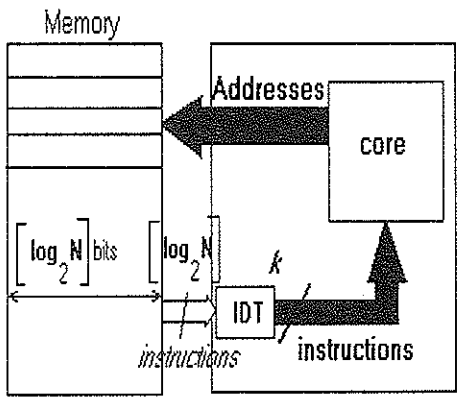

Figure 2 : Modified Architecture

4. Tree Based Compression

Another method [2] proposes a Tree Based Compression (TBC). Instructions are forming an expression tree. The Huffman encoding is such an algorithm, but designing fast Huffman decoders is complicated.

5. Proposed System: Instruction Based Compression(IBC)

In order to simplify the decompression a compression algorithm called "Instruction Based Compression" is suggested, which is based on fixed-length code words.

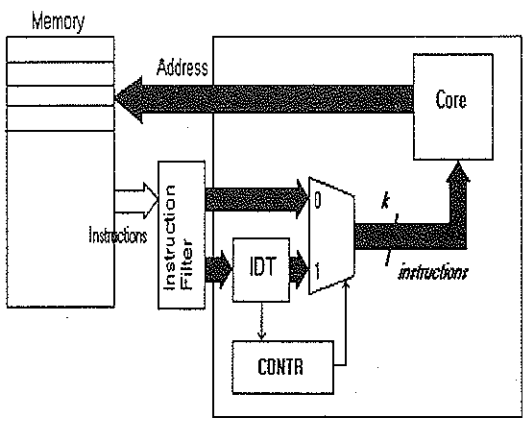

Figure 3 : Proposed Architecture

In a embedded program most of the instruction are repetitive. Taking this advantage instruction compression is proposed. i.e. Compress the instructions that are executed more often, less probable instructions are left unchanged and stored as it in the memory. This option guarantees a fixed and limited size of Instruction Dictionary Table (IDT) for all most used 256 instructions. This requires previous knowledge of a controller to handle the instruction. Because the program stored in memory is, a mix of more compressed and few uncompressed instruction set. The new proposed system for compression is based upon instruction.

Instruction based compression is motivated by the large percentage of expression trees that are composed of single instructions, the most frequent being single instruction trees [1]. Rare trees are also fairly small, while medium frequency trees are larger. So, all instruction in a program is replica of only 18.3% of its instructions, and a similar exponential behavior was again observed for single instructions. The resulting final compression ratio is on average 31.5%, and again it is achieved using only four classes. The reason for best compression is that although two entries in the TBC dictionary stores distinct trees, the trees can have similar instructions. On the other hand, entries in the IBC dictionary are unique instructions.

6. Algorithm

1. The algorithm divides the set of distinct trees into nc classes, each class having nk trees.
2. The number of classes (nc) is determined exhaustively by exploring all possible partitions from two to eight classes. Almost all programs, the minimum compression ratio is achieved when the partition is performed using four classes.

3. Each partition of a given number of classes, determine all possible combinations of class sizes and measure their compression ratio.
4. Smallest compression ratio is then selected as the best partition for that program.
5. Fixed-length code words of size [$\log_2$ nk] are then assigned to trees of class k.
6. For each codeword append a prefix of size [$\log_2$ nc], that is used by the decoder to identify the class.
7. So the compression algorithm substitutes each expression tree in the program by its corresponding prefix and codeword.

The Firmware running on a given embedded processor normally uses a small subset of the instructions supported by the processor. By replacing such instructions with binary patterns of limited width. Memory bandwidth can be reduced, thus decreasing the total energy. Binary patterns are like [$\log_2$N], where N is the number of distinct instructions appearing in the code. Here those two k-bit instructions are said to be distinct if they differ by at least one-bit.

This solution does not require adhoc compilers. The original instructions can be automatically replaced by [$\log_2$N]-bit instructions by means of a script after the subset of instructions actually used by the program is identified through instruction level simulation, and the number of [$\log_2$N] is determined. The original machine code can thus be compressed to reduce the memory bandwidth that is needed to execute the program. The so-called instruction decompression table and the related control circuitry can be designed and placed between the processor and memory. Hence, the architecture of the core processor is left unchanged.

Example:

```
Initialize
        clrf porta
                clrf portb
                bcf status,6
                bsf status,5
        movlw b'11011011'
        *movwf porta
        movlw b'00000000'
        *movwf portb
    ;   movlw b'00000111'
    ;   *movwf adcon1
        movlw b'10000100'
        *movwf adcon1
        bcf status,5
        retlw 00
;----------------------------------------
setdata         andlw 0Fh
                *movwf temp1
                btfss temp1,0
                bcf portb,0
                btfsc temp1,0
                bsf portb,0
                btfss temp1,1
                bcf portb,1
                btfsc temp1,1
                bsf portb,1
                btfss temp1,2
                bcf portb,2
                btfsc temp1,2
                bsf portb,2
                btfss temp1,3
                bcf portb,3
                btfsc temp1,3
                bsf portb,3
                retlw 00
;----------------------------------------
```

*movwf temp
swapf temp,w
call setdata
LCDEN
LCDDIS
movf temp,w
call setdata
LCDEN
LCDDIS
movlw 02
call msecond
retlw 0

Instructions with prefix '*' indicates the same instruction which are repeated more than once in a program. These instructions are to be compressed by using the above-mentioned algorithm.

7. Decompression

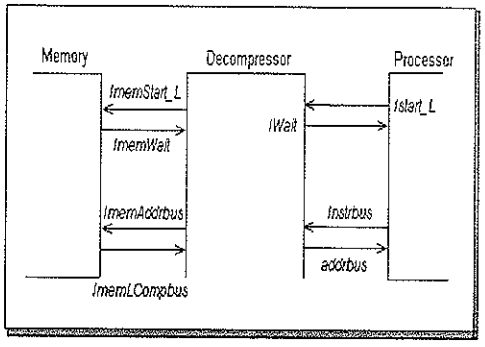

Figure 4. Decompressor Interface

The block diagram of the de-compressor interface with processor and instruction memory is shown in Figure 4. The core communicates with the de-compressor using the static instruction memory interface based on a four-phase process.

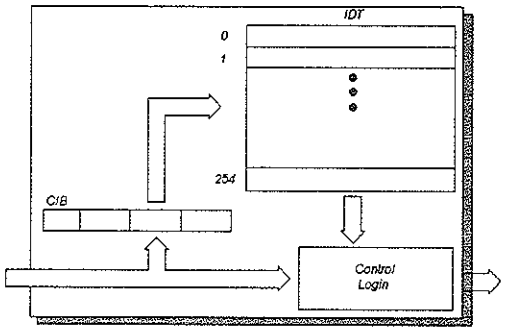

Figure 5. De-Compressor Block Diagram

The de-compressor block diagram is shown in Fig.5. The IDT contains 255 32-bit words. The address each word is the compressed code of the instruction stored in the word.

Decompression is performed by reading the instruction the content of IDT at the specified address by the byte of compressed instruction. The compressed instruction buffer having 32-bit register that can be accessed byte-by-byte from the memory. The results coming from memory are stored in the CIB and are decompressed one byte at a time. Final stage the control logic block gets the instruction interface signal, IDT lookup and CIB read/ write images that direct transfer from memory to processor of compressed instruction.

The key function of logic control block is address generation in refer with the IDT values. If the processor reads the address in sequence the controller generates new memory address every four processor fetch cycles. The remaining three cycles, compressed instructions are executed from CIB. On the contrary if either the processor is the destination address of a branch/jump or a mark is read a cycle to instruction memory is initiated. The IDT is an asynchronous SRAM read with macro generator provided within the STM CAD Design kit. A small

amount of logic has been added between the processor and de-compressor to the MIPS/ DLX interface signals into the standard described above.

The de-compressor is a single clock edge triggered design while the processor uses a two-phase non-overlapping clock. The area and energy consumption of the de-compressor as well as the critical path delay are dominated by the IDT. IDT is a 1-Kbyte SRAM. When the de-compressor is processing compressed instructions, it performs one memory bus cycle every four fetch cycles. When it does not involve CIB refill fetch time for compressed instructions reduces to IDT read time. This is the most common case. In the remaining cases fetch latency is longer. Memory access time plus IDT read time is the time required for fetching a compressed instruction immediately after a CIB refill. The worst fetch time is experienced when the first instruction after a CIB refill is not compressed here two instruction memories reads to fetch an instruction.

Results:

Data Validity:

S.No.	Input File size in KB	Compressed File Size in KB	Compressed Ratio in %
1.	136	57	41
2.	104	38	36
3.	81	36	44
4.	33	7	21
5.	24	6	25
6.	19	8	42

8. Conclusion

By using the selective instruction compression technique, the memory access is reduced and the energy spent during the memory access is also reduced. Therefore, the efficiency will be improved. The compression technique can be adopted for data also. However, data compression is not as efficient as instruction compression because the total number of repeated data is less than the total number of repeated instructions. So the compression technique.

References

[1] C.L.Su, C.Y.Tsui. and A.M.Despain, *"Saving power in the control path of embedded processors"*, IEEE Design Test Compute., Vol.11, No.4, PP 24-30, 1994

[2] L.Benini, A.Macii, E.Macii, M.Poncino and R.Scarsi, *"Architectures and synthesis algorithms for power efficient bus interfaces"*, IEEE Trans.Computer-Aided Design, Vol.19, PP.969-980, Sep 1999

[3] Y.Yoshida, B.Y.Song, H.Okuhata, T.Onoye and I.Shirakawa, *"A Project code compression approach to embedded processors"*, in ACM Int.Symp.Low Power Electronics and Design, Monterey, PP.265-268, 1997

[4] T.C.Bell, J.G.Cleary and I.H.Witten, *"Text Compression,ser.Advanced Reference Series. "* Englewood Cliffs, NJ:Prentice-Hall, 1990

[5] G. Araujo, P.Centoducatte, R.Azevedo and R. Pannain, *"Expression tree based algorithms for code compression on embedded RISC architectures"*, Institute of Computing, Univ. of Campinas. http:// www.dcc.unicamp.br/ic-main/publications-e.html, Jan 2000

[6] David A. Huffman, *"A Method for the Construction of Minimum-Redundancy Codes"*, Proceedings of the I.R.E., Vol.4D, PP.1098-1101.

[7] Kozuch.M and Wolfe.A, *"Compression of embedded system programs"*, in Proceedings of IEEE International Conference Computer Design, Cambridge, PP.270-277, Oct 1994.

[8] Lekatsas.H and Wolf.W, *"Code Compression for Embedded Systems"*, Proceedings of Annual ACM/ IEEE Design Automation Conference, PP. 516-521. June 1998.

[9] Wolfe. A and Chanin. A, *"Executing Compressed Programs on an Embedded RISC Architecture"*, proceedings Of 25th Ann. International Symposium on Microarchitecture, PP.81-91, Dec 1992.

[10] http://www.Woundedmoon.org.

Author's Biography

T.Gnanasekaran graduated from University of Madras in 1989 and completed his post graduate from Anna University, India. He is presently with BannariAmman Institute of Technology, Sathyamangalam, Tamilnadu, India, as Assistant Professor and working towards his Ph.D, in the area of Error Control Coding. His other interests include signal design for WiMax and modulation technique.

Dr.K.Duraiswamy is working as Dean / Academic, K.S.Rangasamy College of Technology, Tiruchengode-637 209, Tamilnadu, India. His areas of interests are Computer communication and Networking.

A Novel Technique for Web Page Informative Content Extraction

M.Karthikeyan[1], Krishnan Nallaperumal[2], K.Senthamarai Kannan[3], T.Rajesh[4]

ABSTRACT

The web has established itself as the dominant medium for doing electronic commerce. Consequently the number of service providers, large and small, advertising their services on the web continues to proliferate. In this paper, new extraction algorithms for mining information from web pages are proposed. Search engines crawl the World Wide Web to collect web pages. These pages are either readily accessible without any activated account or they are restricted by username and password. Whatever be the way the crawlers access these pages, they are (in almost all cases) cached locally and indexed by the search engines. An end-user who performs a search using a search engine is interested in the primary informative content of these web pages. Non-content blocks are very common in dynamically generated web pages. Typically, such blocks contain advertisements, image-maps, plug-ins, logos, counters, search boxes, category information, navigational links, related links, footers, headers and copyright information. These non content sections must be removed from the pages to get the content blocks so that mining of knowledge can be done. The proposed algorithm outperforms to extract the content and also the space needed for storage requirements is also minimized by similarity measurements. This paper deals with the problem of identifying and extracting the informative contents of a web page

[1]Research Scholar, Centre for Information Technology & Engineering, Manonmaniam Sundaranar University, Tirunelveli - 627 012.

[2]Professor and Head, Centre for Information Technology and Engineering, Manonmaniam Sundaranar University, Tirunelveli - 627 012.

[3]Professor, Department of Statistics, Manonmaniam Sundaranar University, Tirunelveli-627 012.

[4]Lecturer, Shirdi Sai Engineering College, Bangalore, India.

1. INTRODUCTION

Extraction algorithms for mining information from web pages are described in this paper. Also a propagation technique for identifying and accumulating all of the attributes related to a service entity in a web page is proposed.

A substantial part of the web pages, especially those that are created dynamically contains information that should not be classified as the primary informative content of the web page. These blocks are seldom sought by the users of the website. Such blocks are referred as non-content blocks [1].

Before the content of a web page can be used, the non-content blocks must be removed based on the tags and some special features [2], so that an end user can get the required contents easily. We have designed two algorithms, based on the general need of the web users, one to extract the required text features and the other to extract the image features from the web pages. To extract the textual content, our algorithm crawls the web pages and removes the non-content information. Once the non-content sections are removed, the remaining can be considered as the content sections [10]. Several algorithms are there to find the primary informative contents from the web pages. Most of them are based on the DOM (Document Object Model) tree of the web

pages. However, because of the flexibility of HTML syntax, a lot of web pages do not obey the W3C (World Wide Web Consortium) HTML specifications, which might cause mistakes in DOM tree structure [7]. Moreover, DOM tree is initially introduced for presentation in the browser rather than description of the semantic structure of the web pages. For example, even though two nodes in the DOM tree have the same parent, it might not be the case that the two nodes are more semantically related to each other, than to other nodes [7]. HTML pages are developed by means of the HTML tags as per the W3C. Our algorithm crawls in to the web pages based on the tags and some key features and removes the non-content sections. Certain non-content sections may have their own specific tags or else they are removed by identifying their specific features [2], which made it easier to remove the non-content sections.

The input to the proposed algorithm is a web page and this algorithm crawls the web page based on the tags and the predefined set of specific functions to remove the unnecessary sections. Once the non-content sections are removed the remaining is considered as the content sections. Before the extracted contents from a web page can be used, it must be subdivided into smaller semantically homogeneous sections based on their contents [1]. Such sections are called blocks. A block 'B' is a portion of a web page enclosed within an open-tag and its matching close-tag, where the open and close tags belong to an ordered tag-set 'T' that includes tags, such as <TR>, <P>, <HR> and <UL>[1]. Text parts alone are extracted from each blocks, based on certain features [1]. Also the system is designed to checks the similarity of each blocks using Competitive Neural Network (CNN) pattern recognition mechanism. The first block is taken as the input and it will be compared with all the other blocks in the page. We used a threshold ( ) with a numerical value of 0.75 in our implementation. If the similarity between the two blocks is greater than the threshold then the blocks are considered as similar and only one block is stored. This minimizes the storage requirements. The threshold can be varied based on the requirements. Several familiar web pages were examined and our algorithm produces best result comparable to other similar algorithms.

An algorithm to retrieve the images from the web pages is also provided. If the user is interested to view only the image or may be, the page is a specialized image based web page, then the user can get best results with our algorithm. As per the W3C guidelines the images are in the tag <IMG>. The proposed algorithm crawls the web page to retrieve the particular tags and a path is made to folder where the images are stored. Only the images that are informative alone is displayed, which is identified by the use of pair of tags into which the images are embedded.

2. Identification Of Non Content Information

As per the guidelines of the W3C, almost in all the web pages, either all or most of the non-content information mentioned earlier are available. The way the designers develop a web page, it may have one or more presentation styles used in it, which leads to the addition of non-content blocks. As well, the contents inside these non-content blocks may have a similarity. The proposed algorithm identifies these presentation styles and or the common contents by the use of a set of inputs provided to the algorithm. The inputs may be some ordered tags or some common features. The algorithm is designed so that the input set can be altered at any time depending on the application. For example the anchor tag <A> is mainly used for the link. The default behavior associated

with a link is to redirect the user to another web resource. Hence this information can be removed by the use of the tag set as an input to remove the non content sections. Like this our algorithm functions and removes all the non-content information and separate the content part as detailed below.

Input : HTML pages H1, H2,...Hn (H), sorted tag and Features (T).

Output : Content Blocks and their associated Page.

Begin

1. **For** each HTML page
2. read H1
3. **For** each input tag and feature **Do**
4. Read input tag and feature
5. **If** non-content blocks present **Then** remove all
6. **End for**
7. Remove tags
8. Store the output blocks B1, B2, ...Bn
9. **For** each block
10. Read Bx
11. Read B1, B2,.....Bn excluding Bx
12. **Sim**(Bx : By)
13. **If** similarity (Bx : By) > ▲
14. **Return** Bx
15. **Read** next B
16. **End for**
17. **Read** next input tag and feature
18. **End for**
19. Read next (H)
20. **End**

Function sim :

Input : Block1, Block2

Output : Similarity measure

Begin

S1 ⟵ String (Block1)

S2 ⟵ String (Block2)

Return Sim(B1 : B2) (percentage)

End

3. Block Features

As mentioned in the previous section, in a HTML web page, let block B1 and block B2 are the portions of the web page, enclosed with an open and its matching close tag. The algorithm is designed in such a way to perform the search operation based on Breadth First Search (BFS) method [1],[9]. The input set of tags as per the W3C guidelines is given. This algorithm takes an input tag from the set of tags and performs the search. If a matching tag set is found, that block is separated or else, the total contents will be the output. Once a block is found, once again BFS is performed in to the separated block to find existence of any matching block is present inside it. If any matching block is found it is also separated.

For example a table can be created using the HTML tag <TABLE> and inside this table, table rows are created using the <TR> tag and the table data are created using the tag <TD>. By the use of the BFS, each and every block is searched for all the input set of tags and the result of the search brings the atomic blocks. Several features are added with their standard tags but not all. Atomic blocks may have features like text, images, applets, java scripts etc. The input tag set includes text, text tag, list, table, object, frame, form and script [8]. If any of the other features are important, our frame work can be modified by adding the new features.

4. Identification Of Similar Blocks

The separated blocks of each web page is stored as tree in a buffer. All the blocks are checked for the similarity between them. The Competitive Neural Tree [3], [4], [5], has a structured architecture, used to identify similar content blocks. A hierarchy of identical nodes forms an m-ary tree as shown in Fig.1(a) and Fig.1(b) shows a node in detail. Each node contains m slots s1, s2, . . . , sm and a counter age that is incremented each time an example is presented to that node. The behavior of the node changes as the counter age increases. Each slot si stores a prototype pi, a counter count, and a pointer to a node. The prototypes pi € P have the same length as the input vectors x. They are trained to match the patterns obtained from each node.

The slot counter count is incremented each time the prototype of that slot is updated to match an example. Finally, the pointer contained in each slot may point to a child-node assigned to that slot.

A NULL pointer indicates that no node was created as a child so far. In this case, the slot is called terminal slot or leaf. Internal slots are slots with an assigned child-node.

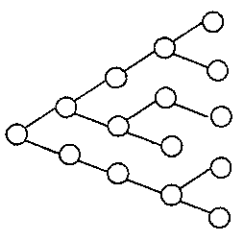

Figure 1 (a) Tree Structure

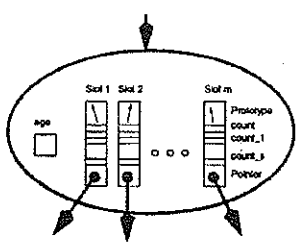

Figure 1 (b) Explanation of individual node

A. Learning at the Node-Level

In the learning phase [5] the tree grows starting from a single node, the root. The prototypes of each node form a minuscule competitive network. All prototypes in a node compete to attract the examples arriving at this node. These networks are trained by competitive learning. When an example $x \in \gamma$ arrives at a node, all of its prototypes $p_1, p_2, \ldots, p_m$ compete to match it. The closest prototype to x is the winner. If $d(x, p_j)$ denotes the distance between x and p_j, the prototype p_k is the winner if $d(x, p_k) < d(x\, p_j)$ ¥ $j \neq k$. The distance measure used in this paper is the squared Euclidean norm, defined

as $$d(x, P_j) = \|x - p_j\|^2 \quad (1)$$

The competitive learning scheme used at the node level resembles an unsupervised learning algorithm proposed to generate crisp c- partitions of a set of unlabeled data vectors [5],[6]. According to this scheme, the winner p_k is the only prototype that is attracted by the input x arriving at the node. More specifically, the winner p_k is updated according to the

equation $$P_k^{new} = P_K^{old} + \alpha(x - P_k^{old}) \quad (2)$$

where α is the learning rate. The learning rate α decreases exponentially with the age of a node according to the

equation $$\alpha = \alpha_0 \exp(-\alpha_d age) \quad (3)$$

where α_0 is the initial value of the learning rate and α_d determines how fast α decreases. The update equation (2) will move the winner P_k closer to the example x and therefore decrease the distance between the two. After a sequence of example presentations and updates, the prototypes will respond each to examples from a particular region of the input space. Each prototype P_j attracts a cluster of examples R_j.

The prototypes split the region of the input space that the node sees into sub regions. The examples that are located in a sub region constitute the input for a node on the next level of the tree that may be created after the node is mature. A new node will be created only if a splitting criterion is true.

B. Life Cycle of Nodes

Each node goes through a life cycle. The node is created and ages with the exposure to examples. When a node is mature, new nodes can be assigned as children to it. A child-node is created by copying properties of the slot that is split to the slots of the new node. More specifically, the child will inherit the prototype of the parent slot. Right after the creation of a node, all its slots are identical. As soon as a child is assigned to a node, that node is frozen. Its prototypes are no longer updated in order to keep the partition of the input space for the child-nodes constant. A node may be destroyed after all of its children have been destroyed.

C. Training Procedure

The generic training procedure is described below:

Do while stopping criterion is FALSE:

- Select a block.
- Traverse the tree starting from the root to find a terminal prototype P_k that is close to x. Let n_l and s_k be the node and the slot that P_k belongs to, respectively.
- If the node n_l is not frozen, then update the prototype P_k according to equation (2).
- If a splitting criterion for slot S_k is TRUE, then assign a new node as child to S_k and freeze node n_l.
- Increment the counter count in slot S_k and the counter age in node n_l

Depending on the type of the search method, the second step is implemented and various learning algorithms can be developed. The search method is the only operation in the learning algorithm that depends on the size of the tree. Hence, the computational complexity of the search method determines the speed of the learning process.

Sample pseudo code used for training the network

/*Assigning inputs to each neuron*/

- Set the initial value i=0
- For all neuron In Input
- Neuron.Output = Inputs(i)
- i = i + 1
- End

/*Calculating the weight of each neuron*/

- For all input neuron connected to This Neuron
- netValue = netValue + (Weight Associated With InputNeuron * Output of InputNeuron)
- End

/*Calculating the error value */

- Delta = Neuron.Output * (1 - Neuron.Output) * ErrorFactor

/*Calculating the output */

- For each layer in Input layers
- neuron.Update(Input* Weight)
- End

/*Calculating the Bias Value */

- Set netValue As Single = bias
- For all input neuron connected to ThisNeuron
- netValue = netValue + (Weight Associated With InputNeuron * Output of InputNeuron)
- End

A human expert is able to provide a better threshold value to get the similarity. Based on various experiments performed on different web sites, a threshold value of $\Delta = 0.75$ is decided, which produces good results. Depending on the application this threshold can be varied to produce a better output [10]. The first block from the buffer is taken and it is compared with all the other blocks in the buffer. If the similarity value between the blocks is greater than the threshold then, the two blocks are considered to have similar contents and only one block is stored. This minimizes the space requirement for the storage purposes. Then the next block is taken and the same procedure of comparison is repeated until all the blocks are compared with all other blocks. Hence the output will not have any redundant blocks and non content blocks.

5. Removal Of Images

The system is designed to retrieve images from the web pages. The Feature Extractor algorithm is able to identify only the text contents. The proposed algorithm is designed so that it can also able to identify the image features and display it on the basis of the users' interest. Initially the images are identified by the use of the <IMG> tag, which is a standard tag for the images in HTML pages as per W3C guidelines. After the image tags are identified a path to the image where it is stored is made and the images are retrieved. In the present work the images with the extension .jpg is alone retrieved as most of the web page designers use the JPEG standard images for the informative part. Images in other format may be used for the advertisement parts. It is also possible to alter the algorithm if any other format images are necessary. Also based on the size (count) of the image pixels the image can be decided as informative or non-informative. This algorithm will find a good application where the users are interested only on the images.

Input : HTML pages H, tag <img>

Output : Images

Begin

1. **For** all HTML pages
2. **Read** H
3. **If** <img> present **Do**
4. Extract the Block and store as I
5. **For** all image blocks I
6. **Read** I
7. **If** extension (.JPEG) **Do**
8. Link Image source
9. Display the Image
10. **Else** read Next (I)
11. **Else** read Next (H)
12. **End** for
13. **End** for

End

6. Experimental Evaluation

In this section, we present an empirical evaluation of our method. We also compare with two other related works.

A. b-Precision

Precision is defined as the ratio of the number of relevant items (actual primary content blocks) r found and the total number of items t (content blocks suggested by an algorithm) found. As the precision is calculated for blocks it is called as b-Precision. **b-Precision = r / t**

B. b-recall

Recall has been defined as the ratio of the number of relevant items found and the desired number of relevant items. The desired number of relevant items includes the number of relevant items found and the missed relevant