Optimizing a Semantic Comparator using CUDA-enabled Graphics Hardware

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

***Abstract*— Emerging semantic search techniques require fast comparison of large "concept trees". This paper addresses the challenges involved in fast computation of similarity between two large concept trees using a CUDA-enabled GPGPU co-processor. We propose efficient techniques for the same using fast hash computations, membership tests using Bloom Filters and parallel reduction. We show how a CUDA-enabled mass produced GPU can form the core of a semantic comparator for better semantic search. We experiment run-time, power and energy consumed for similarity computation on two platforms: (1) traditional sever class Intel x86 processor (2) CUDA enabled graphics hardware. Results show 4x speedup with 78% overall energy reduction over sequential processing approaches. Our design can significantly reduce the number of servers required in a distributed search engine data center and can bring an order of magnitude reduction in energy consumption, operational costs and floor area.**

Keywords: Semantic Comparator, GPGPU, Dot product Computation, semantic-router core, Bloom filter, green computing

I. INTRODUCTION

As the web continues to grow, users demand more sophisticated meaning based (semantic) search capabilities that go beyond string matching based searches[1]. For example, they expect that a search for "work-life balance" (the query) should retrieve documents/web pages related to "time management", "exercise", "nutrition" etc., even though the user's query and the returned documents/pages may not share any common keywords.

Semantic search techniques developed to address this problem involve large scale intensive computations. In order to meet the growing web (~tens of billion webpages/documents) and users demand for more relevant search results (9 billion queries per month or 3500 per second[2]), search indices (search engine cores) are deployed as a large distributed system in data centers[3].

Consequently, a search engine's data centers will need to process more data per clock cycle at each compute node to maintain current performance. As stated in [4], parallelization within a compute node is necessary. Traditional Web search engines currently exploit increased parallelism by adding more computers (coarse-grained task-parallel approach) only.

In this paper, we propose a CPU-GPU Co-Processor hybrid architecture for compute nodes in a semantic search engines. We show that these hybrid nodes will reduce search latency while conserving energy. Our key contributions are: (1) A 4-stage GPU Kernel algorithm which provides a 4x speedup over sequential approaches, (2) 78% average energy reduction per query resolution and (3) Demonstrating scalability over a workload consisting of up to 150k entries.

The rest of the paper is organized as follows. In Section II, we analyze existing search engine compute models, introduce the intuition behind reorganizing a search engine's index using semantic routing, identify scalability bottlenecks, and find a fit for the proposed semantic comparator core. In Section III, we detail the 4-stage comparator architecture. Section IV outlines the experimental setup for execution time and power measurement of the 4-stage architecture. We present and analyze performance characteristics in Section V, contrast with prior art in Section VI and conclude in Section VII.

II. PRELIMINARY INFORMATION

In this section, we analyze existing search engine architectures (Sec. II.A), introduce the concept of semantic routing (Sec.II.B), introduce and describe the challenges involved in building a comparator for semantic search engines (Sec. II.C)

A. Analysis of Existing Search Engines

The following section provides an overview of state-of-the-art in large scale information retrieval systems from published literature[3, 5]. We parameterize these systems and estimate the number of (compute nodes) servers necessary to support a given response time. A large scale information retrieval system must balance engineering tradeoffs[6] between (1) number of documents indexed, (2) number of queries/sec, (3)index freshness/update rate, (4) query latency, (5) meta-data stored regarding each document and more significantly (6) complexity/cost of scoring/retrieval algorithms. In this paper, we focus exclusively on (6) as it is a key bottleneck today.

1) Architectural Overview:

Figure 1 illustrates the architecture of a typical internet search engine core[3]. It is assumed that every document indexed by a search engine is assigned a unique *docid*. A search engine consists of two core components (A) Index Servers and (B) Doc Servers. Users send raw queries *q* (at rate Q) to the front-end server. A query processor multicasts these queries to Ns Index shards (rate reduced to Q/Ns) which constitute the index server. For a given query *q*, index shards return a sorted list of (*similarity_score*, *docid*). Index shards are replicated for capacity (often across geographical locations for

fault-tolerance). For example multiple instances of I_k (shown in Figure 1) form a pool. Given a set of *docid*s for a given query, a document processor returns the relevant (*URL,snippet*) which is returned to the user as a result by the front-end service.

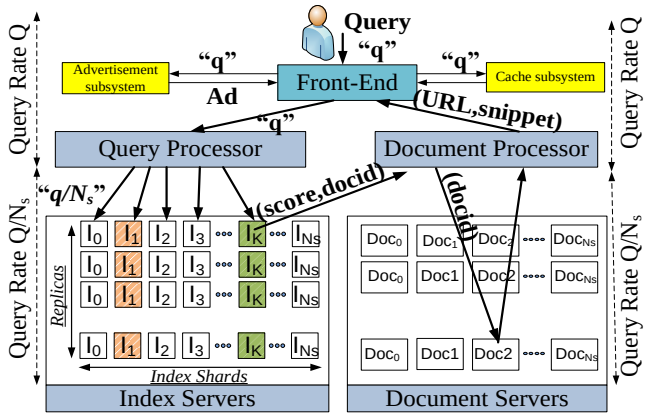

Figure 1. Index distribution and query delivery in a typical search engine (simplified view)

The index is typically generated (using statistical measures like TF-IDF[7] or latent semantic dimensions [8]) with rows representing words/dimensions and several columns representing related *docid*s and their corresponding weights ($W_{i,j}$) (Figure 2). A Jaccard similarity (cosine-product) is computed across the n-dimensions of every index shard. The resulting similarity scores are sorted and further processed by the document servers to return *(URL,snippet)* to the user.

Rows \ Columns (Documents)			
Dimension-1	$DocId_1, W_{1,1}$	$DocId_1, W_{1,2}$	….
Dimension-2	$DocId_2, W_{2,1}$	$DocId_2, W_{2,3}$	
			….
Dimension-n	….	….	….

Figure 2. Simplified inverted index in a typical search engine

2) Parameterzing a Search Engine infrastructure:

In this section, we model a large-scale information retrieval system and estimate the number of servers required to guarantee a given response time. We assume that objects/doc ids and user queries are uniformly distributed across all shards and document corpus respectively. User queries arrive at a rate Q per second to the front-end, and they are consequently multicast by the query processor to all N_s number of index shard pools (Fig. 1). N_p is defined as the capacity of each server to index d documents from among the total number of indexed objects/documents D in the index. Clearly,

$$N_p = D/d \quad (1)$$

The query arrival rate $q_{arrival}$ at each server within each pool depends on the number of servers N_S in each pool. The number of servers needed in each pool is given by the index look up (or query serving) capacity $q_{capacity}$ of each server. As $q_{capacity}$ should match $q_{arrival}$, therefore number of servers N_S shards needed (for each pool) is:

$$N_s = Q/q_{arrival} = Q/q_{capacity} \quad (2)$$

Therefore, the total number of $N_{servers}$ servers needed in the index server pools is:

$$N_{servers} = N_p \times N_s = (D/d)\times(Q/q_{capacity}) \quad (3)$$

3) Estimating the parameters:

A typical search engine like Google[9] claims to have indexed >26 billion web pages/documents out of unique 1 trillion issued URLs, so we assume $D \approx 2.6*10^{10}$. Latent semantic indexing[10] (the best known technique) enables object representation in 1000 or fewer dimensions. Each object (column) and dimension (row) in Figure 2 can be represented using 2 Bytes (the weight coefficient for a matrix cell) to get a compute accuracy of up to 4 decimal places. So for the d rows in Figure 2, the memory foot print is 2KB. Each server (having 2GB RAM[6] or 200GB hard disk) can host (2GB/2KB≈) 10^6 or (200GB/2KB≈) 10^8 entries respectively. Therefore each server can host up to $d \approx 10^6$ or 10^8 number of entries. This indicates that the design parameter $N_P = D/d \approx 2.6*10^4$ or $2.6*10^2$ depending on whether RAM or hard-disk is used for lookup. From[5] we also know that average number of queries $Q \approx 3500$/s. Therefore using (3), we can compute the number total number of servers in use $N_{servers} = N_p \times N_s = 10^3 \times 3500/15 \approx 233{,}333$. If we assume that number of servers in the query processor and object/doc server pools are of similar order (i.e. each system consists of 233,333 servers) because by design they all encounter similar traffic rate, then total number of compute nodes that a typical search engine (Google, Bing, Yahoo) will have to deploy is at least $233{,}333 \times 3 \approx 700{,}000$ (all data centers combined).

B. Semantic search engines – A semantically routed index

New methods to represent "composite" meaning in computers have been designed and proven to be superior to TF-IDF in [11]. This enables conjunction, disambiguation & representation of "complex concepts" their synonyms and hyponyms using a Tensor-based transformation model. The use of this model enables the creation of a Semantic Routed Network[12] (SRN) in the index shards of a search engine. A SRN has been proven to enable the automatic reorganization of the search-index of a distributed search engine based on the principles of a Small World Network[12, 13]. Using this model, an incoming query q can be injected to any random index server shards (selectively unicast) instead of the inefficient multicast mechanism discussed in Sec. II.A. The Index network routes the query until resolved. Figure 3 shows the reorganization of the Index shards using the principles of SRN. A query q is shown injected any one of the Index shards I_1. Since, I_1 does not have an answer it passes the query on to the next "best" known node (I_3 which passes it along to I_{Ns}). Results in [12, 14] show through simulation that convergence to a Small World Network is possible, with bounded overhead. It is proven that query resolution is guaranteed within a maximum of 3 hops from injection on average (for a fully re-organized index). The number of queries which need to be computed at each shard (functioning as a compute node) is reduced from Q/Ns to Q* (<Q/Ns<<Q).

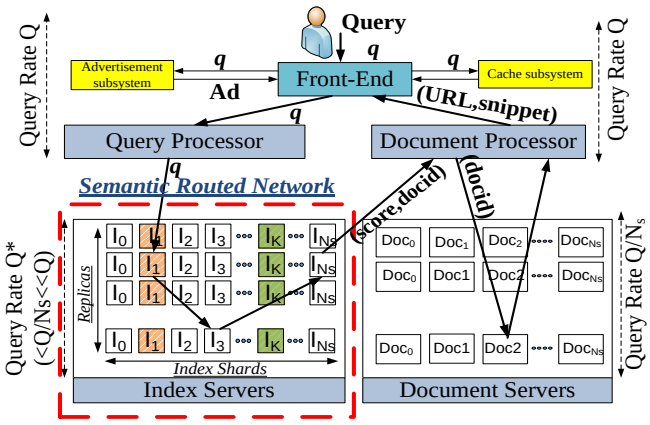

Figure 3. A Search engine involving a Semantic Routed Network will reduce Query Rate to Index Shards

These techniques address users need for "semantically" meaningful and relevant results without an increase in the response time (real-time search). Although an SRN has been proven to theoretically converge (in simulation), our experiments in the next Section emphasize the need for hardware in the index shards to meet the cost of a significantly larger computational load per query resolution using a Tensor model and SRN-based index server pool.

C. Scalability Challenges for a Semantic Comparator

Meaning comparison using the techniques in [11, 13, 14] has been reduced to a sequence of lookup-match-multiply-sum operations on "semantic descriptors". This "semantic comparison" is to be performed by all the index shard servers during query resolution and re-organization in a SRN.

A semantic descriptor can be generated in several ways. The most commonly used technique TF-IDF[15] stores a statistical product term (frequency × inverse document frequency) for all terms in a document. In contrast [11] proposes and evaluates a weighted Concept (ontology) Tree based descriptor taking compositions into account.

A concept tree is a hierarchical acyclic directed n-ary tree where the leaf nodes represent terms whereas the tree itself describes their inter-relationships within a document. Each term is assigned a weight which describes its relative importance. Figure 4 shows two sample trees which represent two distinct "concepts" but use the same keywords (shown as leaf nodes). The intermediate notations (ex. {American, man}) are notional and are shown for convenience.

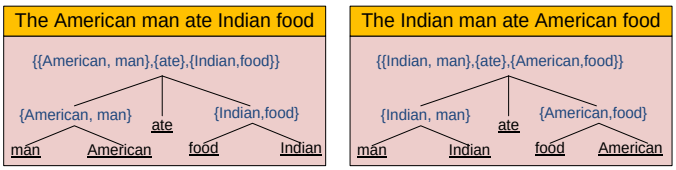

Figure 4. Generating concept trees for two statements. They have same keywords but convey different "meaning". Term based comparison fails

Associated weights for each term (coefficient values describe their relative importance) are not shown for simplicity. Concept tree building has been defined earlier[12] and applied to test cases in [16]. It is important to note that conventional term based weighting cannot differentiate between the two statements (or their trees). A Tensor model, on the other hand, is used to decompose such concept tree into a flat data structure consisting of polyadic concept terms and their normalized coefficients without loss of the semantics contained in the original tree structure.

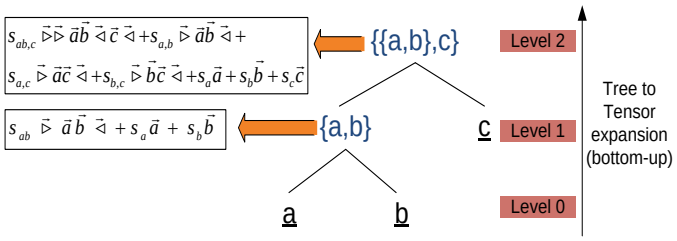

Figure 5. Conversion of a concept tree into its polyadic tensor representation

Figure 5 shows the bottom up expansion of a 3-level, 2 child concept tree. Level 0 shows leaf nodes "a" and "b". Their composition at Level 1{a,b} is defined by ▷ab◁,a and b (▷,◁ are delimiters; s_{ab}, s_a, s_b are normalized weights). The final tensor representation of this concept tree is obtained at Level 2 consisting of weighted polyadic combinations of terms ▷ab◁c, ▷ab◁,▷ac◁,▷bc◁, a, b, c (called basis vectors) and their normalized weights. The nuances of this conversion including the process of determination of weights is detailed in [11, 17].

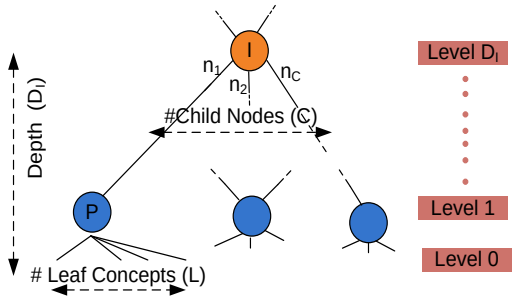

Figure 6. Computing the number of basis vector terms in a concept tree

Figure 6 shows a section of a concept tree. Assume that an intermediate node I contains "C" child nodes (n_1, n_2, … n_C). One of these child nodes P contains "L" leaf terms. The maximum depth of any sub-tree of I is D_I. The number of terms in the tensor representation at Level D_I due to all permutations of leaf terms at P will be $\sum_L \binom{L}{i} = 2^L\text{-}1$. At level D_I, the C sub-trees will produce $\binom{n_1}{1} + \binom{n_2}{1} + .. + \binom{n_1}{1}\binom{n_2}{1}..\binom{n_C}{1} = 2^{n_1+n_2+..+n_C} - 1$ terms. We developed a Java based concept tree to tensor conversion application to simulate this process. Representative results are shown in Table 1. The symbols {..} delimit sub-trees at the same level. Whereas a conventional term-based technique (like TF-IDF) would have needed to index only the leaf terms (Column A), the tensor model needs to index an exponentially larger number of terms (Column B).

Figure 7 outlines the overall sequence of steps necessary for a semantic comparison using this model. An incoming query "q" & crawled documents within a corpus are converted from concept tree to tensor forms as described earlier producing a

tabular coefficient table containing n_1 *Query_Basis* and n_2 *Doc_Basis* terms respectively.

TABLE I. COMPARING SEMANTIC DESCRIPTOR TERMS DUE TO [7] & [11]. ENHANCED MEANING REPRESENTATION TECHNIQUES REQUIRE EXPONENTIALLY LARGER NUMBER OF BASIS VECTOR TERMS

Concept Tree Notation	**Number of Leaf terms**	**Number of Basis Vectors terms**
	TF-IDF[7] indexing (A)	*Tensor Model[11] indexing (B)*
{{a,b},c}	3	7
{{a,b},{c,d},e}	5	31
{{{a,b,c},{d,e,f},{g,h}}}	8	255
{a,b,c},{d,e,f},{g,h,i}	9	511

To determine a similarity value; the "k" common terms need to be identified (membership test) and their corresponding coefficients multiplied to yield interim products. The final stage in processing query "q" is to sum these *k* interim products to yield a similarity value between 0 and 1.

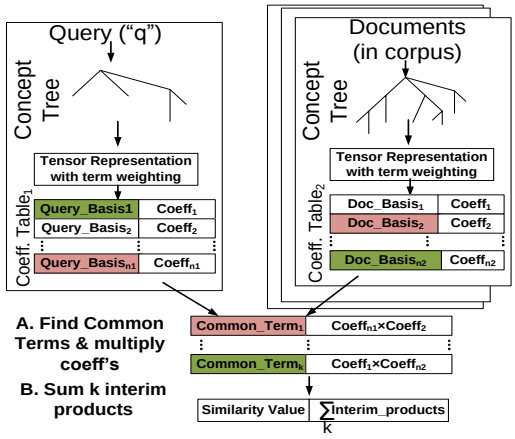

Figure 7. Shows the sequence of operations required in generating a similarity value for an incoming query "q" from documents in a corpus.

III. GPU-BASED SEMANTIC COMPARATOR

Recently, hardware manufacturers such as NVIDIA[18] and AMD[19] have begun shipping graphics cards that contain numerous processing core that can perform parallel computation. These devices contain 100s of lightweight Single-Instruction-Multiple-Data streaming processors that can perform simple computations in a highly multi-threaded manner. Interest has been growing from the IR community towards harnessing the power of these cards for computational jobs that can be done in parallel given that the number of GPU cores/card is growing dramatically; they are virtualized; provide higher memory bandwidth; have well-defined programming frameworks.

We propose using these GPGPUs to form the core of a semantic comparator since they provide the middle ground between traditional multi-core processors (such as the Intel Xeon) which are sequential and the very fast (fine-grained parallelism) but expensive ASIC's[20]. Figure 7 shows a simple control flow amenable to massive parallelization. Suppose the number of basis vector terms in the two coefficient tables to be compared is denoted by n_1 & n_2 ($n_2 << n_1$). When a sequential processors is used (conventionally), this search task has a time complexity of $O(n_1 \log n_2)$ or $O(n_1 \times n_2)$ per core depending on whether a binary or linear search is used respectively. We will distribute the comparison of each of the n_2 terms to a thread reducing the complexity to $O(n_1)$. To further reduce complexity to O(1), we use Bloom Filters as a key component in the co-processor kernels.

A. Bloom Filtering in a Semantic Comparator Kernel

A Bloom filter (BF) is a compact representation of a set[21]. A BF is defined as a large single dimensional bit array ($size_n$ bits) and a set of *k* hash functions to encode it. Consider we have *m* items: $Item_1$, $Item_2$,….,$Item_m$ randomly distributed in two sets A & B. A Bloom filter enables us to determine the items in A∩B in O(1). To do so, all elements of Set A are encoded in the Bloom Filter. Elements in Set B are tested against the BF. We use two hash functions[21] instead of the typical *k* in literature. Figure 8 shows two sets containing *m* unknown elements, a set of mathematical operations to generate *k* BF Indices (BFI) for each element and the $size_n$ long Bloom Filter.

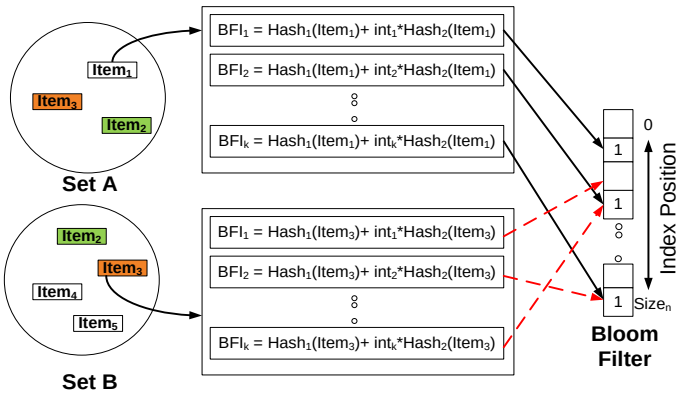

Figure 8. Set Intersection using a Bloom Filter. k indices (BFI) are generated for each element in Set A. BFI's from elements in Set B can be tested in parallel with the resulting Bloom Filter to return Present/Absent result

1) *Inserting an element into a BF*

To insert an item (a number or a text string, say item1) into a BF, we generate its k indices, the k^{th} index using:

$$BFI_k = Hash_1(Item) + int_k \times Hash_2(Item) \quad (4)$$

For the rest of this paper, we choose *Hash1*=FNVHash and *Hash2*=JSHash because of their computational simplicity. int_k is any random integer value. The calculated BFI_k's represent BF bit-vector positions which are activated (turned "1").

2) *Testing an element in a BF*

An element to be tested (say $item_3$ in Figure 8) undergoes the same treatment as above except that the computed BFI's are used to "test" the Bloom filter. All *k* BFI bits being "1" indicates that the test element belongs to the intersection set.

The test of whether an arbitrary element is in the BF can result in false positives (returns true even when the element is not present), but not false negatives (will never return false

when the element is present). The probability of false positives ($p_{false+ve}$) is given by:

$$P_{false+ve} = \left(1 - \left[1 - \frac{1}{size_n}\right]^{km}\right)^k \approx \left(1 - e^{\frac{-km}{size_n}}\right)^k \quad (5)$$

$p_{false+ve}$ can be minimized by choosing large $size_n$, and optimum number of BFI's k ~0.7 × $size_n$/m)[21]. In all experiments that follow, we choose these properties appropriately to minimize $P_{false+ve}$.

B. Semantic Comparator Kernel Algorithm

This section outlines 4 phases of our algorithm to compute the similarity value between two semantic descriptor (coefficient) tables. Section II.B showed that this computation is in the critical path and is particularly challenging given the exponential growth of the number of Basis Vector terms using the Tensor Model (Table I). The rest of this section describes each phase in detail.

C. Phase A: Host-PC →CUDA Global Memory Copy

Coefficient Tables 1 & 2 are copied to CUDA Global Memory. To enable maximize bandwidth we flatten the data structure internally to ensure coalesced memory accesses. This process is described in Figure 9 as Steps 1a and 1b. The transformations ensure maximization of PCIe bandwidth.

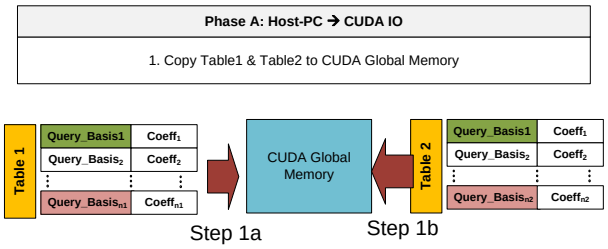

Figure 9. Phase A: Copying Semantic Coefficient Tables into CUDA Global Memory

D. Phase B: Encode Table1 in BF

Figure 10 outlines the procedure used in every kernel call to encode a given $Query\_Basis_i$ into the Bloom Filter. This follows from the discussion in Section II.C. We make at least n_1 concurrent kernel calls (independent threads) so that each row of Table1 is served by at least one CUDA thread. The CUDA occupancy calculator provided by NVIDIA as part of its CUDA toolkit was used to ensure that each multiprocessor has a sufficient number of free registers to prevent blocking.

FNV & JS Hashes are implemented as device functions. They produce 2 hash values (64-bit integers) for each $Query\_Basis_i$ term they operate on. These hash values are then used to compute k different bloom filter index values as described in Equation 4. CUDA texture memory is used to hold the Bloom Filter (defined as a $size_n$ long bit-array). Bloom filter in texture memory avoids latency in memory access among concurrently running kernels.

E. Phase C: Encode Table2 and test in BF

This phase (Figure 11) mirrors Phase B with two differences;(1) Instead of setting a Bloom Filter, it tests for presence/absence of $Query\_Basis_i$ term the kernel is operating on.(2) If bits indicated by the BFI's for $Query\_Basis_i$ are "1" in the Bloom Filter, the corresponding index i is stored in shared memory for use in the next phase. Every kernel call encodes a row of Table2, tests with the previously encoded BF (Phase B) and stores the index values of "matches found". We ensure that at least n_2 concurrent kernels are called in this phase.

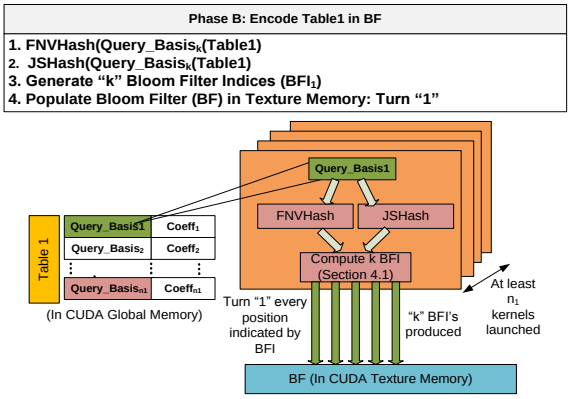

Figure 10. Phase B: n_1 concurrent kernel calls to encode elements of Table1 in a Bloom Filter (in CUDA Texture Memory)

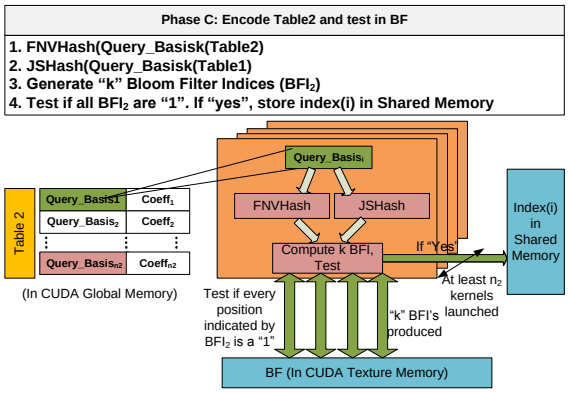

Figure 11. Phase C: n_2 concurrent kernel calls to encode & test elements of Table2 from BF

F. Phase-D: Extract Corresponding Scalar Coefficients and parallel reduction

This is the final phase of the algorithm (Figure 12). Phases A-C enabled the "filtering" of the common basis vectors with O(1) complexity. The "filtered" common basis vector terms are loaded from shared memory (Step1) and the corresponding $Coeff_2$ value is fetched from Global Memory (Step 2). In step 2, the corresponding $Query\_Basis_i$(Table2) is located in Table 1. This is a computationally intensive process but occurs only for a small subset of "filtered" elements. Once the corresponding coefficients ($Coeff_1$ and $Coeff_2$) have been obtained, a partial product is calculated (Step 4). Step 5 of this phase involves parallel reduction of the interim products from each kernel to sum and is performed in accordance with the guidelines in [22].

IV. EXPERIMENTAL SETUP

The goal of the paper is to investigate the feasibility, performance and power consumption of semantic dot product

computation deployed as a GPU application kernel with varying (1) input size (2) degree of similarity. All experiments were carried out on the same workstation.

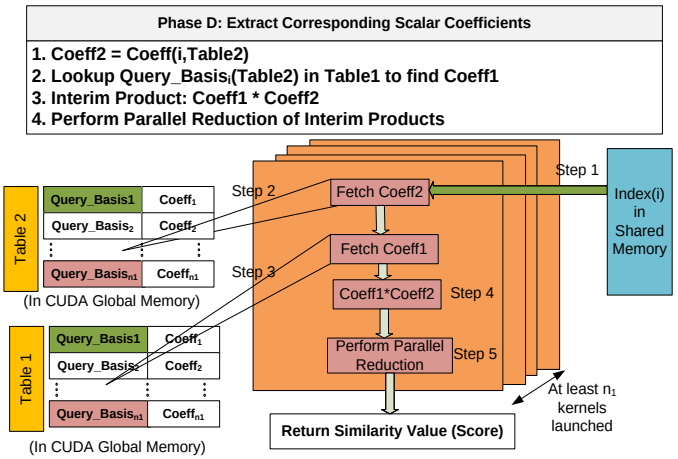

Figure 12. Phase D: Extract coefficients, generate interim products, and perform parallel reduction

The GPU card is an NVIDA Tesla C870 (Compute Capability 1) and uses 65nm technology. We achieve rated bus memory bandwidth of 76.8 GB/s (384 bit wide, GDDR3) during data transfer for all experiments. The GPU contains 16 stream processors with a total of 128 cores each running at 600 MHz. The card has 1.56 GiB of RAM running at 1.6 GHz. The interface to the host PC is over a 16x PCI-Express bus. CUDA Toolkit version 3.1 was used for compilation. The host machine has a Pentium-4 processor with 2 GB of RAM running Ubuntu 9.10.

A. *Power Profiling*

The power monitoring was done using Watts' Up Pro power analyzer from Electronic Educational devices[23]. This measures overall system power consumption. This device is connected in line with the power supply to the host computer as shown in Figure 13.

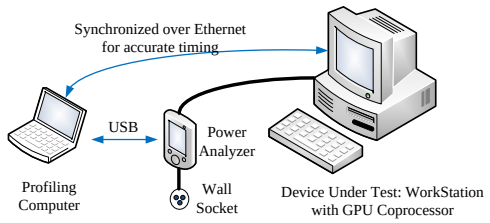

Figure 13. Equipment setup for power profiling

Each experiment executes for at least 10 seconds (multiple iterations used when necessary). In Table II system base power is the static power consumed by the host computer without the GPU present in it. System Idle power is the power consumed with the GPU present but in cold shutdown state. GPU idle power is the power consumed with the GPU awake but not running any specific computation. This is computed by subtracting System Idle Power from power consumed by the system after the GPU is awake but in idle state.

TABLE II. BASELINE POWER FIGURES

System Base Power	115W
System Idle Power (GPU cold shutdown)	150W
System Idle Power (GPU Awake, Idle)	186W
GPU Idle Power	36W

B. *Execution Time Profiling*

We designed a time-accurate simulator which implements the algorithms described in Section III.C-F on the CPU and GPU. The same simulator was also used to estimate the cycle time for the ASIC designs [20, 24]. We use CUDA timers (running on one Stream Processor) and CPU system time to measure the run-time execution time of the core computational kernels and equivalent CPU code respectively. All experiments are run with a synthetic benchmark. For simplicity, we have restricted the number of threads/block to 6.

V. RESULTS

In this section, we present and analyze overall execution time, power, energy, phase-wise execution times and throughput for the semantic comparator core on a CPU and GPU respectively. We have conducted all our experiments with $n_1=n_2=N$. In a real-life search-engine a query-document coefficient tables will have sizes $n_1<<n_2$. Our results represent the worst case situation. We experiment for (1) N varying from 100 to 150000 rows and (2) Similarity c varying between 0.1 and 1(All Match) in the two coefficient tables.

A. *Overall Execution Time*

Figure 14 shows the overall execution time of Phases A-D as outlined in Section III with varying input size of the coefficient tables under experimentation (N). These results are shown for the situation c=0.1 (similarity between simulated tables=10%). Sequential execution performed by the CPU causes an exponential increase in the execution time as the number of entries increases whereas the same operation on the GPU is an order of magnitude faster. .

B. *Overall Power Consumption*

Figure 15 shows the variation of power consumed by the CPU and GPU respectively with varying table size. This experiment was conducted for c=0.5 (50% similarity between table entries). The dynamic power for the GPU is lower than that consumed by the CPU. GPU Power approaches that of the CPU for extremely large datasets (above 50,000 entries). This is a known problem for GPUs[25] – they are energy efficient and not necessarily power efficient.

C. *Energy Consumption*

Energy consumption For 5000<N<150000 and c=0.75 shows the execution time, average power consumption for the CPU and GPU respectively when running Phases A-D for varying sizes of N and c=0.75 (for a single comparison). The average energy saved across varying table sizes is ~78%. These results show that in the long-term a search engine infrastructure using a CPU-GPU hybrid in its compute nodes can (1) reduce its energy footprint or (2) increase its throughput for the same energy footprint.

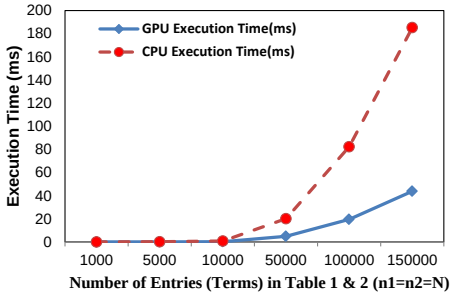

Figure 14. Execution time for Semantic Comparator using GPU Kernel and CPU (~4x speedup provided by a GPU for large tables, c=0.1)

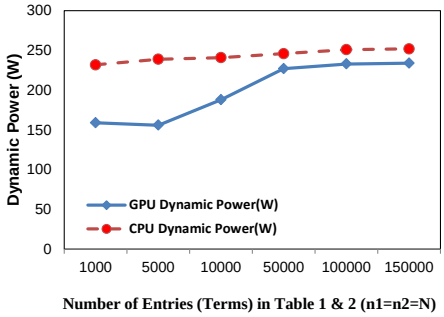

Figure 15. Dynamic Power Consumption (averaged) for Semantic Comparator using GPU Kernel and CPU

TABLE III. ENERGY CONSUMPTION FOR 5000<N<150000 AND C=0.75

Table Size (N)	CPU Exec. Time (s)	CPU Avg. Power (W)	GPU Exec. Time (s)	GPU Avg. Power (W)	Energy Saved (%)
5k	0.18	232	0.05	159	67.59
10k	0.74	239	0.21	156	79.65
50k	20.0	241	4.93	188	77.64
100k	82.4	246	19.57	227	77.27
150k	185.3	251	43.83	233	77.96

D. *Profiling the Kernels (Phase A-D)*

Figure 16 shows a comparative plot of Percentage Execution time for each Phase A-D of our algorithm and for varying table size N. This graph presents the following insights: (1) It conveys counter-intuitively that Phase A (Memcpy from CPU to GPU) ceases to be a bottleneck for extremely large table sizes (5k and above). It peaks at 1k entries and reduces thereafter. Phase D (Extract Corresponding Scalar Coefficients and perform parallel reduction) becomes a bottleneck instead. (2) Phases B and C which perform the hash computation and filtering consume an almost negligible part of the overall execution time. This validates our claim that Bloom Filters are a good candidate for set intersection

We ran experiments at each value of N for randomly varying percentage similarity *c*. This is representative of queries in a real search engine which are diverse. The ensuing execution time showed only small variances (due to the nature of our algorithm).

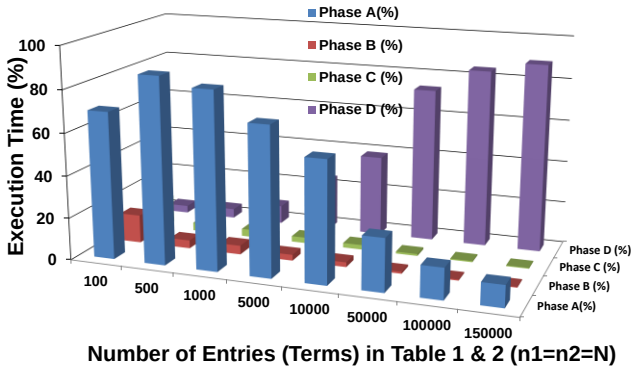

Figure 16. Comparison of percent execution time of the 4 algorithmic phases.

Throughput is defined as the inverse of the averaged execution times for each N. This is reported in Table 3. We see that improvement in GPU throughput is more pronounced at larger table sizes. This is expected because for smaller values of N, the overhead of data transfer from the CPU to GPU dominates. This effect is apparent in Figure 16 for N=100 where ~70% of the overall execution time is taken up in data transfer.

TABLE IV. EXPERIMENTAL THROUGHPUT FOR CPU AND GPU BASED SEMANTIC COMPARATOR WITH RANDOMLY VARYING C

Coeff. Table Size (N)	Sequential Processor (CPU) ($\times 10^3$/s)	Streaming Processor (GPU) ($\times 10^3$/s)	Throughput Gain
5k	53996	173097	3.20
10k	13458	46725	3.47
50k	499	2025	4.05
100k	121	510	4.20
150k	53	228	4.22

VI. RELATED WORK

The most common approach to improving large scale distributed search is by deploying large numbers of commodity servers in massive data centers. This approach is expensive and inefficient as compared to using a CPU-coprocessor hybrid. In [4] authors show that search engine optimizations at the compute node level usually only translates to task parallelism. We cannot compare directly against their model because they are using a traditional inverted index unlike our scheme. GPUminer[26] is the closest parallel data mining system to semantic search. This has been deployed as a CPU-GPU hybrid where the CPU performs storage and I/O management whereas GPU performs parallel mining/visualization. This does not form a valid comparison because the authors have chosen to deploy only k-means clustering and *Apriori* frequent pattern mining algorithms. In [24] & [20] the authors proposed a fine grained parallel ASIC that performs the dot-product computation for a semantic comparator. While the ASIC targeted design is an order of magnitude faster and consumes extremely low power, this design requires additional components (I/O) before it can be integrated into an existing system. A GPU based architecture on the other hand handles much larger data sizes (N=150000 v/s N=1024).

VII. CONCLUSIONS

Search continues to be a much sought after application on the World Wide Web. It is estimated that large internet search engine datacenters consume tens of Megawatts[27]. It is clear that conventional search engines are not scalable for semantically enabled search that users are looking for. It is important to look within each compute node and exploit all available parallelism.

GPU based accelerators are fast becoming popular and mainstream due to their extra-ordinary energy efficiency, ease of use and reducing costs. The reported speedup will translate to higher gains in datacenters where dot-product computation is a bottleneck.

In this paper, we presented a CUDA based architecture that performs semantic comparison for extremely large coefficient tables (~150k entries). We showed that our design provides a reasonable speedup (~4x) and massive energy savings (~78%) compared to conventional sequential approaches in vogue today. To the best of our knowledge this is the first work of its kind which handles semantic coefficient tables of this size. This paper addresses the challenges in building the core of a semantic router using CUDA enabled graphics hardware.

VIII. REFERENCES

[1] J. C. Perez. (2009). *Google joins the crowd, adds semantic search capabilities, ComputerWorld*. Available: http://www.computerworld.com/s/article/9130318/Google_joins_crowd_adds_semantic_search_capabilities_

[2] A. Patriquin. (2009, 2009). *March Search Market Share:Record query growth and the Yahoo/Microsoft search deal by the numbers,Compete Inc.*

[3] L. A. Barroso, J. Dean, and U. Holzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro*, vol. 23, pp. 22-28, 2003.

[4] E. Frachtenberg, "Reducing Query Latencies in Web Search Using Fine-Grained Parallelism," *World Wide Web*, vol. 12, pp. 441-460, 2009.

[5] J. Dean, "Challenges in building large-scale information retrieval systems: invited talk," presented at the Second ACM International Conference on Web Search and Data Mining(WSDM) 2009 Barcelona, Spain, 2009.

[6] J. Dean, "Designs, lessons and advice from building large distributed systems," presented at the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, Big Sky, MT, 2009.

[7] G. Salton and C. Buckley, "Term Weighting Approaches in Automatic Text Retrieval," Cornell University1987.

[8] B. Rosario, "Latent semantic indexing: An overview," *Techn. rep. INFOSYS*, vol. 240, 2000.

[9] J. Alpert and N. Hajaj. (2008). *We knew the web was big*. Available: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

[10] S. T. Dumais, "Latent semantic analysis," *Annual review of information science and technology*, vol. 38, pp. 188-230, 2004.

[11] A. Biswas, S. Mohan, A. Tripathy, J. Panigrahy, and R. Mahapatra, "Semantic Key for Meaning Based Searching," presented at the Proceedings of the 2009 IEEE International Conference on Semantic Computing, Washington, DC, USA, 2009.

[12] A. Biswas, S. Mohan, and R. Mahapatra, "Search Co-Ordination by Semantic Routed Network," presented at the 18th International Conference on Computer Communications and Networks (ICCCN), San Francisco, 2009.

[13] A. Biswas, S. Mohan, J. Panigrahy, A. Tripathy, and R. Mahapatra, "Representation of complex concepts for semantic routed network," presented at the International Conference on Distributed Computing and Networking (ICDCN), 2009.

[14] A. Biswas, S. Mohan, and R. Mahapatra, "Optimization of Semantic Routing Table," presented at the Proceedings of 17th International Conference on Computer Communications and Networks, 2008. ICCCN '08. , St. Thomas, US Virgin Islands 2008.

[15] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, vol. 24, pp. 513-523, 1988.

[16] J. Mitchell and M. Lapata, "Vector-based Models of Semantic Composition," in *Proceedings of ACL-08: HLT*, 2008, pp. 236-244.

[17] J. Panigrahy, "Generating Tensor Representation from Concept Tree in meaning based search," Master of Science, Computer Science & Engineering, Texas A&M University, College Station, 2010.

[18] NVIDIA. (2010). *NVIDIA Fermi Architecture*. Available: http://www.nvidia.com/object/fermi_architecture.html

[19] AMD. (2010). *AMD ATI Stream*. Available: http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/Pages/stream-technology.aspx.

[20] S. Mohan, A. Tripathy, A. Biswas, and R. Mahapatra, "Parallel Processor Core for Semantic Search Engines," presented at the Workshop on Large-Scale Parallel Processing (LSPP) to be held at the IEEE International Parallel and Distributed Processing Symposium (IPDPS'11), Anchorage, Alaska, USA, 2011.

[21] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, pp. 485-509, 2004.

[22] M. Harris. (2010). *Optimizing parallel reduction in CUDA, NVIDIA Developer Technology*.

[23] (2009). *Electronic Educational Devices, Watts up? Pro*. Available: http://www.wattsupmeters.com/.

[24] S. Mohan, A. Biswas, A. Tripathy, J. Pannigrahy, and R. Mahapatra, "A parallel architecture for meaning comparison," presented at the Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on, Atlanta, GA 2010.

[25] Y. Jiao, H. Lin, P. Balaji, and W. Feng, "Power and Performance Characterization of Computational Kernels on the GPU," presented at the 2010 IEEE/ACM Green Computing and Communications (GreenCom), Hangzhou, China, 2010.

[26] W. Fang, K. K. Lau, M. Lu, X. Xiao, C. K. Lam, P. Y. Yang, B. He, Q. Luo, P. V. Sander, and K. Yang, "Parallel Data Mining on Graphics Processors " *Technical Report HKUST-CS08-07*, 2008.

[27] EDL. (2007). *Data centers meeting todays demand,National Electrical Contractors Association (NECA), Tech Report*. Available: www.necanet.org/files/ACF41A4.pdf