HAPI: Hardware Assisted Pruned Index for Consumer Electronics

S. Kagan Agun and Ophir Frieder, *Fellow*, IEEE

Abstract — *We describe HAPI, a novel Hardware Assisted Pruned Index (HAPI) component. HAPI is a content indexing device based on a modified inverted index structure. HAPI accommodates patterns of different lengths and supports a varied posting list versus term count feature sustaining high reusability and efficiency. The developed component can be used either as an internal slave component or as an external co-processor. HAPI is efficient in resource demands since the component controllers take only a minimal percentage of the target device space leaving the majority of the space to term and posting entries*[1].

Index Terms — Inverted index, information retrieval, accelerator, hardware support.

I. Introduction

Inverted index file structures support the efficient searching of documents. Static index pruning [5] reduces the number of posting entries stored in the index while still providing comparable accuracy in query processing. By storing the posting entries of only those documents for which a given term appears in frequently, the size of the posting list is reduced, improving runtime performance. Implementing such an approach in hardware, e.g., our Hardware Assisted Pruned Index (HAPI) component, aids in high-speed document searching of relatively small collections found on consumer devices such as a PDA or intelligent cell phone. The HAPI system was initially introduced as the HAT component [1], but the new name was recently adopted due to commercialization trademark considerations.

Research in application acceleration of consumer electronic devices via hardware support is common. For example, in [11], a multi-resolution block-matching algorithm was proposed for MPEG-2 video encoding with a large search capability. A flexible Block Matching Unit was designed to accelerate content-based motion estimation for real-time MPEG-4 video coding in [9], and in [10], a hardware accelerator to reduce distortion optimization in the new H.264 video coding standard is discussed.

Indexing algorithms for consumer electronics devices likewise exist. In [6], an index-coding algorithm for image vector quantization that uses dynamic tree coding is described. Fast content-based image browsing and retrieval in a database based on the rosette pattern is also proposed for consumer electronics in [8]. Using a neural network extraction technique together with a Discrete Cosine Transform (DCT) domain watermark allows the indexing of the image entirely in [3]. That technique allows for the real-time embedding of indexing data in popular image (JPEG) and video (MPEG) formats.

Regarding our domain of interest, accelerated text search, few hardware-supported accelerators exist. A Field Programmable Gate Array search accelerator that packs the information into pre-formatted text vectors and compares with a query for similarity is introduced in [13], while a programmable approximate string matching processor implementing a parallel similarity pattern matching is found in [4]. A recent effort presented in [2] is a reconfigurable memory based string-matching accelerator for intrusion detection. These efforts demonstrate the research community's interest in hardware acceleration as a means to support efficient search. In terms of commercial interest, the TextFinder FDF4 [12] and the Fast Search Chip [7] are both recent search accelerator products. The HAPI component described herein is a search accelerator custom suited for small text databases found in personal consumer devices.

Rather than focusing on pattern matching or similarity measure computation, HAPI is a component that assists in the management of the inverted index. Mapping the highly accessed inverted index software structure onto a re-configurable chip reduces the processing time associated with index access and simplifies the maintenance. HAPI also achieves high performance through hardware parallelism. The precision and feature-oriented design of the HAPI system supports component instantiation in multiple semiconductor process technologies so as to potentially increase the longevity of the approach.

The remainder of this paper is organized as follows. In Section 2, we review the pruned inverted index algorithm. In Section 3, we present the HAPI architecture, while in Section 4 we present our prototype VHDL implementation of a HAPI component. A comparison of the performance of a VHDL specification of HAPI versus that of a software pruned index module is presented in Section 5. We conclude the paper in Section 6.

II. Pruned Inverted Index

Static index pruning [5] bounds the posting list by removing those posting entries corresponding to documents

[1] S. K. Agun was with the Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois 60616 USA. He is now with Multivision, Inc., a private consulting company, Naperville, Illinois 60563 USA (e-mail: agunsal@iit.edu).

O. Frieder is with the Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois 60616 USA (e-mail: ophir@ir.iit.edu).

for which the term has lower significance. That is, for each term in the index, up to a maximum number of posting entries is maintained. The retained entries are stored in sorted order and correspond to documents where the given term appears the greatest number of times or has the greatest relevance.

A pruned inverted index approach reduces the storage size, I/O, and processing times without noticeably degrading retrieval accuracy. Besides these advantages, the pruned index approach bounds the posting list size and "regularizes" the storage, simplifying the design of a corresponding hardware accelerator.

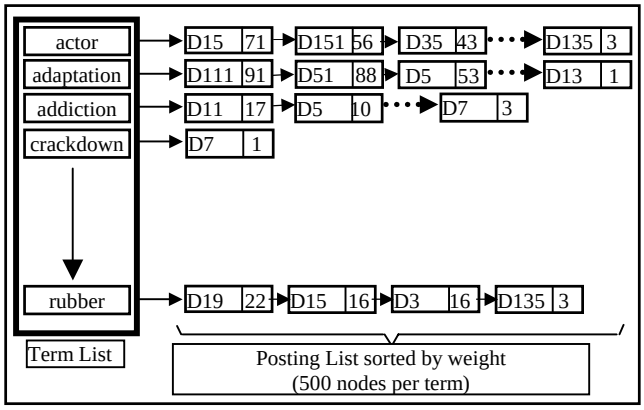


Fig. 1. Logical pruned inverted index structure.

An illustrative example is shown in Figure 1. As shown, for each term stored in the index, corresponding posting entries, up to a maximum number, say 500, are maintained. These posting entries are stored in a sorted order according to the term's frequency of appearance in the given document. Note that although logically pointers are shown, in practice, contiguous blocks of fixed length are allocated to each posting array. This eliminates the need to store the "pointer" addresses, reducing the memory requirements of the posting list as shown in Figure 2.

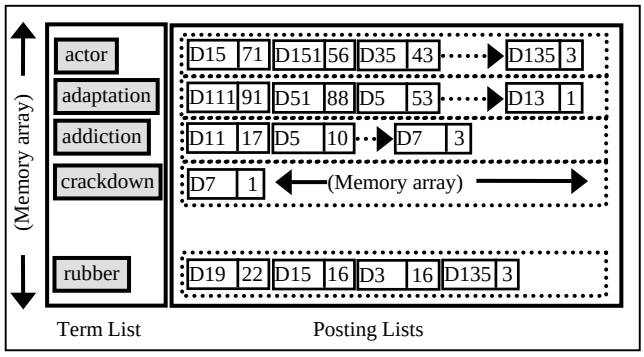


Fig. 2. Physical pruned inverted index structure.

III. HAPI ARCHITECTURE

Each HAPI component design is parameterized so that it can be instantiated to any precision and promotes a variable maximum number of posting entries per term as well as a varying maximum term length. The number of bits is the most common precision in the design (bit-sliced precision), but a component can be divided into identical slices that can be reused to form any size of the same component known as component-sliced precision. The HDL generic statements can instantiate both small components (e.g., 1-bit pattern-matcher) and n-bit pattern-matcher systems in an iterative fashion. Such an approach results in a reusable design. A configuration file is used to keep the precision parameters that instantiate a particular HAPI structure.

Flexibility is critical to maintaining longevity of the approach. Thus, for example, the memory used to construct the term and posting list storage offer both generic memories suitable for any platform or custom memories for a specific target device. It is convenient to implement related features in one design so that this design can be reused, as different features are required. Using the right design component also optimizes the performance through synthesis tools. While a single generic memory of HAPI takes 120 cells at a maximum clock speed of 100 MHz, Xilinx Virtex FPGA internal memory blocks based on select RAM take 55 cells at 300 MHz.

HAPI operates as either a co-processor or as an embedded component. When HAPI is used as an embedded component, it is associated with a general-purpose processor on a single chip as a logical processing unit in the same manner as an Arithmetic Logic Unit (ALU). The HAPI external interface (Figure 3) includes an address bus (Address) and a bi-directional data bus (Data), both can be instantiated to any width. The output lines RW, Enable, and Ready are used to handle the memory access. HAPI also includes asynchronous Reset and Halt signals where reset initializes and halt terminates the operation of the HAPI component. The clock signal is the system clock. The Error output indicates an unrecoverable error state. The term Found, Lstatus (Left Status), and Rstatus (Right Status) signals are used to stack up to 16 HAPI components in an array (Figure 4). These control signals are used to pass the status information including term full signal and term found signal between HAPI units.

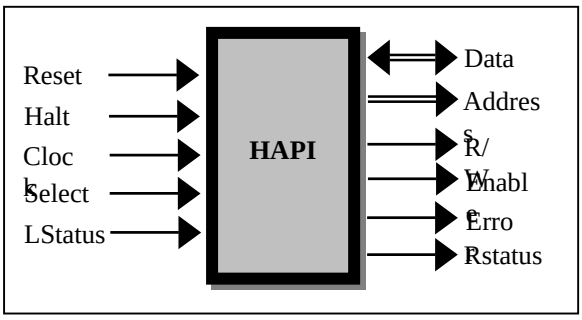


Fig. 3. The HAPI External Interface.

The HAPI architecture includes two main component types: term matching units and array of posting units. Each posting unit handles one posting list which stores the top M

documents in sorted order. These units are connected to the internal data bus and controller signals. These units are highly parallel and support master - slave (client - server) operations. While a query is retrieving data from a posting list unit, another unit executes sorted list update operations.

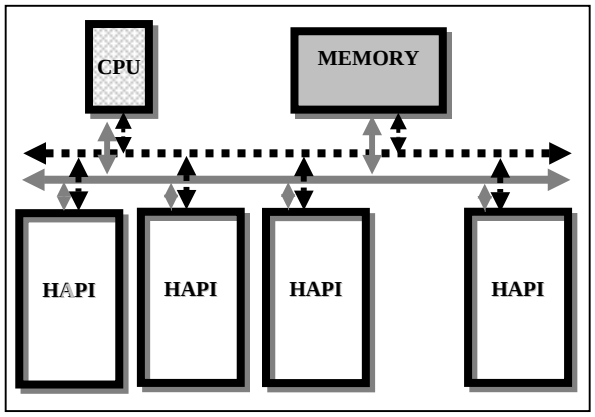


Fig. 4. A Possible HAPI Bank System.

A third component, the HAPI Controller is the master controller that distributes the work and manages the communication with the main processor. The high performance bus interface can be a PCI bus or any other bus available on the market for re-configurable computing. This controller includes a FIFO queue to fetch main processor data and instructions. A DMA unit is considered to provide efficient memory access for the co-processor approach. The entire HAPI Architecture is shown in Figure 5.

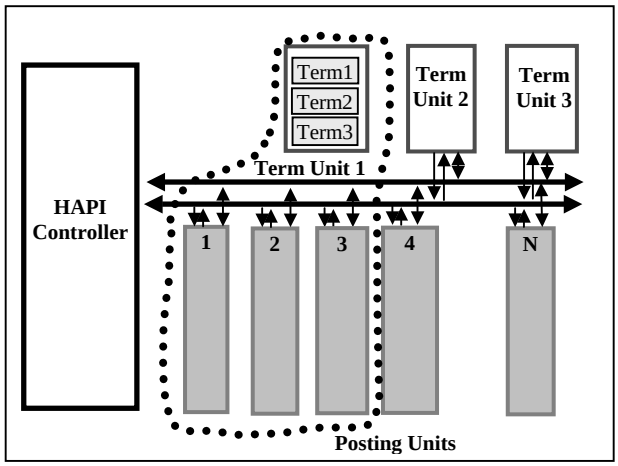


Fig. 5. HAPI Architecture.

A. Term Unit

Rather than a character-by-character comparison, in the developed HAPI Term Unit, term matching is a single atomic action. Matching is achieved by simultaneously comparing all the characters of the term with the characters of the pattern. N terms can be scanned in N clock cycles; that is, an average of one term per cycle. The data retrieval response time increases when the size of the term list increases. HAPI exploits the parallelism in term-matching to reduce the document search time. This provides faster execution since pattern search is done in parallel. The multiple, term-matching units that execute in parallel are shown in Figure 6.

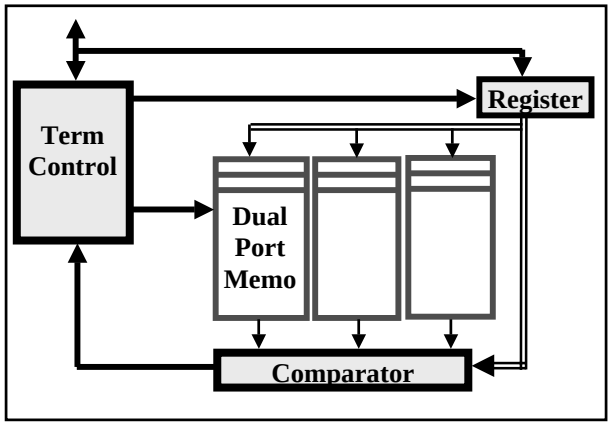


Fig. 6. Term Unit Architecture.

Each term unit stores one section of the term list. Term scanning can be achieved in parallel on all the units, but a new term is inserted (added) into the last unit, if previous units confirm that the term does not exist in their portion (list). A found signal is sent to other units to stop the execution of pattern searching. The result of the search, an index of the term, is returned to the HAPI Controller.

Term lengths between 4 and 12 characters are supported. The HAPI term matching unit provides two approaches for word comparison, namely, either a single 12 character (96-bit comparator) or three 32-bit comparators are available insuring consistency with the 32-bit data path. Each slice has its own memory and comparator. Unused characters in a word are set to zero. A mask signal is provided to compare relevant slices of the word. The parameterized precision-oriented design of the term-matching unit supports a varying number of components and dimensions.

B. Posting List Units

The posting list unit is very similar to the term matching structure. In our implementation, eight bits are used for the term frequency count that supports up to 255 occurrences. (Actually, eight bits can accommodate 256 occurrences since a case with no occurrences will not result in a posting entry; yet for simplicity, we allow only up to 255 occurrences.) Given our 32-bit word instantiation, this leaves 24 bits for the document identifier. Clearly, a collection comprising of more than 16 million documents requires different instantiation parameters. However, at least for current instantiations of HAPI, we do not see a consumer device application requiring the storage of more than 16 million documents. The posting list block architecture is illustrated in Figure 7.

There is one posting list per term. Each posting unit forms a one-dimensional array that is connected to an internal data and control bus and supports master - slave operations. Posting list

sub-processors execute list update and posting list retrieval operations in parallel.

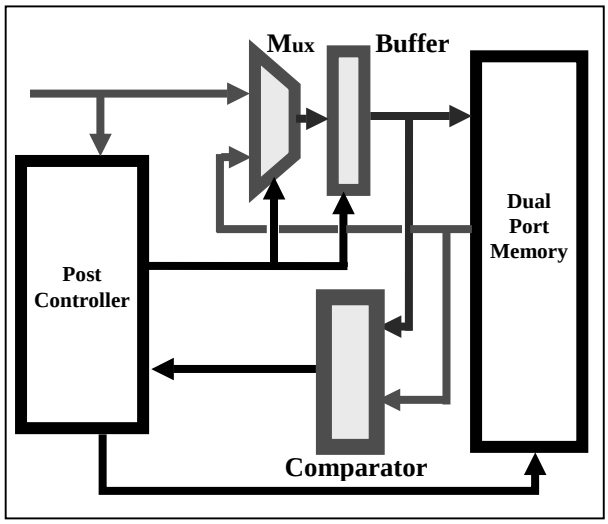


Fig. 7. HAPI Posting List Block Architecture.

The updating of a posting list involves inserting a single document into a list in the proper location, potentially requiring the removal of the least relevant posting entry. It is clearly desirable to accomplish this within a single scan. Thus, the HAPI posting list memory component was designed to read the list at the rising edge of the clock and to write the entry at the falling edge, providing fast self-document list update.

C. HAPI Controller

The HAPI master controller accepts requests from the main processor and executes the operation by synchronizing its subordinate components, namely, the term matching and posting-list processors. A FIFO queue is used to store sequential user or main processor queries. A Direct Memory Access (DMA) unit is used to direct memory transfer between the HAPI and external system memory. An internal register file stores the document identifier number for the posting list insert or update operation.

Two control processes run in parallel to handle the HAPI operations. One process handles the incoming instructions either storing them into the FIFO queue or starting the term related operation of the instructions on the term unit. The immediate status of the term unit and internal bus determines whether the instruction is immediately executed or queued.

IV. IMPLEMENTATION

We used the Altera Leonardo Spectrum, a suite of high-level design tools for hardware synthesis, to design HAPI. In Table 1, we illustrate the logic cell usage and speed requirement of the various HAPI components for the target device Cyclone EP1C20T400C, the smallest device in this category. It is worth noting that the control logic of the components requires only a small amount of cell resources. In addition, embedding internal chip memory as a cache into the design increases the performance. Later we used Altera Quartus 4.1 web edition and Xilinx Foundation series ISA 6.2i synthesis tools to compare the results.

FPGA chip memory varies from device to device. Different hardware synthesis tools result in different memory sizes and speeds. Thus, the generic memory design used in the HAPI system can therefore support varying memory sizes and speeds depending on the particular synthesis tool used. For example, a logic block can be used as memory rather than in-chip memory. Using feature oriented design methodologies, device specific memory components were designed. This approach also supports adding new memory components to the design for future FPGA.

Dual port memory is used in the HAPI components. By initiating the target specific memory feature described above, the HAPI memory component can be mapped into device memory blocks. For example, a HAPI memory component of 32 bit data width and 256 length takes only 8 Kbit (32 x 256) memory space in the target device (Table 1) and runs at 356.2 MHz. The HAPI dual port memory is also the main component of TermUnit and PostingUnit. Controllers of these components (TermUnit and PostingUnit) with 32 x 256 memory unit use only 93 and 178 logic cells, respectively. These totals are less than 1% of the target device.

We instantiated a 100 term HAPI unit with a single 32 bit wide term unit and 32 bit wide 64-entry maximum posting list units as a prototype system. The prototype has a 32-bit data width. Scalable and flexible term and posting list components optimized the performance of the HAPI system. The configured HAPI component runs at 79.7 MHz for the selected target device (Table 1). It should be noted that only academic free-ware tools were used in the design, and hence, the available optimization for both chip speed and space were severely limited.

TABLE I
RESOURCES CLAIMED BY HAPI COMPONENTS.

Component	LogicCell (LC)	Memory (bits)	Frequency (MHz)
Generic DualPort Memory (256x32)	--	8192 (3.13%)	356.2
TermUnit (256x32)	178 (0.89%)	8192 (3.13%)	146.1
Posting Unit (256x32)	93 (0.46%)	8192 (3.13%)	110.8
HAPI (100 terms)	8340 (41.58%)	212992 (81.25%)	79.7

(*Cyclone EP1C20T400C (20,060 LC, 294,912 Memory bits))

A project configuration file is used to instantiate the configuration of the various HAPI components (Figure 8). Clearly, configuring the HAPI system with multiple term units exploits parallelism but at a cost of limiting further the number of supported posting entries per term. Clearly chip space is fixed; hence the greater number of terms, the lower is the maximum number of posting entries per term. Similarly, the maximum character length of the term affects chip space. Regardless, all of these parameters can be instantiated to any

size or number.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

PACKAGE HAPIconfig IS

type syntools is (Synopsis,Maxplus2,LeonardoSpectrum);
type targetdevices is (GEN, GENBUF, LEO, VIRTEX,
XILINX_SELECT, FLEX10K, MAX7000);

CONSTANT TARGET_DEVICE : targetdevices := VIRTEX;
CONSTANT TARGET_TOOL : syntools := LeonardoSpectrum;
..
CONSTANT TERM_address_width : POSITIVE := 7;
CONSTANT TERM_word_width : POSITIVE := 32;
CONSTANT TERM_slice : POSITIVE := 3;
CONSTANT Term_Slice_Address_Width : POSITIVE := 2;
..
CONSTANT TotalTERMUnit : POSITIVE := 1;
CONSTANT TotalTERMUnitAddressWidth : POSITIVE := 8;
..
CONSTANT POSTLIST_address_width : POSITIVE := 5;
CONSTANT POSTLIST_DocID_width : POSITIVE := 8;
CONSTANT POSTLIST_DocFrequency_width : POSITIVE := 16;
```

Fig.8. HAPI Configuration File.

In Tables 2 and 3, various configurations of the HAPI design were evaluated for FPGA and CPLD target devices. Due to availability of target devices and their sizes, all the desired configurations of HAPI could not be tested on all the devices. A HAPI processor with a single term unit and multiple term units running in parallel were also evaluated. Term (16 x 32) unit represents a term unit with 16 terms and each term is 32-bit word length. On the other hand Post (64 x 32) unit consists of 64 posting entries, and each posting entry has a 32-bit length. By default a 32-bit posting list is divided into a 24-bit term number and an 8-bit frequency value. A Term (16 x 32) Unit requires 16 post units of configured size. HAPI with 4 Terms (16 x 32) Posts (64 x 32) consists of 4 Term units and runs at 52.88 MHz while HAPI with 1 Terms (64 x 32) Posts (64 x 32) has the same size term list but runs at 37.52 MHz on an Altera Stratix II device (Table 3). HAPI with 16 Terms (16 x 32) Posts (64 x 32) runs at 49.29 MHz and HAPI with 1 Term (256 x 32) Posts (64 x 32) runs at 18.84 MHz respectively on a Xilinx Virtex II device (Table 3).

TABLE II
RESOURCES CLAIMED BY VARIOUS HAPI COMPONENTS FOR CPLD TARGET DEVICE

Components	Logic Cells	Memory Bits	MHz
1 Term (16x32) Posts(64x32)	2279(12%)	25472(6%)	59.84
1 Term (32x32) Posts(64x32)	4287(23%)	50560(10%)	43.45
1 Term (64x32) Posts(64x32)	8181(46%)	100736(24%)	37.78
1 Term (128x32) Posts(64x32)	--	--	--
1 Term (256x32) Posts(64x32)	--	--	--
2 Term (16x32) Posts(64x32)	4307(34%)	50560(12%)	55.23
4 Term (16x32) Posts(64x32)	8411(67%)	100736(24%)	52.88
8 Term (16x32) Posts(64x32)	--	--	--
16 Term (16x32) Posts(64x32)	--	--	--

TABLE III
RESOURCES CLAIMED BY VARIOUS HAPI COMPONENTS FOR FPGA TARGET DEVICE

Components	LUT	Memory	MHz
1 Term (16x32) Posts(64x32)	2879(3%)	18	53.58
1 Term (32x32) Posts(64x32)	5428(6%)	37	43.37
1 Term (64x32) Posts(64x32)	10512(11%)	66	33.29
1 Term (128x32) Posts(64x32)	20635(19%)	130	23.08
1 Term (256x32) Posts(64x32)	40997(46%)	258	18.84
2 Term (16x32) Posts(64x32)	5508(6%)	35	50.30
4 Term (16x32) Posts(64x32)	10790(12%)	69	50.08
8 Term (16x32) Posts(64x32)	21302(24%)	137	49.64
16 Term (16x32) Posts(64x32)	42451(48%)	279	49.29

HAPI provides a Reduced Instruction Set Computer (RISC) like instruction set. Five basic instructions are defined to perform document update and retrieval. The HAPI instruction set layout is illustrated in Figure 9 with the HAPI instructions listed in Table 4. Posting list identifiers are stored in the register file by calling the StoreDoc instruction before calling the UpdateTerm or InserTermDoc instructions. The StoreDoc instruction is an independent operation, which can be called at any time while HAPI is running. Term data follows the HAPI instruction. The OpX function value determines the number of the term slices that are the width of the data. The HAPI controller reads the term from the data bus and either directs it to TermUnit or stores it in FIFO if TermUnit is busy.

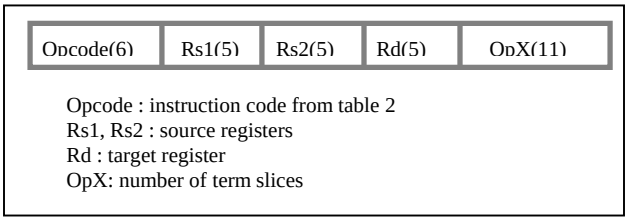


Fig. 9. HAPI R – Type instruction layout.

TABLE IV
HAPI R–TYPE INSTRUCTION SET

OpCodes	Code	Description
StoreDoc	B"010001"	Store document id into the registers.
UpdateTerm	B"000101"	Searches or adds the term, adds the posting list.
InsertTerm	B"001001"	Inserts the term without searching.
RetrieveTerm	B"000001"	Searches the term, returns posting list
InsertTermDoc	B"011001"	Inserts the term without searching, adds the posting list.

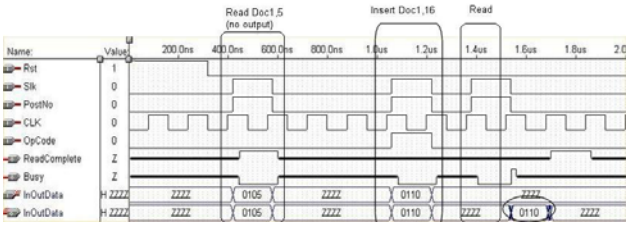
The UpdateTerm instruction searches the term in the term list. If it does not find the term it adds it to the term list and inserts the posting as the first entry. However, if it finds the term, the posting entry is inserted into the right location based on the frequency of the updated term. On the other hand InsertTerm and InsertTermDoc are used to upload pre-listed terms into term unit. InsertTermDoc also inserts the document identifier and the frequency into the posting unit. The

RetrieveTerm searches the term in the TermUnit and returns the corresponding entries from Posting Unit.

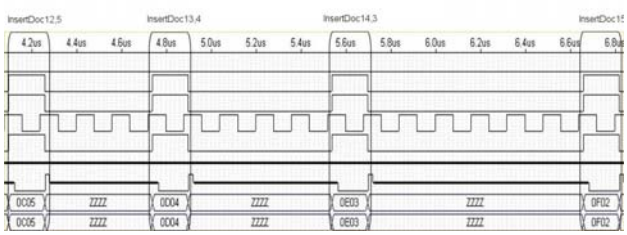
Figure 10 shows the waveform results of the posting list operations. Additional I/O signals were added to the design to demonstrate the internal operation of the posting list unit of HAPI processor. We explain those signals respectively in the next paragraphs. The posting list is configured to show clear waveform results. The size of the document identifier field and the frequency are set to one byte length. Input and output data are represented in hexadecimal form in the waveform. Documents with different frequency are inserted randomly. The last read instruction shows the data in sorted order.

The Rst signal resets the HAPI component by setting all the index references and counters to zeros. HAPI processes / reads the instruction when the Select signal is set to one. The Clk signal represents clock signal. Postno, Busy, ReadComplete, OpCode, and InOutData are internal signals to demonstrate how the HAPI system works. The InOutData is the internal data bus that connects the HAPI controller, the posting units, and the term units. The Read Doc1, 5 instruction is divided into 2 sections; opcode "Read" and the data "0105". "01" is the document and "05" the frequency, respectively. The ReadComplete signal is a response signal from the posting units. It indicates that the last posting list data were put into databus in this case InOutData.

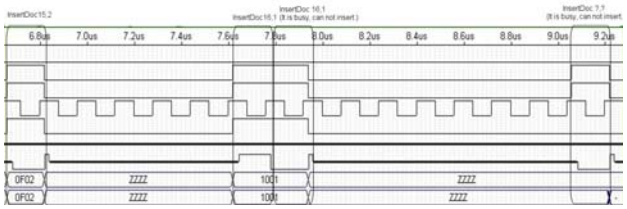
In waveform part 1, first we reset the posting unit, and then execute the read instruction. The response of the unit was complete because there are no data in the posting list unit. The second instruction is insert document 1 with frequency number of 16. The third instruction is read. In the first clock cycle of the read instruction, the posting list unit gets the instructions. In the second clock cycle, it puts the first entry on the data bus (see InOutData output port), and in the third clock cycle, the posting list unit sets the ReadComplete signal. We inserted several documents into the posting list. In part 3 of the waveform, while the posting list is updating its entries for the insert instructions, we send another insert instruction, and the posting list busy signal is set to '0' (busy). The last instruction shown in part 4 is the read instruction, and the results of the posting list appear on the output port of the InOutData.



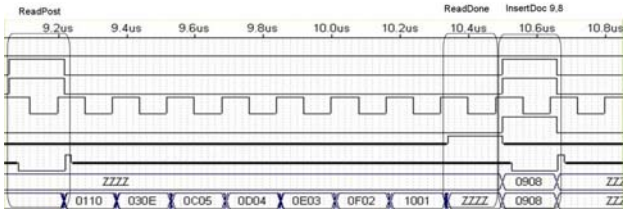
(a)



(b)



(c)



(d)

Fig. 10. Waveform of Posting List reads and updates.

V. EVALUATION

We compare the HAPI component against a software implementation of a pruned inverted index file approach using 500 documents, a total of 378989 words. These documents were chosen among the various web sites. In Table 5, we show the source of these web pages. The web pages were indexed and saved for both a software implementation and a HAPI emulation. A total of 182736 words were indexed. Both the software pruned indexing and HAPI emulation program were written using c++ in the Visual Studio .NET 2003 development platform and compiled for high performance to demonstrate a real world software approach for pruned indexing. The program implements a software-implemented pruned inverted index file algorithm and measures the computation time for the requested operations. The same program also executes the HAPI specification and counts clock ticks to emulate the hardware.

TABLE V
SOURCE OF THE DOCUMENTS

Web Sites	Links
www.abcnews.com	**112**
www.bbc.co.uk	78
www.cbsnews.com	65
www.cnn.com	47
www.foxnews.com	63
www.msnbc.com	40
www.dailynews.com	16
www.chicagosuntimes.com	43
Others	7

To eliminate unnecessary graphics and advertisement information, printable versions of the pages were scanned to get the word and term counts. In Figure 11, we present the distribution of the total word count results. 65.3% of the words of the 500 web sites were unique. Pronouns, conjunction, and auxiliary verbs were counted as known words. Numbers were counted separately. These two types of words represented 32.1% and 2.5% of the total words, respectively. In Figure 12, we show the processed term distribution: the unique terms (85.6%), known terms (10.8%), and numeric terms (3.6%).

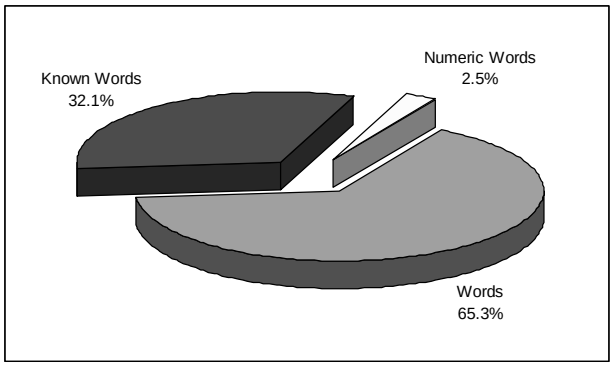


Fig. 11. Total word distribution of the documents.

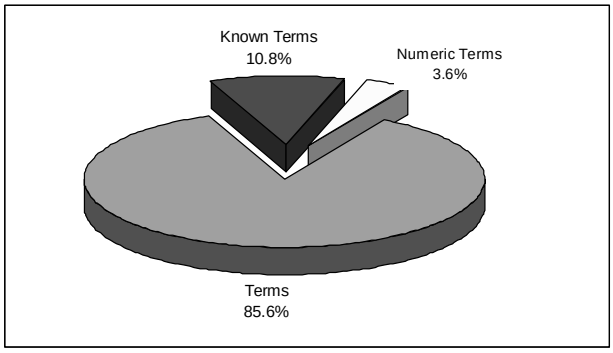


Fig. 12. Distribution of the processed terms in the documents.

TABLE VI
WORD AND TERM COUNTS

	Unique	Known	Numeric
Total Words	247551	121776	9662
Processed Terms	156489	19664	6584
Result Terms	18542	54	533

The HAPI component demonstrates high performance for the 500 web page document collection. In Figure 13, we show the average number of ticks in hardware (including term units of size 512 and posting units of size 256) and the software implementation. While the software implementation depends on the computer hardware and the operating system, we ran the application several times with the same data and calculated the average number of ticks used to insert every term. A total of 182736 terms (Table 6) were used to evaluate the various operations. It is worth noting that the software implementation needs more time when the term list grows. The measured computation times of the software and hardware implementations do not include the input-output time.

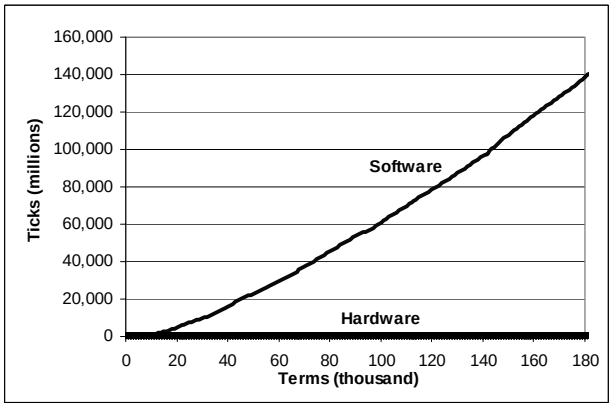


Fig 13. Software and hardware ticks.

TABLE VII
RESULTS OF HARDWARE AND SOFTWARE EMULATIONS IN TICKS

Number of Words	Hardware Ticks*	Software Ticks*
40,000	12	15,825
80,000	23	45,524
120,000	34	78,511
160,000	44	118,110
180,000	49	138,588

* Millions of ticks, hardware system includes term units of size 512 and posting units of size 256

In Table 7, we show the average number of ticks in hardware (including term units of size 512 and posting units of size 256) and the software implementation. We instrumented the software inverted index algorithm on a 2.8 GHz Pentium IV machine running Windows XP. A two-plus order of magnitude performance improvement in terms of execution time over the software approach was noted.

Finally, we compared various configurations of a HAPI component. We chose term unit sizes of 2048, 1024, 512, 256, and 128 terms. Smaller term unit sizes result in a higher control logic cost. In Figure 14, we show the average number of ticks measured in the various HAPI systems based on varying term unit sizes. The fastest one term processing speed of the software inverted index algorithm measured on 2.8 GHz Pentium IV machine running Windows XP is 773955 ticks. If we choose a 512 Term HAPI, the overall speed-up factor is 2817 assuming that the HAPI component can run at equal speeds. Even at a speed of only 70 MHz (slower than even our implementation), HAPI still yields a speed-up factor of over 70.

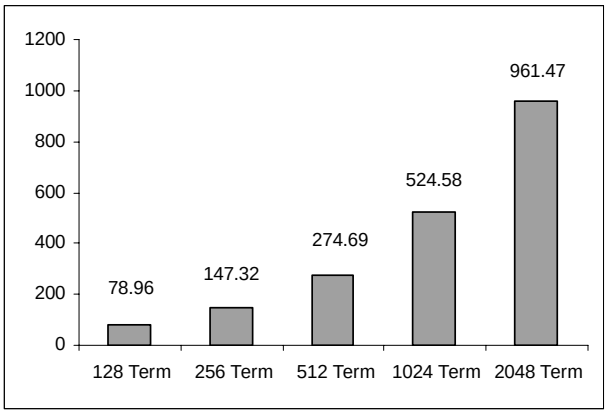


Fig. 14. Various size HAPI units.

VI. CONCLUSION

HAPI sustains high search performance through hardware acceleration and is ideally suited for consumer devices that maintain small search repositories. In comparison to a corresponding software implementation, up to a three order of magnitude improvement was found for some configurations. Using a pruned index algorithm improves response time and the performance of inverted index files. We developed a re-configurable and reusable hardware architecture called HAPI that maintains a pruned inverted index. Using a parameterized design, a varying number of varying length terms can be stored in each system. The HAPI system executes as either an attached co-processor or as a master-slave component.

We developed HAPI that maintains a pruned inverted index. Efficient text search was our primary problem. The HAPI design approach achieved high performance through hardware parallelism and flexibility to changes in retrieval algorithms and data structures through reconfigurable computing. HAPI can be adopted by many applications that use inverted index or pruned index algorithms. HAPI can easily be modified for such application. Finding those applications and adjusting HAPI for them can be done in the future.

Using 500 web documents, we analyzed our implementation of the HAPI component. The timings of a software pruned inverted index implementation and HAPI were compared over 182736 term operations. The various term length HAPI systems evaluated outperformed the fastest average speed recorded on the software approach. Intellectual property related discussions are ongoing.

REFERENCES

[1] S. K. Agun and O. Frieder, "HAT: a hardware assisted Top-Doc inverted index component," *ACM Twenty-Sixth SIGIR*, Toronto, Ontario, Canada, July 2003.

[2] M. Aldwairi, T. Conte, and P. Franzon, "Configurable string matching hardware for speeding up intrusion detection," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 1, pp. 99-107, 2005.

[3] A. Armstrong and J. Jiang, "Watermark embedded DCT indexing keys for portable imaging devices," *International Conference on Consumer Electronics*, Digest of Technical Papers, pp. 126-127, 2002.

[4] H.-M. Bluthgen, and T. G. Noll, "A programmable processor for approximate string matching with high throughput rate," *IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pp. 309-316, 2000.

[5] D. Carmel and et al., "Static index pruning for information retrieval systems," *ACM Twenty-Fourth SIGIR*, pp. 43-50, Sept. 2001.

[6] P.-Y. Chen and R.-D. Chen, "An index coding algorithm for image vector quantization," *Transaction on Consumer Electronics*, vol. 49, no. 4, pp. 1513-1520, Nov. 2003.

[7] Fast Search Chip, www.fastsearch.com, 2002.

[8] E. K. Kang, S. G. Jahng and J. S. Choi "A new indexing method for video retrieval using the rosette pattern," *Transaction on Consumer Electronics*, vol. 46, no. 3, pp. 780-784, 2000.

[9] S. Li, T. Ikenaga, H. Takeda, M. Matsui and S. Goto, "A hardware implementation of a content-based motion estimation algorithm for real-time MPEG-4 video coding," *Transaction on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E89-A, no. 4, pp. 932-40, 2006.

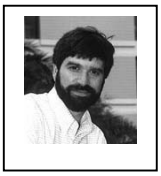
[10] J. L. Nunez-Yanez, V. A. Chouliaras, D. Alfonso, "Hardware assisted rate distortion optimization with embedded CABAC accelerator for the H. 264 advanced video codec," *International Conference on Consumer Electronics*, pp. 95-96, 2005.

[11] B. C. Song; N. H. Kim, D. K. Lim, T. H. Kim, J. H. Ko, and K. W. Chun, "Fast multi-resolution motion estimation algorithm and its VLSI architecture," *International Conference on Consumer Electronics*, pp. 71-72, 2005.

[12] TextFinder FDF4, www.paracel.com/products/textfinder.html, Paracel Corporation, 2002.

[13] H. Xu, Y. Mita, and T. Shibata, "Intelligent internet search applications based on VLSI associate processors," *Symposium on Applications and the Internet*, pp. 230-237, Feb. 2002.

S. Kagan Agun received Ms degree in computer science in 1996 and PhD degree in 2004 from Illinois Institute of technology. He was a student member of IEEE from 1994 to 2004, He has lectured at Illinois Institute of Technology in the Computer Science Department while he is completing his doctoral studies. He was a computer consultant to the information technology companies since 2004. His research interests include information retrieval, hardware support to information retrieval and reconfigurable computing.



Ophir Frieder (SM'93, F'02) is the IITRI Chair Professor of Computer Science and the Director of the Information Retrieval Laboratory at the Illinois Institute of Technology. His research focuses on scalable information processing systems. In addition to the IEEE, he is a Fellow of the AAAS and ACM.