Algorithm and Architecture for a Multiple-Field Context-Driven Search Engine Using Fully-Parallel Clustered Associative Memories

Hooman Jarollahi*, Naoya Onizawa†, Vincent Gripon‡, Takahiro Hanyu† and Warren J. Gross*
*Department of Electrical and Computer Engineering, McGill University, Montreal, Quebec H3A 0E9
† Research Institute of Electrical Communication, Tohoku University, Sendai, Japan
‡ Electronics Department, Télécom Bretagne, Brest, France.
Email: hooman.jarollahi@mail.mcgill.ca, warren.gross@mcgill.ca

***Abstract*—In this paper, a context-driven search engine is presented based on a new family of associative memories. It stores only the associations between items from multiple search fields in the form of binary links, and merges repeated field items to reduce the memory requirements. It achieves 13.6× reduction in memory bits and accesses, and 8.6× reduced number of clock cycles in search operation compared to a classical field-based search structure using content-addressable memory. Furthermore, using parallel computational nodes in the proposed search engine, it achieves five orders of magnitude reduced number of clock cycles compared to a CPU-based counterpart running a classical search algorithm in software.**

I. Introduction

There is a significant need for energy-efficient context-driven search (CDS) engine [1] due to the drastic growth in the amount of stored information in complex digital databases such as DBLP, Twitter, YouTube, or LinkedIn. In such applications, *items* from multiple *search-fields* are often queried to refine the search results.

Conventional software-based multiple-field search engines such as [1] consume high energy since exhaustive back-and-forth memory-access operations through I/O drivers, and also high-latency comparison operations are performed. Content-Addressable Memory (CAM)-based [2], [3], Trie-based [4], and Hash-based schemes [5], [6] are alternative search approaches that when used in hardware, are typically intended for high-speed Longest-Prefix Matching (LPM) in network processing, for which fast IP-lookup operations are required against the stored contents.

CAMs employ a brute-force search scheme by matching an input search-word against all of the CAM entries in an attempt to find a matched word or words that point to the corresponding output in another memory module, that is typically a Static Random Access Memory (SRAM). Therefore, CAMs consume large amounts of dynamic, as well as standby energies in modern CMOS technologies due to the increase of leakage energy dissipation of SRAMs. Another reason for high energy consumption of CAMs, when used for multiple-field search applications, is due to their data-storage inefficiency. In order to be able to search for a set of outputs, associated with a single input item, the shared input item must be stored redundantly for every association. For example, if an input item "Blue" is associated with two different entries, "A", and "B", two copies of "Blue" are stored in the CAM array dedicated for the search field of "Color". On the other hand, if a single output is associated with multiple input items, redundant SRAM words are occupied with the same output. These redundancies, especially if full text of the items and outputs are stored, result in increasing the memory usage, the number of search operations, and thus standby and dynamic energy consumptions.

The two-field structure of the CAM-SRAM array, i.e. tag-output, requires dedicating an array for each search field. Therefore, independent search operations result in large number of outputs, whose intersections are the actual desired search results. Therefore, large delays in transferring and post-processing of the independent search results are required.

Trie-bases search structures perform the search operation in a tree-like process searching a few bits at a time. This approach results in multiple-levels of searching, and thus increases the latency and the memory usage to store pointers from nodes to the children [5]. Hash-based search architectures store compressed versions of the information and often require large-memory usage, and result in output collisions which would require post-processing operations that typically incorporate multiple hash-tables for each length of the stored entry. This is unattractive since the length of an item in a search engine can be arbitrary unlike that of the IP addresses in networking applications.

In this paper, we present a system-level algorithm and architecture of an associative memory-based multiple-field search engine. L-SCAN (Link, Sparse, Context, Associative, Network) reduces the memory requirement of the search operation and can be tuned to confine the search scope to either one or a few possibilities. A false-negative result never occurs, i.e., if a matched entry exists, it is always found.

The proposed system is an application-specific variant of a recently-introduced family of associative memories based on Sparse Clustered Networks [7]–[9] (SCN). It eliminates the necessity for storing the text of the search items and the outputs. Consequently, it also eliminates the energy-hungry brute-force search operations. Furthermore, in contrast to CAMs, it requires a single storage of repeating items as well as outputs, as well as the associations between them for multiple database entries.

Once the proposed architecture is trained with the associations, it retrieves associated outputs given partial input-items, without storing the data bits directly, which would otherwise lead to a large memory requirement as in CAMs. Instead, only the associations between the related items from different search-fields are stored in the form of binary links. The location of such binary links are determined by the values of the items. What makes the data-representation different from a hash-based counterpart is that in the proposed system input items are segmented into multiple parts when required, which reduces the memory requirement. This segmentation is performed when the number of different items a search field is required to cover is large. Therefore, the length of each database entry is considered only to avoid collisions. Once the length is enforced by the data-distribution in the application, the diversity of the database entries is realized in selecting the hardware-parameters.

The organization of the paper is as follows: In Section II, a system-level algorithm and architecture is presented. Characteristics of the proposed system are evaluated in Section III. In Section IV, it is compared with classical approaches in terms of the memory requirement and delay. Section V concludes the paper.

II. An Associative Memory-Based Multiple-Field Search Engine

The proposed system is depicted in Fig. 1 showing the communication structure of a co-processor with the main processor running the interface software program, and the external storage device. A database entry consisting of fields, including inputs and outputs, is also shown.

The proposed co-processor is an SCN-based associative memory performing multiple-field search operations by recovering the previously trained search results given a collection of input items from different search fields. The training process is performed prior to the search operation. For example, in an article-publishing database, field items such as specific keywords, list of authors, publication year, and the text of the paper are stored in a file, with a unique file ID. The files are stored in the external storage device.

The generated search results from the co-processor may include a few detectable false-positives, which can be filtered externally in software or directly displayed. However, a false-negative never occurs. The number of false positives depends on the density of the used connections. In this paper, the ratio between the average number of retrieved results, including the desired ones, to that of the total possibilities in the search scope is referred to *Search Focus Rate* (SFR). SFR is tunable using the design parameters of the proposed system, some of which influence its hardware complexity.

The role of the co-processor during search is thus to focus the search scope from a large number of possibilities to a few

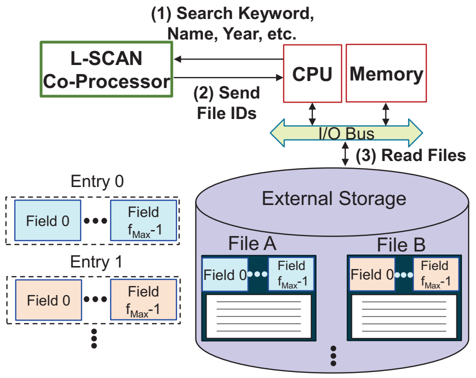

Fig. 1. System level block diagram of the proposed system showing how the co-processor communicates with the CPU during search.

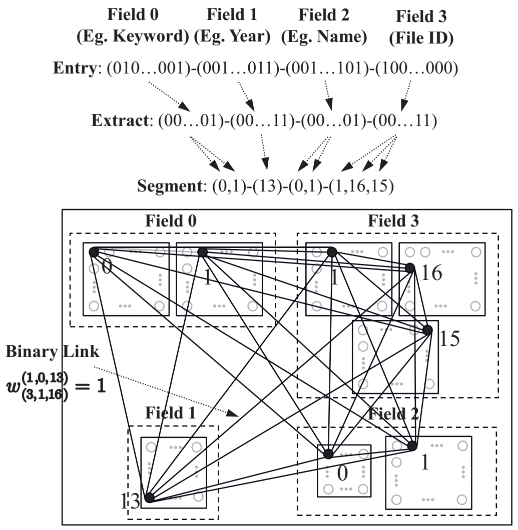
Field 0 (Eg. Keyword)
Field 1 (Eg. Year)
Field 2 (Eg. Name)
Field 3 (File ID)
Entry: (010…001)-(001…011)-(001…101)-(100…000)
Extract: (00…01)-(00…11)-(00…01)-(00…11)
Segment: (0,1)-(13)-(0,1)-(1,16,15)
Binary Link
$w^{(1,0,13)}_{(3,1,16)} = 1$
Field 0
Field 1
Field 2
Field 3
0
1
13
1
16
15
0
1

Fig. 2. An example of the proposed SCN-based multiple-field search engine consisting of 4 fields with various number of clusters in each field and various number of nodes in the clusters.

ones such that the number of generated results is much smaller than the total number entries a CPU-based alternative would search through by itself otherwise.

The average number of generated search results is tunable by optimizing the design parameters of the proposed system such as the length of the inputs, and the total number of database entries.

The SCN-based associative memory used in the co-processor is illustrated in Fig. 2 using an example showing a network consisting of four fields: three input fields (Field 0 - Field 2), and an output field (Field 3). In general, it consists of f_{Max} fields, where each field consists of c_f $(0 \leq f \leq f_{Max} - 1)$ clusters. The number of nodes in the i-th cluster $(0 \leq i \leq c_f - 1)$ of the f-th field, $l_{(f,i)}$, is not necessarily equal in all clusters. The concatenation of the indices of one node per cluster in all clusters of a field is the place holder of an item. The connections within the clusters of a field permit reconfigurability of the system, since depending on the application, the clusters can be grouped in other fields if required. The operation of the proposed system is realized in two stages: (i) Training the co-processor: where association between the related search items and the search results (File IDs) are stored in the co-processor. (ii) Search operation: where the file contents are searched by presenting the known items in the search fields.

A. Training

Training process of the proposed co-processor is performed by finding and storing the association of the field items. During the training stage of the co-processor, a series of operations are performed:

1) Data Collection: A database entry consisting of items from various search fields and the associated file ID(s) is obtained. Each search field covers a collection of search items that are shared among multiple database entries unlike the CAM-based or CPU-based counterparts, where two database entries do not share an item, and thus duplicate items need to be stored.

2) Input Preparation: Each input item in a field is first reduced in length by extracting K_f bits to be used as inputs in the proposed architecture. In this paper, a maximum length of 256 bits is dedicated for each search field before the extraction. K_f can be selected depending on the diversity of the items of field f, D_f, and the minimum required SFR. The extraction method depends on the data distribution of the field items and aims to reduce collisions between the extracted items.

If the items for each field have a uniform-random distribution, i.e. no particular similarity patterns can be discovered between the input patterns, the extracted bits can be selected randomly. The index (position) of the selected bits for extraction are consistent in all inputs. In databases, where a field receives inputs with high degrees of similarity, an SCN-based associative memory still generates the correct results, as shown in [10]. However, higher number of pattern similarities result in the generation of a larger number of output search results that include the desired ones.

One simple method is to reduce the similarities by using random vector-projection of the inputs, where an H-bit input pattern is first multiplied (using logical XOR operations) by a $[H \times N]$ random matrix ($N > H$) of binary elements to generate an alternative input pattern with fewer similar bits. The extracted items are then segmented into c_f parts.

3) SCN Mapping and Link Storage in the Co-processor: During mapping, nodes in clusters corresponding to the input segments are activated (set to '1') according to a specific rule describing the relationship between the segment value and the index of the node to be activated. Then the binary links are added between the activated nodes. To establish the binary links between the nodes, an input pattern with length of K_f-bits is divided into c_f segments such that:

$$w_{(f',i',j')}^{(f,i,j)} = \begin{cases} 1, & \text{if} \begin{cases} i \neq i' \\ \text{and } \exists m_n \in \{m_0 ... m_{M-1}\} \\ s_{(f,i,m_n)} = j \text{ and } s_{(f',i',m_n)} = j' \end{cases} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $w_{(f',i',j')}^{(f,i,j)}$ denotes the binary value of the connection from the j-th node of the i-th cluster in the f-th field to the j'-th node of the i'-th cluster of the f'-th field. The link values are stored in memory for later use during search. m_n is the n-th database entry among M total entries combining the information from all the input/output fields. It is segmented into f_{Max} fields and c_f segments in each field. The i-th segment ($0 \leq i \leq c_f - 1$) of the f-th field of m_n is denoted by $s_{(f,i,m_n)}$, and its length in bits is denoted by $\kappa_{(f,i)}$ for all M database entries such that:

$$\kappa_{(f,i)} = \log_2(l_{(f,i)}). \tag{2}$$

In (1), the mapping rule is that the value of segment $s_{(m_n,f,i)}$ is directly mapped to the index of the node to be activated. Therefore, there should exist $2^{\kappa_{(f,i)}}$ nodes to cover all possibilities.

The definition of density in [7] does not directly apply since the number of nodes in the clusters are not necessarily equal. Therefore, assuming that in M entries trained to the co-processor, all fields are used to define a clique, a new definition of density can be defined between two specific clusters as *Local Density*:

$$d_{(f,i,f',i')} = 1 - \left(1 - \frac{1}{l_{(f,i)} l_{(f',i')}}\right)^M, \tag{3}$$

where (3) is the ratio between the number of used binary links to that of the total possible connections between clusters i and i'. In [7], it is shown how increasing the value of density affects the message error rate in an associative memory. In the proposed system, increasing the value of local densities can increase the number of false-positive search results and thus increases SFR. Therefore, in order to reduce SFR as a means to improve the search quality, one can reduce the value of the local densities in two ways: (i) by reducing the total number of stored entries, (ii) and/or increasing the number of nodes in clusters with large diversities of the items. It is interesting to note that the local density is not directly proportional to the number of clusters. However, due to the limitation of silicon area, it is possible to increase the number of clusters in order to increase diversity (without changing the total number of nodes), but with the cost of increasing the density and hence the number of false positives.

B. Search Operation

Once the co-processor has been trained with the database entries, the search operation can be initiated. The search operation in the proposed system is partially illustrated in Fig. 1. The search process in more detail is a series of operations:

1) Data Collection: This operation is similar to that of the training process. The search queries are obtained from the search fields with available information.

2) Input Preparation: In this step, required data is generated for the input of the co-processor using a similar approach that was explained in the input preparation operation of the training process. The difference is in the status of the inputs as some parts of the inputs are missing such as the file IDs.

3) Search Operation in the Co-processor: Once the search operation is initiated after the prepared data inputs are presented to the co-processor, the CPU waits for the co-processor to perform the search operation and transfers its outputs to the CPU. The search process in the co-processor is performed by:

a) Local Decoding: The K_f-bit input, prepared in the input preparation operation, is divided into $\kappa_{(f,i)}$ $(0 \leq i \leq c_f - 1)$ segments. Each segment is mapped to activate a binary node in the i-th cluster of the f-th field. The mapping method is the same as the SCN mapping operation in the co-processor training. A field with missing input information, such as that of the File ID field, will have all of its nodes activated to permit all possible search results.

b) Global Decoding: Once the corresponding nodes to the input segments are activated in all the clusters, global decoding is performed using the stored links and the indices of the activated nodes. This process can be iterative, for which a node in any cluster remains or becomes activated in each iteration, if and only if $v^*_{(f,i,j)} =$ '1':

$$v^*_{(f,i,j)} = \left(\bigwedge_{\substack{i'=0 \\ (f',i')\neq(f,i)}}^{\psi-1} \bigvee_{j'=0}^{\gamma-1} w^{(f,i,j)}_{(f',i',j')} v_{(f',i',j')} \right) \bigwedge v_{(f,i,j)}, \tag{4}$$

where $v^*_{(f,i,j)}$ is the updated value of the j-th node of the i-th cluster in the f-th field after an iteration. ψ is equal to $\sum_{i'=0}^{c_f-1}$, and γ is equal to $l_{(f,i')}$. The indices of the activated nodes in the output field are transferred to the CPU after global decoding operation is completed in the co-processor. An iteration may thus reduce the number of falsely-activated nodes, and may thus reduce the delay to transfer the results.

4) External-Storage Content-Retrieval: In this operation, the CPU reads the co-processor outputs, and generates the required addresses for the external storage device using the co-processor outputs.

a) Formation of File IDs: The file IDs are formed from the co-processor outputs by concatenating the indices of the activated nodes in the file ID field, and then realizing all possible combinations that can be created using the index of an activated node in a cluster to all those of other activated nodes in other clusters in that field. For instance, if nodes 3 and 4 are activated in cluster 0, and node 5 is activated in cluster 1 of the file ID field consisting of only two clusters, the possible file IDs are: "3,5" and "4,5". Therefore, iterations can affect both the number of activated nodes, and the formation delay.

b) File Access: In the next step, the content of the corresponding files to the formed file IDs are retrieved from the external storage device. The addresses are either realized directly by the file IDs, or by mapping them to another set of numbers. In either of the two situations, if a generated address is not valid (eg. if the address is not in the scope of the addresses), the CPU drops it before accessing the external storage device.

5) Filtration and Display of Search Results: Since the formed file IDs may include false-positive results, the retrieved file contents may also include a few false-positive contents. The search results at this step can be further processed in either of the two ways: (i) The false positive contents are post-processed (filtered) in software by matching the search inputs with the information included in the file contents. Then the search results are displayed. (ii) or the search results are directly displayed without any filtration in form of a list to select from.

C. Updating

In order to update the co-processor with a new database entry, following scenarios are possible depending on the application:

1) Frequent Updates: If frequent updates are required in an application, only a single, but large cluster, can be dedicated for the output-field. This way, once an association is requested to removed, and updated with a new set of input items, then all the connections from the nodes of the output clusters can be removed. Large clusters increase the memory requirement to store the connections.

2) Moderately or Rarely-Frequent Updates: If less-frequent updates occur in an application, it is possible to temporarily mark *dirty* the associations that are no longer valid in software, and drop them if searched. This approach increases the network density, which results in increasing the number of false-positives that can be removed in a post-processing software program. However, no false-negative results are ever produced. Eventually, the co-processor is retrained with freshly updated associations after several updates take place.

III. Evaluation

In [7], the authors showed that density, number of clusters, and number of erased clusters determine the error performance of an SCN, that translates into the number of falsely activated nodes. The total number of clusters and the number of erased clusters are determined by the search-engine requirements such as how many fields are more frequently used during search, or the desired diversity of search items in each field. For example, if 1024 keywords are required, then either a single cluster consisting of 1024 nodes must be considered, or if constrained by the silicon-area, that single cluster can be divided into multiple clusters to reduce the number of nodes per cluster, and thus reduce the total memory requirement. On the other hand, increasing the number of clusters exacerbates the number of false-positives as the values of local densities are increased.

Selecting the value of local density between two fields that are frequently used during search (eg. Keyword, and file ID), can be used as a guide to create the intelligent guess for the number of nodes in a cluster. First, a target density, e.g. 60% is selected that determines the error rate. Then, according to (3), given a specific number of SCN messages, (eg. $M = 10,000$), and assuming equal number of nodes between the two clusters, the number of nodes in each cluster is rounded to 105. A set of parameters has been selected after various simulations, for the case study of the proposed system, as shown in Table I.

TABLE I
CASE STUDY PARAMETERS

Parameter	Value
Input Search Fields	3: Field 0 (Eg. Keywords), Field 1 (Eg. Year), Field 2 (Eg. Name)
Output Fields	1: File ID (Field 3)
f_{Max}	4
Clusters/Field (c_f)	$c_0 = 2, c_1 = 1, c_2 = 2, c_3 = 3$
Clusters ($\sum_{f=0}^{f_{\text{Max}}-1} c_f$)	8
Nodes/Cluster	$l_{(0,0)} = l_{(0,1)} = 128$ $l_{(1,0)} = 32$ $l_{(2,0)} = 16, l_{(2,1)} = 128$ $l_{(3,0)} = l_{(3,1)} = l_{(3,2)} = 128$
Iterations	1
Unique file IDs	2,000
Field 0 items	16,384
Field 1 items	32
Field 2 items	2,048
Entries (M)	10,000
Field 0 items/File ID	5 (Avg.)
Search Item length (bits)	256
Segmented Item Length (bits)	$\kappa_{(0,0)} = \kappa_{(0,1)} = 7$ $\kappa_{(1,0)} = 5$ $\kappa_{(2,0)} = 4, \kappa_{(2,1)} = 7$ $\kappa_{(3,0)} = \kappa_{(3,1)} = \kappa_{(3,2)} = 7$

The simulation results in this section are presented prior to the optional filtration operation in software as discussed step 5 in Section II-B.

As the proposed co-processor stores the associations within the database entries, the number of the used connections is increased, causing the local density values to also increase. Fig. 3 depicts the relationship between the number of stored database entries (i.e. uniquely associated file IDs), and the number of generated search results in the proposed system for various search scenarios. The rate of change in the number of generated search results (R_g) to the total number of the associated file IDs (N_{ID}) is equal to:

$$SFR = \frac{dR_g}{dN_{ID}}. \tag{5}$$

The generated co-processor signals are the indices of the activated nodes in the clusters that, when processed, create the file IDs. The generated file IDs include the desired file ID or IDs if they exist along with a few detectable false-positive results. The number of generated search results in the proposed system depends on the search scenario, such as how many search fields with valid search information have been presented. In Fig. 3, four search scenarios are presented. The advantage of the proposed system in focusing the search scope is realized, as shown in Fig. 3, when large number of search fields with presented inputs are required to identify desired search results. Fig. 4 depicts the effect

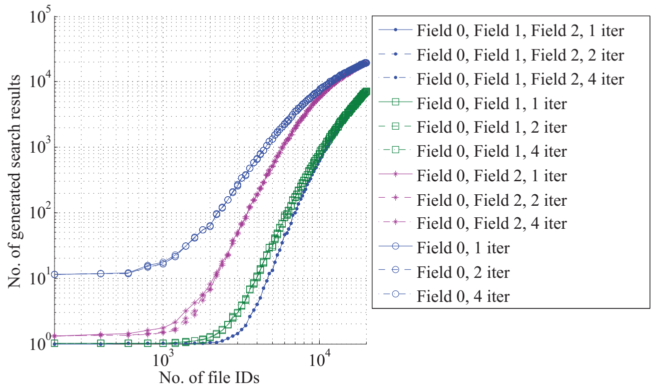

Fig. 3. Relationship between the number of search results, generated by L-SCAN, and the total number of associated database entries in the form of file IDs for various search scenarios.

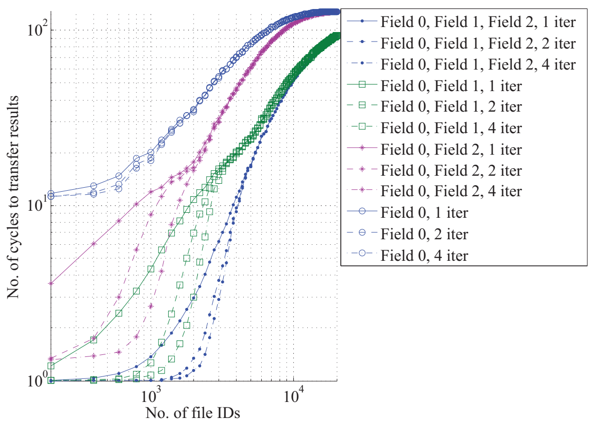

Fig. 4. Relationship between the number of search results, generated by L-SCAN, and the number of required clock cycles to transfer the node indices for various search scenarios.

of iterations on the number of clock cycles to transfer the search results from the co-processor to the CPU for various search scenarios. The number of clock cycles are simulated by evaluating the maximum number of activated nodes among all output clusters, and assuming that each activated node requires a clock cycle to transfer its index. It is assumed that the indices of the activated nodes in a cluster are transferred serially for each cluster but in parallel with other clusters. Considering the observations from Fig. 3 and Fig. 4, it can be concluded that the search scope after using the co-processor can be significantly focused. Only a single matched result is determined if the total number of associated file IDs (stored database entries) falls into a region, where a small rate of changes in the number of search results can be observed (Eg. when the number of file IDs are $< 2,000$). This rate of change is SFR, that is a parameter which can be defined in the design requirement for a specific database to control the number of false-positive results.

IV. COMPARISON

In the co-processor, the total number of required memory bits, P, is given by:

$$P = \sum_{f=0}^{f_{\text{Max}}-1} \sum_{i=0}^{c_f-1} 2^{\kappa_{(f,i)}} \cdot \sum_{f'=0}^{f_{\text{Max}}-1} \sum_{\substack{i'=0 \\ i' \neq i}}^{c_f-1} 2^{\kappa_{(f',i')}}, \tag{6}$$

TABLE II
FEATURE COMPARISON

Parameter	L-SCAN (This work)	CPU-CDS	CAM-CDS
Required Physical Memory (Mb)	0.54	7.32	7.32
Min./Average Transfer Cycles	3/16.69	-	5/158.75
Min./Average Search Cycles	2/2	240 / 3.4×10^6	2/2
Min./Average Search + Transfer Cycles	5/18.69	240 / 3.4×10^6	7/160.75

TABLE III
SEARCH RESULT TRANSFER CYCLES

Field(s) with Input	L-SCAN (This work)	CAM-CDS
0	35.27	5
0,1	10.82	312.5
0,2	17.67	5
0,1,2	2.98	312.5

whereas a CAM-based counterpart would require $f_{\text{Max}} \times Q \times M$ CAM cells, where Q is the required length of a search item in the database. This memory requirement does not take into account the number of memory bits in the SRAM array that is attached to each CAM array. In a CAM-based counterpart, the SRAM array stores the file IDs. Table II shows the comparison of the memory requirement and search delays in clock cycles, for the proposed system for selected search scenarios, a Texas Instrument MSP430 embedded processor, and CAM-based search engines. The processor has 16-bit instruction, 32-bit data and 64-KB cache memory performing an exhaustive context-driven search in software with related search parameters in Table I. L-SCAN requires 13.6× fewer memory-bits compared to those of CAM-CDS. The number of co-processor search cycles are according to that in [9]. The search cycles for CAM are for receiving an input and then registering the generated matched results.

As shown in Table III, the average delay to transfer the generated search results, in clock cycles, can be calculated using Fig. 4 in terms of the clock cycles, and considering one transfer per cycle. This delay is, for an average search scenario, 16.69 cycles, which is 9.5× smaller than that of a CAM-based architecture transferring independent matched results. Compared to a CAM, the proposed system has in total 8.6× reduced number of clock cycles on average to search and transfer results. Since it is required to transfer the results serially after the search is completed, the co-processor, and CAM architectures cannot be easily pipelined. After the proposed co-processor computes the search results, the state of memory units can be changed to standby while the results are transferred to CPU from the output registers.

V. CONCLUSION

In this paper, a multiple-field search engine was presented employing the concept of a new family of associative memories known as sparse clustered networks. In the proposed system, the memory requirement to construct a search engine is reduced by 13.6× compared to a classical hardware-based search architecture based on content-addressable-memories (CAMs). Furthermore, for comparison purposes, the search delay of a CPU running a classical serial search algorithm was also simulated, and its number of clock cycles were measured on average to complete several search operations with different scenarios. Due to its parallel computation structure, the proposed system achieves 8.6× reduced number of clock cycles on average in performing a search operation compared to CAMs, and five orders of magnitude reduced number of clock cycles compared to a CPU-based counterpart running an exhaustive search in software.

We expect that the large reduction in the number of clock-cycles results in large delay improvements when L-SCAN is implemented in ASIC following the results reported in [9]. Future work includes hardware-implementation and fabrication of the proposed co-processor as an energy-efficient search engine.

REFERENCES

[1] K. Taha and R. Elmasri, "XCDSearch: An XML context-driven search engine," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 12, pp. 1781–1796, 2010.

[2] I. Hayashi, T. Amano, N. Watanabe, Y. Yano, Y. Kuroda, M. Shirata, K. Dosaka, K. Nii, H. Noda, and H. Kawai, "A 250-mhz 18-Mb full ternary CAM with low-voltage matchline sensing scheme in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 11, pp. 2671–2680, 2013.

[3] I. Arsovski, T. Hebig, D. Dobson, and R. Wistort, "A 32 nm 0.58-fJ/Bit/Search 1-GHz ternary content addressable memory compiler using silicon-aware early-predict late-correct sensing with embedded deep-trench capacitor noise mitigation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 4, pp. 932–939, 2013.

[4] Y. Chang, F. Kuo, H. Guo, and C. Su, "Layeredtrees: Most specific prefix based pipelined design for on-chip ip address lookups," *IEEE Transactions on Computers*, 2013.

[5] J. Hasan, S. Cadambi, V. Jakkula, and S. Chakradhar, "Chisel: A storage-efficient, collision-free hash-based network processing architecture," in *Computer Architecture, 2006. ISCA '06. 33rd International Symposium on*, 2006, pp. 203–215.

[6] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor, "Longest prefix matching using bloom filters," *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 397–409, 2006.

[7] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *IEEE Transactions on Neural Networks*, vol. 22, no. 7, pp. 1087–1096, Jul. 2011.

[8] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross, "Architecture and implementation of an associative memory using sparse clustered networks," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Seoul, Korea, May 2012, pp. 2901–2904.

[9] ——, "Reduced-complexity binary-weight-coded associative memories," in *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2013, pp. 2523–2527.

[10] H. Jarollahi, V. Gripon, N. Onizawa, and W. J. Gross, "A low-power content-addressable memory based on clustered-sparse networks," in *Proceedings of the 24th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Jun. 2013, pp. 305–308.