

# Parallel Processor Core for Semantic Search Engines

Suneil Mohan, Aalap Tripathy, Amitava Biswas, Rabi Mahapatra  
Department of Computer Science and Engineering,  
Texas A&M University,  
College Station, Texas, USA  
{suneil,aalap,amitabi,rabi}@cse.tamu.edu

**Abstract**—Superior and fast semantic comparison improves the quality of web-search. Semantic comparison involves dot product computation of large sparse tensors which is time consuming and expensive. In this paper we present a low power parallel architecture that consumes only 15.41 Watts and demonstrates a speed-up in the order of  $10^5$  compared to a contemporary hardware design, and in the order of  $10^4$  compared to a purely software approach. Such high performance low power architecture can be used in semantic routers to elegantly implement energy efficient distributed search engines.

**Keywords**-Meaning comparison; hardware accelerator; green computing; information retrieval; dot product computation.

## I. INTRODUCTION

Increasing computation speed by deploying multiple processors and running parts of the computation in parallel is a recognized method of speeding up processing. However, this coarse grained hardware-thread approach is expensive, when the number of processors/cores required becomes large ( $> 100$ ). Due to significant resource management overhead, the speedup only scales sub-linearly with number of processors. In addition, this approach requires a considerable re-design effort, involving: designing a new memory controller, interconnect architectures and a new programming model including parallel compilers and libraries.

In applications which have short parallel paths/threads with a small memory footprint, there is an opportunity to deploy an alternative fine grained circuit level parallelization design. This requires a relatively inexpensive co-design effort, involving: design of a hardware centric algorithm & mapping parallel algorithmic steps to simple digital circuit instances which can then be replicated and integrated by simple interconnects.

In this paper, we present a low power semantic comparison core that follows the above fine grained parallelization approach for an important application area (search), and we demonstrate impressive speedup and power saving benefits. Semantic Comparison is carried out as a Tensor dot product operation. This can be used to improve the performance of a distributed web search engine [1] [2]. A specialized co-processor of this type is needed to materialize a distributed system appliance known as a semantic router [3]. A semantic router enables a novel overlay network called Semantic

Routed Network (SRN) where messages are routed to the destination based on its meaning [3]. The SRN can be useful in automatically reorganizing the index of a search engine thereby reducing the number of required servers [1]. Computational savings are also possible if the semantic comparison (dot product) processor core is used in a search engine index server to speedup index lookup operations. This hardware as co-processor allows for speeding up of tasks. Hence a single server can run a larger number of tasks, thereby obviating need for multiple servers and save energy (details in section V-C).

In [1], [2] we presented the mathematical fundamentals and the computational model for fast dot product computation of large vectors. [4] presented a preliminary design that performed part of this comparison algorithm (Details in Section III.B). In this paper, we present a hardware-centric design of a complete semantic comparator core which includes parallelized low-power hash computations, Bloom Filters [5], optimized interconnects and scheduling operations. ([4] assumed that Bloom Filter generation is done off-chip.) Through our designs and experiments in this paper, we show that hash computation can be done efficiently on chip with minimal increase in execution time and power consumption while performing more operations. We show that our lower power (15.41 Watts) single chip design speeds up semantic comparisons by 291,275 times compared to the hardware in [6] and by 41,796 times compared to software executing the same computation on an Intel Xeon processor for large sparse tensors. If traditional multiprocessor parallelization is used, an equivalent speedup will need  $\approx 41,796$  cores/processors which will be expensive and power hungry. Our design performs better for large sparse tensors (with large number of non-zero co-efficient basis vectors) compared to the available hardware designs [6],[7],[8].

This paper makes the following contributions:

- Implements a complete semantic comparison core for a semantic router with timing analysis.
- Support for large sparse tensors unlike existing hardware designs [6] - [8]
- Data-parallel FNV based hash computation.
- Provides analytical timing model for a complete semantic core.

- Low power efficient design allowing for scaling up.

As search has become a critical service with rapidly growing demand (13 billion search queries per month with 38% annual growth [9]), more accurate meaning representation and comparison that enables precise meaning based searching is important. Large sparse tensors have been proven to be necessary for accurate meaning representation to enable a better search experience

This complete hardware design enables realization of viable semantic routers enabling Semantic Networks and hence leading to massive power savings in search engine data centers.

The rest of the paper is organized as follows: In section II we present an application for semantic routing and demonstrate how energy can be saved. We also present preliminary explanations of traditional search paradigms and explain the core operational principles of our design. In Section III we introduce the algorithms being used and propose hardware centric algorithms. We explain our design details in Section IV and present analytical timing model in Section V. Power, timing analytics and comparison with prior work is presented in Section VI. We conclude our paper in Section VII.

## II. APPLICATION CONTEXT

### A. Semantic Routing

The concept of semantic routing is best illustrated using the following example. Mary, a graduate student of Microbiology wants details about the Swine Flu outbreak. She asks her friend David who is a post-doctoral researcher if he has specific information. David feels that his supervisor Dr. Frank is more knowledgeable on the topic, so David forwards Mary's request to Dr. Frank. Dr. Frank responds to Mary's questions and offers to help her. In this scenario, each person is a "semantic router", who determines the next hop based on who is the best (knowledgeable) person to handle this query. The query here is the 'request for information on Swine Flu' and the best identified resource was Dr. Frank who responds to Mary with the offer to help.

"Semantic routers" have been proposed to have routing tables for destination lookups based on the search topics (query keys) and will route messages based on them. A SRN will be built by an overlay network of semantic routers physically connected by the IP network or possibly by an underlying web-service infrastructure; therefore what may be actually returned could be IP addresses, URLs or web-service access mechanisms of the resources. More details on the working of the SRN are provided in [3],[10].

### B. Need for Suitable Index Organization

One direct application of a SRN is to enable the automatic reorganization of the search-index of a distributed search engine. This would allow for the reduction in the number of servers deployed by the search engine.

To achieve scalability and speed for computationally intensive search, search indexes are deployed as large distributed systems in data centers [18]. For example, Google indexes tens of billions of web-pages/documents and services nine billion queries per month (or  $\approx 3500$  per second) [9]. Google is estimated to have over half a million servers in multiple data centers [11], each of which consumes tens of megawatts of power. Such high power demand is a major hindrance to freely deploying data centers [12] and will be a significant cause of green-house gas emissions [13] as web based search engine infrastructures scale up to match the growing web. In addition, as search engines scramble to provide more meaningful and relevant responses without increasing the response time (real-time search), the computational load per query increases. Hence there is a pressing need for energy efficient systems and infrastructure designs that can enable fast, efficient semantic (meaning) based search.

### C. Need for Semantic Comparison

In [3], we explained how a Semantic Routed Network (SRN) comprised of fast response appliances called semantic routers can be used for two purposes: (1) to selectively forward/route a (query) message to an index shard based on the meaning of the query; and (2) to automatically re-distribute index entries based on meaning of the documents/objects. These semantic routers use a data structure called semantic key to represent meaning of a message and use semantic comparison (comparison of semantic keys) as a primitive computation to decide the next hop message destination(s) [3]. In addition to index re-organization and query delivery, meaning comparison can be also used to carry out index lookup necessary for meaning based search operations [2].

### D. Vector Based Semantic Comparison

Contemporary search engines use vector models for automatic text retrieval [14]. Techniques described by us previously in [1] extend these vector models for more efficient semantic comparisons using tensors.

Algebraically, a tensor is represented as a sum of scalar ( $w_i$ ) weighted basis vectors ( $v_i$ ) as shown in Fig. 1. Each basis vector denotes elementary meaning which typically is a term or a phrase (character strings) from a controlled vocabulary/dictionary or selectively picked from a text object (e.g. a sentence in Fig. 1) and assigned weights (scalar coefficients) depending on the model used [14].

In Fig. 1, we show text fragments from two documents and their corresponding tensor representation (D1 & D2). The similarity between the meanings represented by the two vectors is given by their cosine (dot) product. The dot product of two vectors/tensors is given by the sum of products of weights of the basis vectors (having non-zero weights) that are common in both vectors.

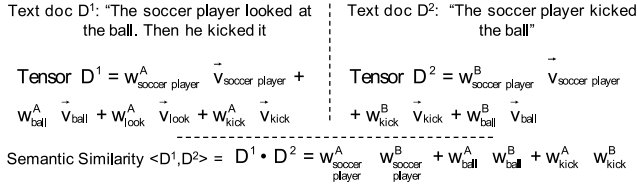


Figure 1. Tensor Model of Semantic Comparison

In a computer, these tensors can be represented by a table of character strings and their corresponding coefficient weights as shown in Fig. 2.

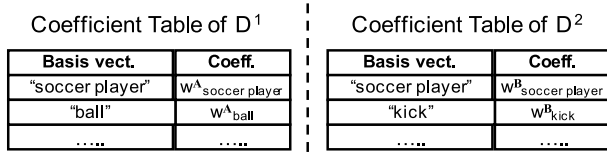


Figure 2. Representation of Tensors in computer memory.

### III. THE PROBLEM OF DOT PRODUCT COMPUTATION

A representation of composite meanings is necessary to ensure "meaningful" semantic comparison as proposed. To enable this, semantic comparison operates on an infinite dimensional vector space leading to very large tensor sizes ( $\approx 10^3$ ).

Dot product computation on a pair of tensors involves (1) identifying their common basis vectors (strings) and (2) pairing their weights for multiplication and summation.

If the number of basis vectors in the two meaning tensors are denoted as  $n_1$  &  $n_2$ , then when a sequential processor is used, this identifying task has a time complexity of  $O(n_1 \cdot n_2)$  or  $O(\log n_2)$  or  $O(n_1 \cdot n_2)$  depending on whether a binary or linear search algorithm is used. However, we do this identification using a Bloom filter based algorithm, which is hardware-centric and has smaller time complexity (details in Section III.A).

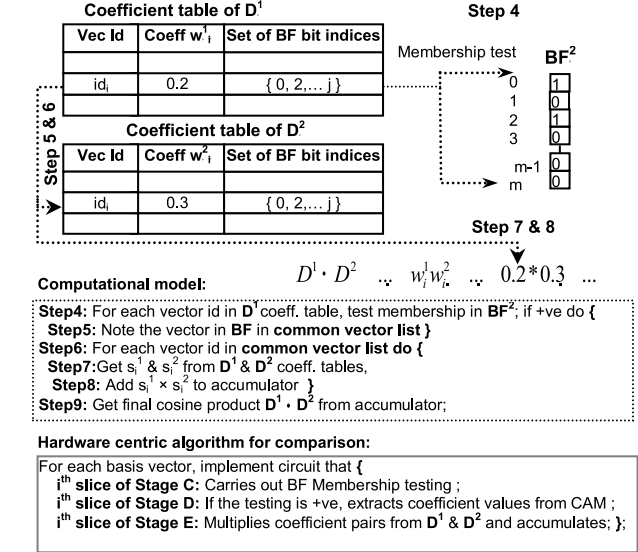


Figure 4. Cosine Similarity Comparison

Since the tensors are large, the number of common basis vectors is expected to be small. In such cases, the multiplication operation becomes superfluous when a match is not found (coefficient is zero). Such superfluous operations can be tolerated with small finite-dimensional tensors. However for infinite (or large) dimension tensors, it is imperative to eliminate such superfluous operations because multiplication is computationally expensive. Our Bloom filter based algorithm eliminates these superfluous operations by only identifying the matching basis vectors having non-zero coefficients.

Since the tensors are large, the number of common basis vectors is expected to be small. In such cases, the multiplication operation becomes superfluous when a match is not found (coefficient is zero). Such superfluous operations can be tolerated with small finite-dimensional tensors. However for infinite (or large) dimension tensors, it is imperative to eliminate such superfluous operations because multiplication is computationally expensive. Our Bloom filter based algorithm eliminates these superfluous operations by only identifying the matching basis vectors having non-zero coefficients.

### IV. HARDWARE CENTRIC ALGORITHMS

In [1],[2] we proposed a method to compute the similarity of two given tensor/vectors. This method constitutes a two part computation. First an abstract data structure is generated from a given tensor/vector (explained in sub-section A), next two data structures of the two tensors are used to derive their dot product which gives the similarity value (explained in sub-section B). Based on this two part computational model, we propose hardware centric algorithms and a consolidated hardware design in the following sub-sections.

#### A. Data Structure Generation & Analysis

The abstract data structure proposed in [2] has two components: (1) a Co-efficient table and (2) a large  $m$  ( $\approx 128K$ ) bit long Bloom Filter (BF) using  $k(= 7)$  hash functions [5]. The details of this data structure are shown

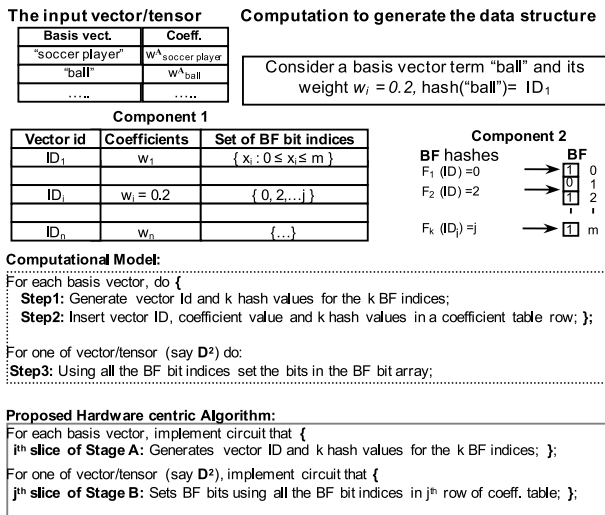


Figure 3. Descriptor Data Structure Generation

in Fig. 3 Each row of the coefficient table consists of three columns: (1) Vector ID (e.g. “ID1” in Fig. 3); (2) 16 bit fixed point scalar coefficient of a basis vector (e.g.  $w_i$ ); and (3) Set of BF indices ( $x_1 : 0 \leq x_i \leq m$ ). The combination of the coefficient table and the bloom filter represents the vector/tensor representing the meaning of an object (text/non text document). A 64 or 128 bit hash of a basis vector term is generated and inserted as the Vector ID in the coefficient table (Fig. 3, Computation Model). To generate a set of  $k$  BF indices, each Vector ID (or the basis vector character string) is further hashed by  $k$  hash functions and the resultant indices are stored in the third column while the corresponding bit locations are set in the BF. The Step 1,2 of the computational model is materialized by the stage A and Step 3 by stage B in the hardware centric algorithm (Fig. 3). The notion of slices and stages are explained in detail in section IV.

For each basis vectors, the time complexity of stage A (steps 1,2) is  $O(k)$ , and for total  $n$  ( $< 10^3$ ) basis vectors the order of the entire data structure generation computation is  $O(n_1k)$ . As computation of each basis vector is independent, each of these can be computed in parallel using  $r$  ( $\approx n_1$ ) circuits within  $O(n_1 \cdot k/r)$  time. For  $r \ll n$ , this is  $O(k)$  with  $k$  generally  $< 20$  [5].

### B. Dot Product Algorithm & Analysis

Figure 4 shows the second part of the computation model as presented in [1] and the corresponding hardware centric algorithm. Two data structures (D1 and D2) are taken as input to generate the cosine similarity value ( $D1 \cdot D2$ ) as follows: (a) Step 4 & 5 (stage C in the hardware algorithm): Identify the common basis vectors from the first coefficient table (Component 1, Figure 4) by verifying which vector IDs are in the second BF (BF2, Figure 4) by using the set of BF bit indices in the first table. (b) Step 6 & 7 (stage D): If a vector is present in the BF2 then we use that common vector ID as the key, and extract the coefficient value from the coefficient lookup table of the second data structure (Coefficient Table of D2). This is carried by a content addressable memory (CAM) lookup mechanism with vector ID as the key. (c) Step 8 & 9 (stage E): Multiply the pair of coefficients for each identified common basis vectors (from both tables) and add all the products to get the similarity metric.

For each basis vectors the complexity of step 4, 5 is  $O(k)$  or  $O(1)$  depending on the type of Bloom filter used [5]. The order of step 6 & 7 is  $O(1)$  and step 8 & 9 is  $O(1)$ . Therefore for  $n$  basis vectors the order of the entire algorithm is  $O(n_1 \cdot k)$  or  $O(n_1)$ . The computation of each basis vector being independent, a parallel computation using  $r$  ( $\approx n_1$ ) circuits has time complexity of  $O(n_1 \cdot k/r)$  or  $O(n_1/r)$ , which is  $O(k)$  or  $O(1)$  for  $r \ll n$ .

## V. PARALLEL HARDWARE ARCHITECTURE

### A. Working

The parallel threads in the computational models shown in Figure 3 & Figure 4 are short (small, simple computations) and require small amount of memory. This makes these computations unsuitable for large scale multi-processor based parallelization because the multi-processor/core systems require significant amount of time to distribute and consolidate the threads across large number of cores. We explain this in further detail in Section VI.C. In addition, general purpose processors take more time to compute simple arithmetic and logical operations (e.g. addition operation takes 6 clock cycles on an Intel Xeon, compared to 1 cycle on a hardware adder). For short threads, these overheads are significant compared to the thread execution time, therefore multiprocessor parallelization will not give good speedup. Hence we use an alternative parallelization approach as follows. The proposed parallel processor core for semantic search engines has five basic stages as shown in Figure 5 which do the following operations: Stage (A) generates vector IDs and  $k$  hash values for each basis vector; Stage (B) populates the Bloom filter of descriptor D2 using the  $k$  hash values; Stage (C) identifies the common basis vectors using BF membership testing; Stage (D) extracts the matching pairs of scalar coefficients; and Stage (E) multiplies the corresponding pairs of scalar coefficients and calculates the sum to obtain the dot product. For each row in the coefficient table, steps 1 and 2 of the algorithm in Figure 3 and steps 4 to 8 in Figure 4 can be executed in parallel, as processing of each of these rows are independent. Each of these parallel processing instances, which can be considered as a thread, is executed by each horizontal slice of each circuit stage (A to E), as shown in Figure 5. There are  $n$  slices in stage A to C, and  $b$  slices in stage D and  $p$  slices in stage E. Stage E also consolidates all the processing. For maximum parallelization we would like to use maximum number of slices in each stage as necessary, so  $r \ll n$ , where  $n$  is the number of rows in a coefficient table (i.e. number of basis vectors). In [2] we proved that in a semantic routed network,  $c$  (the number of common vectors) is  $\frac{1}{1000} \times n = 0.01\%$  of  $n$ . The expected number of CAM lookups that will take place is given by  $(c + (n - c) \cdot P_{false+ve})$ , where  $P_{false+ve}$  ( $\approx 10^{-9}$ ) is the probability of false positive for the Bloom filter being used ( $m \approx 10^5, k \approx 7$ ) [5]. Therefore we expect only small number of CAM lookups (0.01% of  $n$ ) and hence we use a small number of slices in stage D ( $b \ll n$ ). Stages C-E in [4] was optimized with addition of “interconnect-2”. This helps reduce power consumption significantly because of a smaller number of CAM blocks. This essentially reduces the design requirement from  $n$ -CAM blocks to  $b$ -CAM blocks (where  $b \ll n$ ).

Further, as multiplier units are expensive in terms of power and area requirements, we use an even smaller

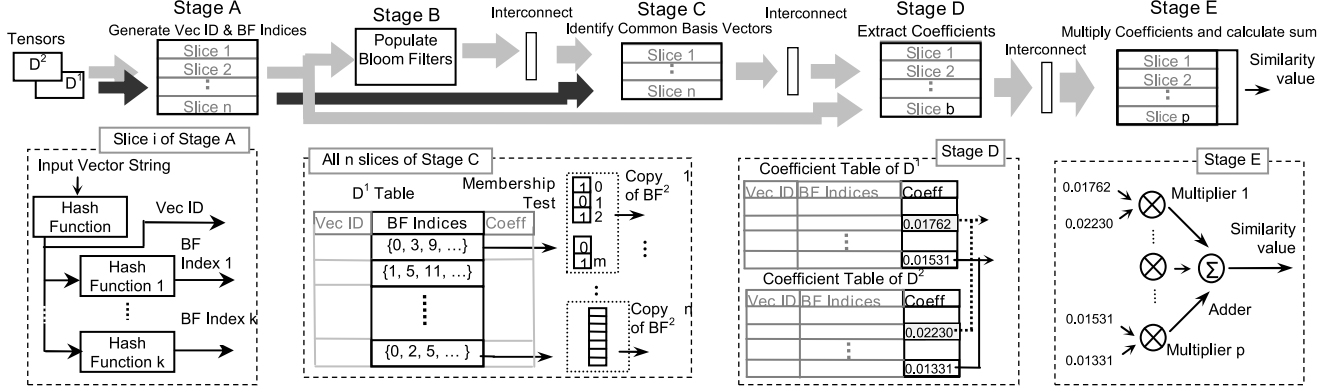


Figure 5. Overall Architecture

number ( $p < b, p \ll n$ ) of slices in the multiplication stage E.

### B. Bloom Filter Generation (Indices & Consolidation)

In Figure 6, we present a block schematic of Stage-A where the BF indices are generated by the various hash functions. We use the FNV hash algorithm as the first stage of the BF index generation operation. The 64 bit output of the FNV hash module is duplicated. The first copy ( $f_1$ ) is rotated by 33 bits and then truncated to 17 bits using XOR folding. The second copy ( $f_2$ ) is directly truncated to 17 bits using XOR folding. For each BF index  $BF_i$  (where  $0 \leq i \leq k$ ); a copy of  $f_1$  is rotated by  $i$  bits and then XOR'd with  $f_2$ . For example, the second BF index BF2 is obtained by the XOR operation between  $f_2$  and  $f_1$  rotated by two bit positions. This enables the creation of  $k$  different index values in parallel which are then used to set the appropriate BF indices. As shown in Figure 5,  $n$  copies of this stage are used in parallel to allow for the BF indices of all  $n$  rows to be processed in parallel. This allows for bloom index generation in finite time.

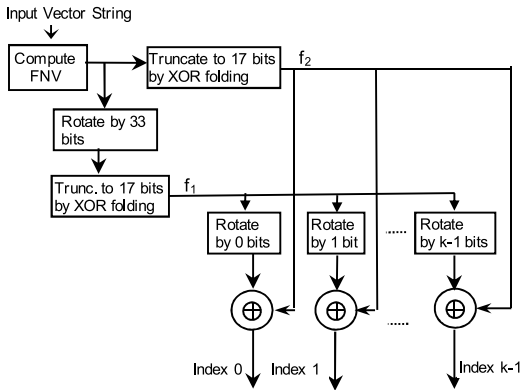


Figure 6. Generating Vector ID and Bloom Filter Indices

These bloom indices allow parallel setting of the corresponding bit position in a memory element, hence creating

a Bloom Filter for each row (Stage B in Figure 7). The  $n$  individual BFs (one per row) are then consolidated into a single BF using cascaded OR-Gates (as shown in Figure 7) and distributed to the  $n$  rows in stage-C (in Figure 5) for use in further stages.

### C. Interconnects

To facilitate smooth flow of data between the unequal number of slices in different stages and enable maximum utilization of slices, special interconnects are used between the stages (Interconnects 1, 2 & 3 as shown in Figure 5). The Interconnect 1 between stage B and C distributes the consolidated Bloom filter to  $n$  slices in stage C. This is a simple distribution network similar to a Clock-Tree. Since, Stage C (in Figure 5) identifies the common basis vectors using BF membership testing, we now need to extract their corresponding coefficients.

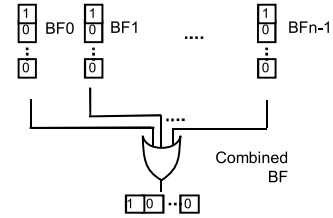


Figure 7. Consolidation of Bloom filters.

Interconnect 2 (as shown in Figure 8) between stages C and D performs this function by scheduling the parallel extraction of these coefficient values. This interconnect operates on  $n$  one-bit signals from Stage-C (RowSelect) that indicates that the corresponding row is a potential candidate match. RAM units shown in Figure 10 contain the coefficient values of Table 1 (b-RAM copies are present). The b-CAM units on the other hand store copies of the coefficient values of Table 2. RowSelect bus consists of  $n$  one-bit signals from Stage-C that indicates which rows of Table 1 had a BF Membership test match. The interconnect logic schedules

data reads from the RAM units (corresponding coefficients of Table1) in a staggered manner because Stage D has  $b$  slices whereas Stage C has  $n$  slices ( $b \ll n$ ).

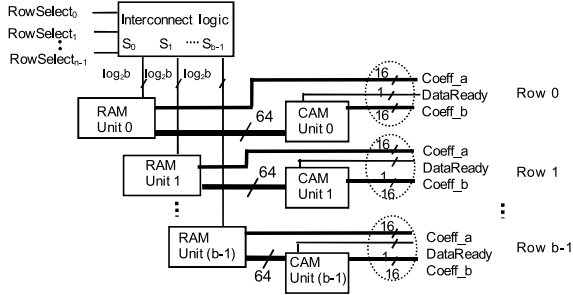


Figure 8. Interconnect Details

## VI. PROCESSING TIME ANALYSIS

In this section we present an analytical timing model for our proposed hardware architecture. The total processing time  $t$  taken by our design can be split into two components  $t_1$  and  $t_2$ . Where  $t_1$  is the time taken by the Bloom Filter generation, consolidation and distribution steps (Section IV.B) and  $t_2$  is the time taken by the remaining Stages C-E (including the interconnects). The total time taken by Algorithm-1 involves creation of the data structure. Since the two input tables have different processing requirements, the time taken for this stage ( $t$ ) can be denoted as a sum of computation time for Table-1 ( $t_{1a}$ ) and computation time for Table-2 ( $t_{1b}$ ).

$$t = t_1 + t_2 \quad (1)$$

$$t_{1a} = t_{FNV} + W + O + D \quad (2)$$

$$t_{1b} = t_{FNV} + W \quad (3)$$

Here,  $t_{FNV}$  is the number of cycles taken to process each input string into its corresponding hash value (using the FNV hash algorithm),  $W$  is the time taken to write the BF indices. (common to both tables) As explained previously,  $n$  individual copies of the BFs need to be consolidated into a single BF prior to distribution, hence  $O$  is the time taken to OR together the individual bloom filters. (It should be obvious that only Table-1 needs to undergo this step).  $D$  is the time taken to load the multiple copies of the BFs to the corresponding slices in Stage-C (common to both tables). In our design  $W = k$ , and  $O = D = 1$ .

Algorithm-2 involves the computation of cosine similarity. The computation time  $t_2$  can be denoted as:

$$t_2 = \left( \left\lceil \frac{n}{r} \right\rceil * k \right) + E + \left\lceil \frac{G}{b} \right\rceil + S + \left\lceil \frac{|n * c|}{p} \right\rceil + L + A \quad (4)$$

where

$$G = (c + (n - c) \cdot P_{false+ve}) \quad (5)$$

Where,  $n$  = number of basis vectors in the smallest vector/tensor,  $r$  = number of parallel bloom filter test circuits,  $k$  = Number of Bloom Filter Indices,  $E$  = Number of cycles needed to extract the coefficients (using CAM),  $b$  = Number of CAM blocks,  $S$  = Number of cycles to schedule the multiplications,  $c$  = percentage similarity between the two tables being compared,  $p$  = Number of multipliers available,  $L$  = Latency of each multiplier,  $A$  = Latency of the adder and  $\lceil \cdot \rceil$  denotes the ceiling function. In our design,  $E = S = A = 1$  and  $L = 5$ . Here the bottleneck is the multiplier stage as only a few multipliers can be used because they are expensive in terms of power and silicon area. The BF will help us identify the  $c + (n - c) \cdot P_{false+ve}$  suspected matching basis vectors (where  $P_{false+ve}$  is the probability of BF false positives). In the CAM lookup stage, the  $c$  suspects are confirmed and  $(n - c) \cdot P_{false+ve}$  false positive suspects are rejected. Hence, there are only  $n \cdot c$  multiplications that need to be carried out by the  $p$  multipliers in the last stage (Figure 5). The computation time is not completely deterministic due to the potential for Bloom filter false positives.

## VII. EXPERIMENTS AND RESULTS

### A. Tools & Experiments

The functional verification of the proposed hardware design was performed using ModelSim from Mentor Graphics after being implemented in Verilog. The design was then synthesized using Design Compiler from Synopsys at a clock speed of 3Ghz along with the 90nm technology library from TSMC. The reported circuit power results were obtained from this synthesis. The memory power data was obtained using the CACTI power model. The synthesizability and the feasible circuit power figures establish the viability of the design. The execution timing for a representative server class processor (Intel Xeon) was measured for software code which computes the dot product. The dot product software code (shown in Fig. 9) identifies the common basis vectors by first constructing a balanced binary search tree of basis vectors (char strings) and coefficient pairs (with vectors as search keys) from the second table, and searching for each vector from the first table. Thus this computation is of the order of  $O(n_1 \cdot \log n_2)$ . The balanced tree was implemented using the C++ STL's highly optimized map container which implements Red-Black tree. The pseudo-code in Fig. 9 also indicates the segment for which execution time is measured.

We do not consider an alternative method involving the software implementation of the Bloom Filter based approach. This is because the Bloom Filter approach involves multiple hash computations for each basis vector and would be computationally more expensive and hence suboptimal. Hence this approach is not suitable for comparison. We

```

// Table data structure declarations.
struct rows{ string basis_vector; float coefficient };
.....
rows coeff_table1[NUM_BASIS_VECT1], coeff_table2[NUM_BASIS_VECT2];
.....
// Declaration for map data structure for the red-black tree.
std::map < string, float> rbtree;
std::map < string, float>::iterator rbtree_itr;
.....
//Execution time measurement starts here
//Code to construct tree
for (i = 0; i < NUM_BASIS_VECT1; i++)
    rbtree.insert( std::make_pair( coeff_table1[i].basis_vector, coeff_table1[i].coeff) );
//Code to search tree
for (i = 0, dot_prod =0; i < NUM_BASIS_VECT2; i++) {
    rbtree_itr= rbtree.find(coeff_table1[i].basis_vector);
    if ( rbtree_itr!=rbtree.end() )
        dot_prod = dot_prod + ( (*rbtree_itr).second * coeff_table2[counter].coeff); }
//Execution time measurement ends here

```

Figure 9. Pseudo-code for optimum software comparison

use clock cycles in lieu of execution time to compare performance of the designs in a clock speed neutral manner. This was done to fairly compare the hardware designs that have been implemented and evaluated with different clock speeds [6] - [8].

For all experiments below, we considered: number of basis vectors  $n = 1024$ , number of parallel circuit instances  $r = n$ , number of common basis vectors  $c = 10^2$  (10% of  $n$ , which is very conservative estimate, see Section IV), number of CAM units  $b$  in stage D = 16 ( $\approx 20\%$  of  $c$ ), number of multipliers  $p$  in stage E = 8 (half the number of CAM units), BF length  $m = 131,072$  (optimum), number of BF hash functions  $k = 7$  (optimum), and BF false positive probability  $P_{false+ve} = 1.2 \times 10^{-9}$ , unless different values are implied. By choosing a low number of multipliers and CAM blocks, we achieve a low power figure but take a small increase in the overall processing cycles.

### B. Results & Comparison with Related Work

In Table- I we present the superior performance of our design in terms of speed (clock cycles to perform semantic comparison for a pair of vectors) and circuit power draw (for 90nm technology, 3Ghz) compared to an Intel Xeon processor (a representative server class processor). The Intel Xeon (3Ghz version), used in this example, has a maximum Instruction Per Cycle (IPC) figure of 4. All other high performance sequential processors have IPC of very similar order. Hence the performance cannot be significantly improved any further on a traditional server class processor. We discuss the limitations of multi-core/GPU based systems in the next section (Section. V-C).

To the best of our knowledge, we are the first to investigate the concept of hardware based semantic comparison. Hence direct comparison against other semantic comparators is not possible since they do not exist. However, hardware to compute cosine similarity (which is a part of our computation) has been investigated by [6] - [8]. Fig. 10 shows the comparison of speedup of our design against that of the other available designs (compared to the execution time of

Table I  
SUPERIOR PERFORMANCE OF HARDWARE DESIGN

Comparison	Execution time (in cycles)		Power Draw
	c=10%	c=100%	
Proposed Hardware	131	303	15.41 Watts
Intel Xeon	5,456,464	6,050,780	40-80 Watts/core
Comments	Speedup of 41,796	Speedup of 19,969	<b>74% less power</b>

an equivalent efficient software code). We show comparison with both  $c=100\%$  and  $c=10\%$  by converting the reported execution times in [6] - [8] to clock cycles.

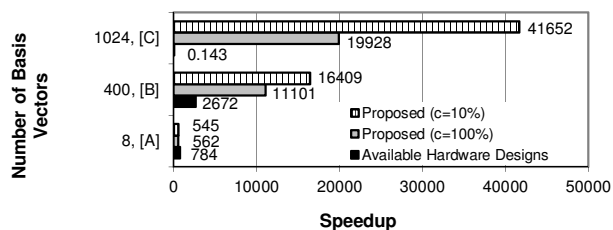


Figure 10. Comparison of Speedup against other hardware designs, [A]=Ref. [6], [B]=Ref. [8], [C]=Ref. [7]

The other hardware designs do not take into consideration the number of common basis vectors. Our design is consistently doing better due to fine grained parallelism in our hardware for large meaning vectors (number of basis vectors = 400, 1024). Such parallelism has not been exploited by other hardware based designs [6] - [8], which carry out the computations sequentially.

The addition of Interconnect-2 to the design presented in [4] allowed us to significantly reduce the number of CAM blocks required. This along with a lower number of multipliers, contributed towards a reduction in power consumption from the then reported figure of 109W [4] to 15.41 Watts in this design, while additional computational stages were added.

In Table- II we present a comparison of our hardware with those presented in [6] - [8] and show the factors of speedup by which our design performs better. For large meaning vectors (number of basis vectors = 1024) our design gives a speedup increase of 291,275 times for  $c=10\%$  and 139,357 times for  $c=100\%$  compared to the hardware in [6]. The other hardware designs do not address power issues and hence it is difficult to compare power consumption.

Our design can handle a larger number of input rows ( $n > 1024$ ) by splitting the rows into multiple partitions of 1024 rows each. For example if the input row size were 2048, it would be split into two partitions of 1024 rows each and then processed. However, since we were unable to find other work that handles such large numbers of input sizes, we do not

Table II  
SPEEDUP COMPARISON WITH OTHER HARDWARE DESIGNS

Number of Basis Vectors	Compared Against	Improvement in Speedup (times)	
		c=10%	c=100%
8	Ref. [8]	0.696	0.717
400	Ref. [7]	6.14	4.15
1024	Ref. [6]	291,275	139,357

present this data in this paper. Fig. 11 shows the comparison of number of cycles required by the proposed hardware and Xeon for different values of  $n$  (number of basis vectors) to show their relative scaling behavior with  $n$ .

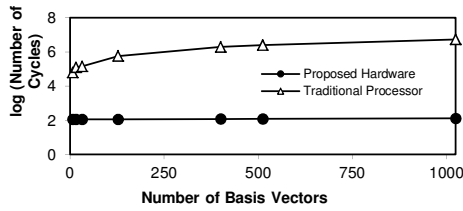


Figure 11. Variation of number of cycles between proposed hardware and traditional processor

Fig. 12 shows that the execution clock cycles varies linearly (with a small slope) with number of basis vectors  $n$  (scaling behavior) as there are limited number of CAM and multiplier units.

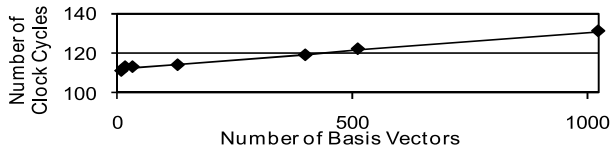


Figure 12. Variation of execution time with number of rows.

Fig. 13 shows the variation in Speedup with change in percentage similarity among the two Tables. Smaller value of  $c$  leads to lesser number of multiplications. The variation is bounded and gives a speedup of at-least 19,969 for  $c = 100\%$  (worst-case)

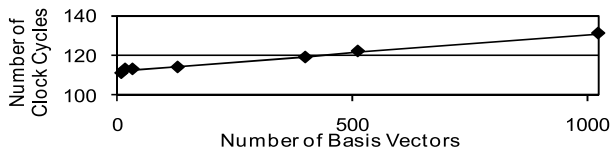


Figure 13. Variation of speedup with Similarity for  $n=1024$ ,  $k=7$ ,  $b=16$ ,  $p=8$

All hardware clock cycles reported in this section were obtained by averaging computation time using a 1024 element long data structure for 100,000 iterations using a

software simulation of the proposed hardware. As explained earlier, the computation time defined in Equations 4 and 5 is not fully deterministic because of the false positive rate of the Bloom Filter ( $P_{false+ve}$ ). Our results show that the computational time in our framework not only converges but is also bounded. We restricted our results to those that deal with the core computation of our design - dot-product, since designing a complete (semantic) search engine is out of the scope of this paper.

### C. Benefits, Need & Utility of the Proposed Design

Multi-core GPUs such as the NVIDIA Fermi architecture have a limited number of cores ( $\approx 500$  [15]), hence can't give very large ( $\approx$  tens of thousands) speedup. To obtain the large speedup as offered by our design on a traditional implementation, large computing clusters (or clusters of GPUs) or super computers would be required. However, such architectures pose significant processing time (setup time) overheads. The parallelism of software threads is limited by the actual number of hardware threads in the processor. Parallel software threads (when greater than the number of hardware threads) run by performing context switches as processor time is divided between the threads. Context switches are inherently expensive. A speedup of 19,969 would need 19,969 parallel hardware threads and 19,969 cores/processors (assuming one thread in each core), in such system. Creation and consolidation of large number of threads is very expensive due to the fixed overheads. These overheads are tolerable [16] in situations when each thread's execution time is large enough compared to its setup time. Since our dot product computation has short execution time (one multiplication followed by an addition), it is not worthwhile to run them on a multi-core/processor system. A 19,969 Xeon-core based system could achieve similar speedup but would consume  $19969 \times 40 = 798,760W$ . In contrast (as shown in Table-1), our design consumes only 15.6W for the same speedup. Therefore our design is efficient in terms of power and cost. Hence, approaches involving GPUs or large computing clusters are not a viable solution to our problem of meaning comparison.

Further savings are also possible if our design is used in the index servers to perform computation necessary during index lookup based on key comparisons as explained in [2]. This is because our design can compute faster (41,796 speedup). Considering that we can create a co-processor using this design to take advantage of the large 41,796 speedup, the dot product computations can be performed much faster in this co-processor (rather than the server's main CPU). Hence in a distributed system where a large numbers of servers work in parallel and share a large number of dot product processing tasks (arising out of large number of simultaneous user queries [17] ), a single server can possibly replace in order of 41,796 servers. Assuming a regular server consumes 500 Watts, then a server with

this co-processor will consume  $500 + 45 = 545$  Watts and can replace 41,796 servers each consuming 500W. So power savings of a factor  $\approx \left(1 - \frac{545}{41796 \times 500}\right) \approx 99.99\%$  is theoretically possible. Actual savings will be smaller due to implementation issues.

It is possible to obtain data transfer rates of  $\approx 10$ GBps using the PCI-Express bus to integrate the co-processor with the rest of the system. However, this would be a system integration issue and is outside the scope of this paper.

## VIII. CONCLUSIONS

In applications where threads are short and need limited memory, fine grained circuit level parallelization can be a viable alternative to multi-core processor enabled parallelization. The proposed parallelization scheme avoids the expensive hardware-software design effort and high overheads associated with multi-core processor based designs. In this paper we showed how a hardware circuit algorithm can enable circuit level parallelization, deliver superior performance as compared to contemporary hardware designs or purely software implementations. We presented the application context and design specifications, which involved: design of a hardware centric algorithm; mapping of the algorithm to hardware. This work also has a special significance from a “search” application viewpoint. Our design of the semantic comparator hardware is the key to realization of appliance called semantic routers which will enable considerably low power, less hardware intensive organization and more efficient operations of distributed index in a web search engine.

## REFERENCES

- [1] A. Biswas, S. Mohan, J. Panigrahy, A. Tripathy, and R. Mahapatra, “Representation of complex concepts for semantic routed network,” in *ICDCN*, ser. Lecture Notes in Computer Science, V. K. Garg, R. Wattenhofer, and K. Kothapalli, Eds., vol. 5408. Springer, 2009, pp. 127–138.
- [2] A. Biswas, S. Mohan, A. Tripathy, J. Panigrahy, and R. Mahapatra, “Semantic key for meaning based searching,” in *Semantic Computing, Proceedings of the 3rd IEEE International Conference on (ICSC 2009)*, Berkeley, CA, USA, September 2009, pp. 209–214.
- [3] A. Biswas, S. Mohan, and R. Mahapatra, “Search coordination by semantic routed network,” in *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th International Conference on*, 2009, pp. 1–7. [Online]. Available: <http://dx.doi.org/10.1109/ICCCN.2009.5235253>
- [4] S. Mohan, A. Biswas, A. Tripathy, J. Panigrahy, and R. Mahapatra, “A parallel architecture for meaning comparison,” in *IPDPS*. IEEE Computer Society, April 2010. [Online]. Available: <http://dx.doi.org/10.1109/IPDPS.2010.5470371>
- [5] A. Broder and M. Mitzenmacher, “Network applications of bloom filters: A survey,” *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.98>
- [6] D. G. Perera and K. F. Li, “Parallel computation of similarity measures using an fpga-based processor array,” in *Advanced Information Networking and Applications, Proceedings of International Conference on*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 955–962. [Online]. Available: <http://dx.doi.org/10.1109/AINA.2008.97>
- [7] D. G. Perera and K. F. Li, “On-chip hardware support for similarity measures,” in *Communications, Computers and Signal Processing, 2007, IEEE Pacific Rim Conference on*, 2007, pp. 354–358. [Online]. Available: <http://dx.doi.org/10.1109/PACRIM.2007.4313247>
- [8] M. Freeman, M. Weeks, and J. Austin, “Hardware implementation of similarity functions,” in *IADIS AC*, N. Guimarães and P. T. Isaias, Eds. IADIS, 2005, pp. 329–332.
- [9] A. Patriquin, “March search market share: Record query growth and the yahoo/microsoft search deal by the numbers,” *Compete, Inc*, April 2008, <http://blog.compete.com/2009/04/13/search-market-share-march-google-yahoo-msn-live-ask-aol-2/>; Accessed April 15th, 2009.
- [10] A. Biswas, S. Mohan, and R. Mahapatra, “Optimization of semantic routing table,” in *Computer Communications and Networks, 2008. ICCCN 2008. Proceedings of 17th International Conference on*, 2008, pp. 298–303. [Online]. Available: <http://dx.doi.org/10.1109/ICCCN.2008.ECP.69>
- [11] D. McKay, “Google searches, energy cost, carbon footprint, and cups of tea,” *Sustainable Energy*, January 2009, <http://withoutoutair.blogspot.com/2009/01/google-searches-energy-cost-carbon.html>; Accessed July 15th, 2009.
- [12] B. Botelho, “Gartner predicts data center power and cooling crisis,” *SearchDataCenter.com*, June 2009, <http://searchdatacenter.techtarget.com/news/1260874/Gartner-predicts-data-center-power-and-cooling-crisis>; Accessed June 15th, 2009.
- [13] U.S. Environmental Protection Agency, *Report to Congress on Server and Data Center Energy Efficiency Public Law*, ser. ENERGY STAR Program, August 2007, pp. 109–431.
- [14] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Inf. Process. Manage.*, vol. 24, no. 5, pp. 513–523, 1988. [Online]. Available: [http://dx.doi.org/10.1016/0306-4573\(88\)90021-0](http://dx.doi.org/10.1016/0306-4573(88)90021-0)
- [15] NVIDIA Corporation, *NVIDIA Fermi Architecture*. [Online]. Available: [http://www.nvidia.com/object/fermi\\_architecture.html](http://www.nvidia.com/object/fermi_architecture.html)
- [16] T. Kaldewey, J. Hagen, A. Di Blas, and E. Sedlar, “Parallel Search On Video Cards,” in *First USENIX Workshop on Hot Topics in Parallelism (HOTPAR 09)*. USENIX, March 2009. [Online]. Available: [http://www.usenix.org/event/hotpar09/tech/full\\_papers/kaldewey/kaldewey\\_html/](http://www.usenix.org/event/hotpar09/tech/full_papers/kaldewey/kaldewey_html/)
- [17] L. A. Barroso, J. Dean, and U. Lzle, “Web search for a planet: The google cluster architecture,” *IEEE Micro*, vol. 23, no. 02, pp. 22–28, March 2003. [Online]. Available: <http://dx.doi.org/10.1109/MM.2003.1196112>