Parallelizing Latent Semantic Indexing Using an FPGA-based Architecture

Xinying Wang and Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa, USA
Email: {xinying, zambreno}@iastate.edu

***Abstract*—Latent Semantic Indexing (LSI) has played a significant role in discovering patterns on the relationships between query terms and unstructured documents. However, the inherent characteristics of complex matrix factorization in LSI make it difficult to meet stringent performance requirements. In this paper, we present a deeply pipelined reconfigurable architecture for LSI, which parallelizes the matrix factorization and dimensionality reduction, computation of cosine similarity between vectors, and the ranking of documents. Our architecture implements the reduced Singular Value Decomposition with Hestenes-Jacobi algorithm, in which both singular values and orthogonal vectors are collected, and its components can be reconfigured to update query vector coordinate and calculate query-document similarity. In addition, an ordered tree structure is used to reduce the matrix dimension and rank the documents. Analysis of our design indicates the potential to achieve a performance of 8.9 GFLOPS with dimension-dependent speedups over an optimized software implementation that range from $3.8\times$ to $10.1\times$ in terms of computation time.**

I. Introduction

In many scientific and engineering applications (e.g., text processing [1], information retrieval [2], and bioinformatics [3]) , Latent Semantic Indexing (LSI) has been widely used as an information analysis technique to identify the relationships between the query terms and the content of unstructured documents. LSI commonly implements a linear factorization tool named reduced Singular Value Decomposition (rSVD) at its core, followed by ranking process according to the cosine similarity between query terms and documents. LSI is considered computationally expensive, and in many applications, sequential LSI implementations are unlikely to satisfy requirements of querying semantical retrieval over millions of documents in a few milliseconds [4].

FPGAs have shown the promise to provide fine-grained parallelism, and are more compatible with highly data-dependent transformations in computing SVD, compared to other hardware accelerators. Previous FPGA-based implementations of Latent semantic processing [4]–[7] reformulated the computationally intensive portion of Latent semantic processing as matrix or vector operations (e.g., multiplication, addition). Although speedups are achieved, the reconfigurability and flexibility of FPGAs have shown potentials to further improve the performance by parallelizing all phases of Latent semantic processing, including SVD computation, the ranking process, and vector coordinate updates.

This work is supported in part by the National Science Foundation (NSF) under awards CNS-1116810 and CCF-1149539.

In this paper, we propose an FPGA-based architecture to accelerate LSI, which parallelizes the SVD computation, vector coordinate updates, cosine similarity calculation, and the process of ranking selected documents in an order of the calculated query-document cosine similarity. Our deeply pipelined reconfigurable architecture implements the reduced SVD with Hestenes-Jacobi algorithm [8] that collects both the singular values and orthogonal vectors in the reduced k-dimensional space. In addition, the individual components of our architecture can be dynamically configured to update the query vector coordinates and calculate query-document cosine similarity. Also, we utilize an ordered tree structure to perform dimensionality reduction and rank the documents. Analysis of our design indicates the potential to achieve a performance of 8.9 GFLOPS with dimensional dependent speedups over an optimized software implementation that range from $3.8\times$ to $10.1\times$ in terms of computation time.

II. Theoretical Background

A. The Process of Latent Semantic Indexing

Latent Semantic Indexing is a mathematical technique to analyze the correlation between query terms and a collection of documents. Traditionally, information retrieval is processed through lexically matching query terms with concepts in documents. However, its accuracy is impaired by synonymy that a given concept can be expressed in multiple ways, and polysemy that a word is able to convey many different meanings [9]. To solve this problem, LSI introduces a method to retrieve information through matching the context of query terms and documents. The documents are ranked by query-document cosine similarity, which has no direct relationship with the number of shared terms.

To perform LSI, the documents and query terms are modeled by term-document matrix D and query vector q. D is an $m \times n$ matrix and q is a vector with a length of m, where m and n are the number of selected terms (key words) and documents, respectively.

To reduce the dimensions of the semantic space, the reduced Singular Value Decomposition (rSVD) is performed on the term-document matrix D in the form given by eq. (1)

$$D_{m\times n} \approx U_{m\times k}\Sigma_{k\times k}V^T_{k\times n} \qquad k < \min(m,n) \qquad (1)$$

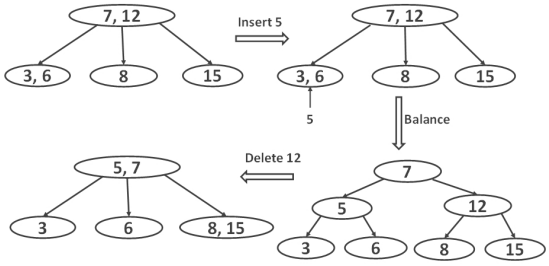

Fig. 1: 2-3 Search Tree Structure and Operations.

where U and V are orthogonal matrices and Σ is a diagonal matrix with singular values as its diagonal elements. Matrix V contains n documents coordinates in the reduced k-dimensional space. After rSVD, query vector q is updated to the new coordinates in the reduced k-dimensional space in the form of eq. (2).

$$q' = q^T U \Sigma^{-1} \tag{2}$$

The query-document cosine similarity between query vector and every document vector in matrix V in the reduced k-dimensional space is calculated in the form given by eq. (3)

$$sim(q, d_i) = \frac{q \cdot d_i}{\|q\| \cdot \|d_i\|} \tag{3}$$

where the values of $sim(q, d_i)$ are used to rank the documents for their association with query terms. A high cosine similarity indicates the close relationship between query term and document.

B. 2-3 Tree Structure

Sorting is necessary to reduce the dimension and rank the documents based on the calculated cosine similarity. In our design, we select the $2-3$ tree data structure (see Fig. 1) to sort the documents since it is considered as a self-balanced search tree that can be easily parallelized [10].

The $2-3$ tree either has one element with two children or two elements with three children attached [10]. When an internal node has two elements p and q, the elements in its left and right children nodes are smaller and greater than both p and q respectively, while its middle children node contains the elements with the values between p and q ($p \leq q$).

III. Related Work

FPGAs have been widely investigated in accelerating Latent Semantic Analysis (LSA). Majumdar et al. proposed an FPGA-based accelerator for supervised Semantic Indexing [4], in which an FPGA provides a solution to parallelize a huge amount of dot products with fine granularity. Eick et al. [5] use an FPGA to accelerate Latent Semantic processing by mapping three compute-bound operations onto highly parallel platform. To improve the scalability, a parallel programmable learning and classification accelerator was presented by Cadambi et al. [6], which uses on-chip memory for intermediate data, and banked off-chip memory with independent processing elements group assigned. Graf et al. [7] implemented arrays of variable-resolution arithmetic vector processing elements (VPEs) on FPGAs to accelerate a learning process.

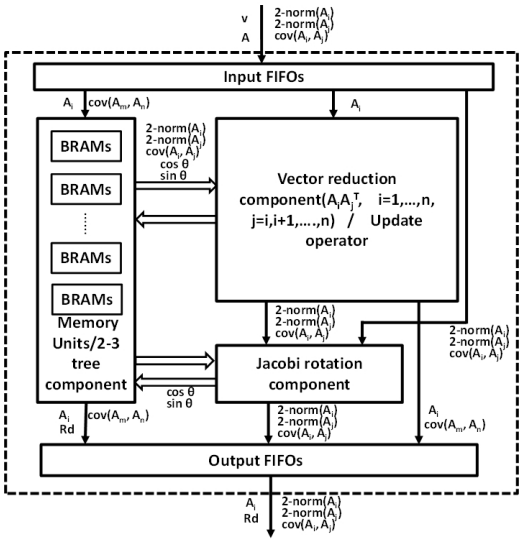

Fig. 2: The proposed architecture for Latent Semantic Indexing.

IV. Proposed Architecture

LSI primarily consists of four computational or logic operations: (1) calculating the squared norms of vectors and the covariances between vector pairs for the SVD and cosine similarity computation; (2) conducting Jacobi rotations with paired squared norms and their respective covariances; (3) updating the matrix elements, the newly generated right orthogonal matrix elements, and the covariances affected by rotation; (4) sorting the singular values and the calculated cosine similarity for dimensionality reduction and the final output respectively.

In our architecture for LSI (see Fig. 2), we created four fully pipelined components: the *Vector reduction component*, the *Jacobi rotation component*, the *Update component*, and the *2-3 tree sorting component*, in which the Vector reduction component and the Update component reuse the same computational resources. The Vector reduction component is responsible for the computation of squared vector norms and associated covariances in SVD and the cosine similarity computing afterwards. The Jacobi rotation component is used to perform plane rotation with squared vector norms to annihilate their related vector covariance. The update component is employed to update the elements affected by rotations. The final result of this architecture is produced by the $2-3$ tree sorting component, which sorts the document vectors according to their cosine similarity with query vector.

A. Vector reduction component

The Vector reduction component extends the Hestenes Preprocessor from previous work [8] to compute vector dot products for the norms and covariances computing, the query vector coordinates updates, and cosine similarity calculation. The

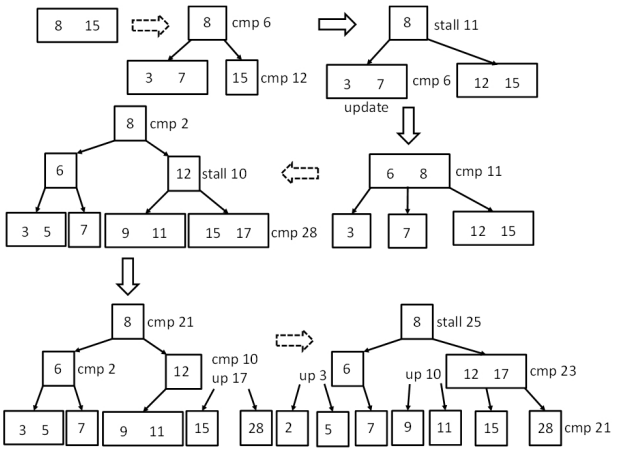

Fig. 3: Parallel process of new element insertion and 2-3 tree structure updates.

Vector reduction component compute squared column 2-norms and covariance between column vectors through $A_i^T * A_j$ with a design of multiple layers of pipelined multiplier-arrays. In this design, a multiplier-array is responsible for calculating the partial results of different squared norms and their related covariances, and operands are reused by all the multipliers in a pipelined manner. The "reduce" process is performed through summing up the calculated product of a multiplier with the results of its corresponding multiplications across all the layers, who share the same matrix column indexes.

B. Jacobi rotation component

Similar to the Jacobi rotation component from previous work [8], this Jacobi rotation component is mainly responsible for performing orthogonal transformation between column vectors through a series of operations with paired squared column 2-norms and the associate covariance. In addition, the divider in this component is also reused in later computations for query vector coordinates updates and vector cosine similarity computation.

C. Update component

The Update component performs element-wise update on matrix column entries and covariances which are affected by the processed Jacobi rotations in the SVD computing. Generated rotation angle parameters $\cos$ and $\sin$ are employed to update the matrix column covariances before they are used by later rotations, and calculate matrix column entries of both left and right orthogonal matrices. To optimize the hardware resource usage, the multiplier-arrays and their direct connected adders in Vector reduction component are reused as Update component at runtime.

D. 2-3 tree sorting component

The $2-3$ tree sorting component organizes BRAMs into unit to store tree node information, and uses a group of BRAMs to maintain a full tree structure. Among them, the BRAM named $DataMem$ is used to store the floating-point numerics, and each entry of which has corresponding entries in BRAMs $LeftNode$, $MidNode$, $RightNode$, and $ParentNode$ to store the address pointers of its parent, left child, center child, and right child entries. Meanwhile, the BRAM $SortedLink$ keeps the address pointers of the last and next elements in the sorted order, whose access produces the final sorted result. Besides, additional BRAMs are used to record the tree node status, which determines the operations on the tree.

The $2-3$ tree is a self-balanced search tree data structure, and the operations performed on the $2-3$ tree mainly include searching, insertion, update and deletion. When a new data element arrives, the process starts with searching the proper position for insertion. The search function starts with examining the root node, and then is directed to the proper subtree according to the key value of the new element. By recursively performing comparisons from the root node to leaf level-by-level, the search process is ended when a leaf node is reached, and the insertion operation takes place. If only one element exists in a leaf node, the new data element can be directly added into this node, otherwise update operation is started, due to the maximum capacity of a single node being violated with the new element inserted. The update operation splits a tree node into two tree nodes, and moves the elements with middle key up to the parent node. This update operation is performed upwards recursively if parent tree node capacity is exceeded. A new root node will be generated if current root node has more than two data elements. Also, the node deletion requires iterative updates from child node to parent node until the $2-3$ tree property is satisfied, but with data moving downwards.

To parallelize the $2-3$ tree operations, numerous search processes can be executed concurrently, and the key value comparisons at different levels can be performed in parallel. As searching the tree does not alter its structure, the parallelism of the search operation is straightforward. However, after inserting a new element or deleting an old one, the tree structure needs to be updated if the tree property is violated. In most cases, the search, insertion, deletion and update operations can be performed simultaneously, since this component maintains the tree structure and update the tree nodes connections locally. An example $2-3$ tree operation process is demonstrated in Fig. 3. Here, independent comparisons are performed concurrently at different level except the stalling happens at the second level tree node when 10 arrives, due to its right children leaf node starting an update to recover the tree property violation at this cycle. Additionally, the $2-3$ tree is partitioned into numerous groups of on-chip BRAM units, which are operated in parallel.

V. Implementation and Experimental Evaluation

A. Implementation and experimental setup

To evaluate the performance of our Latent Sematic Indexing design, we implement our architecture on a single Xilinx Virtex-5 XC5VLX330 FPGA on the Convey HC-2 system [11]. In our implementation, we generate the double-precision floating-point computational cores by using Xilinx Coregen generator [12]. In the Vector reduction component, eight layers

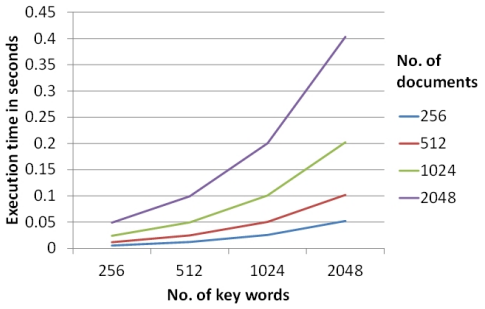

Fig. 4: LSI computation time (in seconds) for matrices with different dimensions(k is 128)

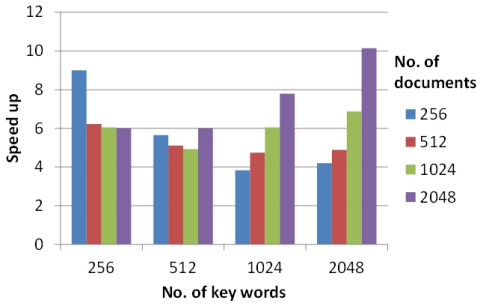

Fig. 5: Speedups of our LSI process compare to Matlab LSI program execution.

of multiplier-array are implemented, in which 32 multipliers and 32 adders are used. In the Jacobi rotation component, 1 multiplier, 2 adder, 1 divider and 1 square-root operator are employed, which initializes 8 independent Jacobi rotations in the pipeline in every 64 clock cycles. In our system, our generated computational cores are configured with default latencies as 9, 14, 57, 57 clock cycles for multiplier, adder, divider and square-root calculator respectively. Four groups of eight simple dual port RAMs are employed for the $2-3$ tree sorting component. The system is evaluated by executing at 150 Mhz, in which the SVD computing is an iterative process, and each element is rotated by 6 times.

B. Performance analysis

We experiment with both square and rectangular matrices with different dimensions of reduced k-subspace, the performance for matrices with dimensions from 256 to 1024 is demonstrated in Fig. 4, in which the dimension of the subspace is set at 128. The experimental results demonstrate that the execution time grows significantly as the number of documents increases, due to the amount of updates for the right orthogonal matrices, and matrix vector covariances is determined by the quantity of documents. Comparably, the number of key words, which determines the dimensions of the left orthogonal matrix, has smaller impact on the overall performance. However, the LSI process requires the usage of both left and right orthogonal matrices, and the matrices elements updates after each rotation usually dominates the performance of the SVD computation for medium to large sized matrices.

Comparisons of execution times have been made between our implementation and Matlab LSI program, and in Fig. 5, the dimensional speedups of our design compared to the Matlab 7.10.0 LSI program running on a 2.2 GHz dual core Intel Xeon processor are demonstrated, in which Matlab uses the SVD and sorting routines. By analyzing those data points in Fig. 5, our architecture shows better efficiency than Matlab implementation, with dimensional speedups that can be achieved range from $3.8\times$ to $10.1\times$ for matrices with dimensions from 256 to 2048. The speedup decreases as the I/O limits start to affect the overall performance, and the speedups then gradually increase as the dimensions have further growth due to the improved efficiency of the pipelined computational cores.

VI. Conclusion

An FPGA-based hardware architecture is proposed and implemented to perform Latent Semantic Indexing, which parallelizes the Hestenes-Jacobi SVD computation, the vector computation, and the ordered tree-based sorting process. The performance analysis indicates our design has achieved dimensional-dependent speedups range from $3.8\times$ to $10.1\times$ compared to a standard software solution.

References

[1] K. R. Gee, "Using Latent Semantic Indexing to filter spam," in *Proceedings of the ACM Symposium on Applied Computing*, 2003, pp. 460–464.

[2] J. Maletic and A. Marcus, "Using Latent Semantic Analysis to identify similarities in source code to support program understanding," in *Proceedings of IEEE International Conference on Tools with Artificial Intelligence*, 2000, pp. 46–53.

[3] B. Vanteru, J. Shaik, and M. Yeasin, "Semantically linking and browsing PubMed abstracts with gene ontology," *Journal of BMC Genomics*, vol. 9, no. Suppl 1, p. S10, 2008.

[4] A. Majumdar, S. Cadambi, S. Chakradhar, and H. Graf, "A parallel accelerator for semantic search," in *Proceedings of IEEE Symposium on Application Specific Processors*, June 2011, pp. 122–128.

[5] S. Eick, J. Lockwood, R. Loui, A. Levine, J. Mauger, D. Weishar, A. Ratner, and J. Byrnes, "Hardware accelerated algorithms for semantic processing of document streams," in *Proceedings of IEEE Aerospace Conference*, 2006.

[6] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf, "A programmable parallel accelerator for learning and classification," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2010, pp. 273–284.

[7] H. P. Graf, S. Cadambi, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and I. Dourdanovic, "A massively parallel digital learning processor," in *Advances in Neural Information Processing Systems 21*. Curran Associates, Inc., 2009, pp. 529–536.

[8] X. Wang and J. Zambreno, "An FPGA implementation of the hestenes-jacobi algorithm for Singular Value Decomposition," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium Workshops*, 2014, pp. 220–227.

[9] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[10] Y.-H. E. Yang and V. K. Prasanna, "High throughput and large capacity pipelined dynamic search tree on FPGA," in *Proceedings of the Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2010, pp. 83–92.

[11] "The convey hc-2 computer architecture overview." [Online]. Available: http://www.conveycomputer.com/

[12] "Logicore IP floating-point operator data sheet," March 2011. [Online]. Available: http://www.xilinx.com/