PARALLEL LATENT SEMANTIC INDEXING ALGORITHM ON CELL BROADBAND ENGINE ARCHITECTURE

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree
of
MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

By

KANASE-PATIL PADMAJA BABURAO

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)

MAY, 2010

Candidate's Declaration

I hereby declare that the work being presented in the dissertation report titled **"Parallel Latent Semantic Indexing Algorithm on Cell Broadband Engine Architecture"** in partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Computer Science and Engineering**, submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, is an authentic record of my own work carried out under the guidance of Dr. Kuldip Singh and Dr. Ankush Mittal (Ex-faculty) in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee. I have not submitted the matter embodied in this dissertation report for the award of any other degree.

Dated: 28/05/2010

Place: IIT, Roorkee

[illegible]

Kanase Patil Padmaja Baburao

Certificate

This is to certify that above statements made by the candidate are correct to the best of our knowledge and belief.

Dated: 28/05/2010

Place: IIT, Roorkee

[illegible]

Dr. Kuldip Singh,
Professor,
Department of Electronics and
Computer Engineering.

[illegible]

Dr. Ankush Mittal
Former Associate Professor,
Department of Electronics and
Computer Engineering

ACKNOWLEDGEMENTS

I am thankful to Indian Institute of Technology Roorkee for giving me this opportunity. It is my privilege to express my thanks to my guide **Dr. Kuldip Singh** whose constant motivation, patience and enthusiasm were immensely helpful in carrying out this work. .I would like to express my deep sense of gratitude to my ex-guide **Dr. Ankush Mittal**, for his invaluable guidance and constant encouragement throughout the dissertation. I would like to thank **Dr. S. N. Sinha**, Head of ECE Department, for supporting us at critical juncture during the thesis submission.

I acknowledge Georgia Institute of Technology, its Sony-Toshiba-IBM Center of Competence, and the National Science Foundation, for the use of Cell Broadband Engine resources that have contributed to this research. I would also like to thank the authors of book "Numerical Recipes in C" for developing the uniprocessor implementation of the SVD algorithm that I have modified and parallelized in this work.

I take this opportunity to extend my sincere thanks to all my friends at IIT Roorkee for the constant support throughout my stay here. I am highly grateful to my brother **Mr. Amarsingh** and my parents for their love and blessings which always motivated me to work better. Finally I thank god for being kind to me and driving me through this journey.

(Kanase Patil Padmaja Baburao)

ABSTRACT

With the growing use of search engines, information retrieval has become a latest research area. The growth of internet and explosion of information are the major factors due to which the size of documents has increased many folds. Thus there is a need of effective and efficient information retrieval mechanism. Latent Semantic Indexing (LSI) is one of the techniques for information retrieval which is very effective in correlating and retrieving relevant documents.

The critical step involved in LSI algorithm is Singular Value Decomposition (SVD). SVD is a mathematical technique which is basically a matrix decomposition method. The SVD is highly effective to derive the semantic relevance, but it is computationally very expensive in terms of time and memory. Thus SVD becomes a bottleneck for quick retrieval of matching documents from large database. This necessitates the optimization of algorithm by parallelization using high performing architecture.

Multicore processors can be used for such type of problems which are computationally intensive. The Cell Broadband Engine is one such multicore processor consisting of a traditional PowerPC based master core meant to run the operating system, and 8 delegate slave processors built for compute intensive processing. This work introduces a modification on the serial singular value decomposition algorithm. It describes parallel implementation of the modified algorithm on Cell BE and issues involved. Exposure of system level optimization features in Cell BE has been employed on algorithm specific operations to achieve improvements to a great extent. The implementation achieves significant performance, thereby giving about 8 times speedup over sequential implementation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

CBE	Cell Broadband Engine
DMA	Direct Memory Access
EIB	Element Interconnect Bus
LS	Local Store
LSI	Latent Semantic Indexing
MFC	Memory Flow controller
MIC	Memory Interface Card
PPE	PowerPC Processing Element
SIMD	Single Instruction Multiple Data
SNR	Signal Notification Register
SPE	Synergistic Processing Element
SPU	Synergistic Processing Unit
SVD	Singular Value Decomposition

LIST OF PUBLICATIONS

1. Padmaja Kanase-Patil, Ankush Mittal, Kuldip Singh, “Implementation of Minimum Spanning Tree Algorithm on Cell BE”, Accepted in International Conference on Advanced Computation Engineering, Bangalore, 21-22 June 2010

2. Padmaja Kanase-Patil, Kuldip Singh, Ankush Mittal, “Parallel Householder Bidiagonalization on Cell Broadband Engine Architecture”, Accepted in Fifth International Conference on Industrial and Information System, NIT Surathkal, 29th July-1st August, 2010

1.1 Introduction to the problem

The amount of information that is retrieved and processed is growing rapidly and hence there is requirement for a faster and effective information retrieval tool. There are a number of ambiguities associated with the English language like polysemy and synonymy which makes information retrieval more complex. Many of such problems can be minimized when the query is provided in context. Latent Semantic Indexing (LSI) [1] is one of the techniques for information retrieval which is used to retrieve the documents from set of documents which are semantically relevant. It is a statistical technique which can extract contextual and structural information within words and sentences in a document [2].

LSI technique works on Term-Document Matrix (TDM). TDM is generated by arranging all content words within all documents in a document set along the vertical axis and all documents along the horizontal axis. LSI technique breaks down this matrix into smaller components using matrix decomposition method called as Singular Value Decomposition (SVD) [3]. SVD is a mathematical technique which decomposes the TDM matrix into a set of smaller components. These smaller components are then used to find the semantic relevance between query provided by user and document. Basically SVD is used to discard redundant information and focus only on essential semantic information.

SVD is extraordinarily useful mathematical technique which breaks the TDM matrix A in the form of $A = U \Sigma V^T$. U is orthogonal matrix which represents term vectors in new latent semantic space. Σ is diagonal matrix. The diagonal entries are known as the singular value which is the measure for relevance of terms and documents. V^T is orthogonal matrix which represents document vectors in new latent semantic space. We can create an approximation A' to the original A by choosing the top singular values in Σ, and setting the rest to zero. This is the step where we actually remove redundant and noisy information. A number of methods and approaches have been proposed for calculation of SVD [12].

Though the SVD is highly effective to derive the semantic relevance between the documents, it is computationally very expensive in terms of time and memory. Thus, implementation of LSI requires investment of storage and computing time [1].

1.2 Motivation

Size of document set has increased many folds with the growth of internet and an explosion of available information. LSI based systems are very effective at retrieving the correct documents, but they become slow and take a lot of processing power to process such large document set. LSI is very expensive to perform on real life sets of documents, because very large matrices are involved and SVD on such large matrices is very costly in terms of time [4]. Thus it requires a powerful hardware to make it fast. One solution is to have a large number of computers in a cluster or a grid. However this is a costly method. An alternate method can be devised in the form of multicore computer like Cell Processor.

With introduction of Moore's law the microprocessor industry is making a consistent effort to deliver a higher throughput. The recent trend of technology suggests the use of more number of execution cores in each processor. Multicore technology focuses on delivering pure performance and optimizing power requirements. Multiple cores introduced in a processor are capable of sharing work and executing independent tasks simultaneously. The other benefits are enhanced performance and reduced power consumption.

Multicore processors can be used for the problems which are computationally intensive. They can provide a significant amount of peak performance gain as compared to the uniprocessor. The Cell BE processor is one such multi-core processor. It is designed with the computationally intensive applications in mind, often used to achieve real time processing and reduce the execution time considerably for various applications.

Cell processor bears a huge potential for compute-intensive applications. But, it poses new implementation challenges because of its unique architecture and often requires substantial re-engineering of the existing algorithms. Motivation of this thesis is to utilize the dense computational power of the Cell architecture for parallelization of SVD and achieve the maximum performance.

1.3 Problem Statement

In this dissertation work we propose and implement the SVD of a matrix on Cell BE while addressing the real time processing aspects of the algorithm. We aim to parallelize the algorithm efficiently to reduce the running time of algorithm.

The major contributions of this dissertation work lie in:

1. Identifying major issues involved in porting a sequential SVD algorithm on a cell processor.
2. Proposing potential solutions to these issues through parallel implementation.
3. Evaluation of these solutions by porting parallel SVD algorithm on cell processors.

This dissertation work presents the parallel implementation of SVD algorithm on multicore CBE Architecture. The implementation tries to utilize all the features of this architecture and at the same time it considers the architectural limitations, thereby obtains considerable performance gain.

1.5 Organization of Thesis

This thesis proposes the parallel model for SVD on CBE. The organization of the thesis is as follows:

Chapter 2 discusses the hardware architecture of the multicore processor STI CBE and describes its various components. This also discusses some programming features of

Cell Broadband Engine which plays major role in achieving the significant performance.

Chapter 3 discusses the background details of LSI and SVD computation of matrix. It also describes the various techniques suggested to carry out the SVD and approach used in the dissertation work for calculation of SVD.

Chapter 4 discusses the sequential Householder Bidiagonalization algorithm, issues involved in implementing parallel version of it on Cell BE architecture and the solutions to those issues.

Chapter 5 discusses the issues involved in implementing the diagonalization of bidiagonal matrix on Cell BE architecture and then provides solutions to those issues.

Chapter 6 discusses work environmental setup on which experiment is carried out and the experimental results for various modules.

Finally chapter 7 concludes the dissertation work and also discusses possible extensions.

CELL BROADBAND ENGINE ARCHITECTURE — CHAPTER 2

Cell Broadband Engine is a joint venture of Sony, Toshiba and IBM Corporation formed in 2001. This collaboration of three companies is known as STI. The CBE processor is the first implementation of a new family of multiprocessors which extends 64 bit Power PC Architecture. The Cell BE is mainly intended for application in game consoles and media-rich consumer-electronics devices such as high-definition televisions. But the architecture and the implementation have been designed to enable fundamental advances in processor performance [6].

The CBE is a single-chip multiprocessor with nine processors operating on a shared and coherent memory. All the nine cores share the main memory. The most distinguishing feature of the CBE is that, the function of these cores is specialized into two types: the PowerPC Processor Element (PPE), and the Synergistic Processor Element (SPE). The CBE has one PPE and eight SPEs [7].

2.1 Overcoming the Performance Limits

The performance of present high-frequency processors is limited by three major factors.

1. ***Memory Wall***: Program performance is dominated by the activity of moving data between main storage and the processor.
2. ***Frequency Wall***: Conventional processors require increasingly deeper instruction pipelines to achieve higher operating frequencies. This technique has reached a point of diminishing returns
3. ***Power Wall***: Higher frequency implies higher operating voltage and hence, higher heat dissipation.

Cell BE overcomes the memory wall by using a 3-level memory architecture consisting of disk memory, shared main memory between SPEs and PPE and a local store in each SPE. Use of asynchronous DMA transfer and DMA multiple double buffering further add the speedup. It overcomes the frequency wall by making use of non- homogeneous parallelization. By specializing the PPE and the SPEs for control and compute-intensive tasks, cell BE works at high frequency without excessive overhead. In CBE, the power efficiency is increased by providing a general-purpose PPE to run the operating system and other control-plane code, and eight SPEs specialized for computing data-rich (data-plane) applications [7].

2.2 Architecture

The CBE increases concurrency through the use of multiple processing cores and increases the specialization in execution through non-homogeneous parallelization. For this purpose, it employs 8 SPEs onto which threads of an application can be mapped and these SPEs are controlled by PPE. PPE and SPEs communicate through a common internal high-speed Element Interconnect Bus (EIB). The SPE offers a high bandwidth interface to a direct memory access (DMA) that can transfer 32 GB/sec to and from the 256 KB local memory. The CBE has clock speed of 3.2 GHz [6, 8]. The architecture is shown in Figure 2.1.

2.2.1 PowerPC Processor Element (PPE)

The PPE is a dual thread PowerPC architecture RISC core and support a PowerPC virtual memory subsystem. It has 32KB L1 instruction and data caches and 512 KB L2 (instruction and data) cache. It runs an operating system, manages system resources, and is intended primarily for control processing, including the allocation and management of SPE threads. The instruction set for PPE is an extension of the PowerPC instruction set. It also includes a vector multimedia extension unit, called SIMD, so that it can do multiple operations simultaneously with a single instruction [6].

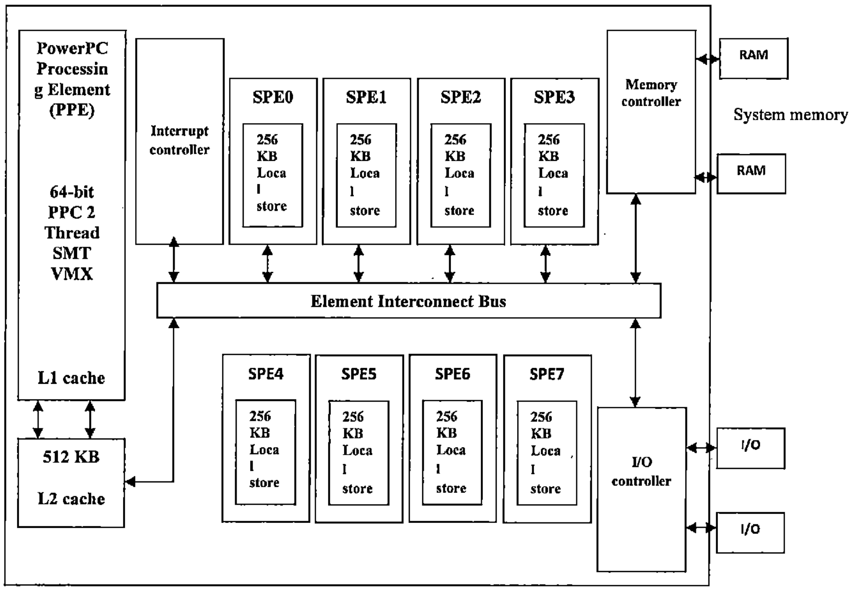


Figure 2.1Cell Broadband Engine Architecture [9]

2.2.2 Synergistic Processor Elements (SPEs)

Eight homogeneous SPEs are SIMD processor elements that are optimized for data-rich operations. It consists of two main units, the Synergistic Processor Unit (SPU) and the Memory Flow Controller (MFC). The SPE deals with instruction control and execution. It includes a single register file with 128 registers (each one 128 bits wide), a unified (instructions and data) 256-KB local store (LS), an instruction-control unit, a load and store unit, two fixed-point units, a floating-point unit, and a channel-and-DMA interface. The SPE implements a new SIMD instruction set, the SPE Instruction Set Architecture.

The MFC contains a DMA controller that supports DMA transfers. Programs running on the SPE use the MFC's DMA transfers to move instructions and data between the SPE's LS and main storage. The MFC interfaces the SPE to the EIB which manages bus bandwidth-reservation and synchronization operations between the SPE and all other processors in the system [6].

To support DMA transfers, the MFC maintains and processes queues of DMA commands [7]. After a DMA command has been queued to the MFC, the SPE can continue to execute instructions while the MFC processes the DMA command autonomously and asynchronously.

2.2.3 Element Interconnect Bus (EIB)

The EIB is a communication bus internal to the Cell processor which connects the PPE, the memory controller (MIC), eight SPE coprocessors, and two off-chip I/O interfaces, for a total of 12 participants. The EIB is presently implemented as a circular ring consisting of four 16B-wide unidirectional channels. Out of these four channels, two run in clockwise direction and rest two run in anticlockwise direction. When traffic patterns permit, each channel can convey up to three transactions concurrently. As the EIB runs at half the system clock rate the effective channel rate is 16B every two system clocks. At maximum concurrency, with three active transactions on each of the four rings, the peak instantaneous EIB bandwidth is 96B per clock (12 concurrent transactions * 16B wide / 2 system clocks per transfer).

2.3 Programming Features of Cell-BE

2.3.1 SIMD Vectorization

Support for SIMD operations is pervasive in the CBE. SIMD operands are vectors. A vector is an instruction operand containing a set of data elements packed into a one dimensional array. The elements can be integer or floating-point values. In the PPE, they are supported by the Vector/SIMD Multimedia Extension instruction set. In the SPEs, they are supported by the SPU instruction set. In both the PPE and SPEs, vector registers hold multiple data elements as a single vector. The data paths and registers supporting SIMD operations are 128 bits wide, corresponding to four full 32-bit words. This means that four 32-bit words can be loaded into a single register and single operation (for example, addition) can be performed on these four data simultaneously. Similar operations can be performed on vector operands containing 16 bytes, 8 half-words, or 2 double-words [7]. The following figure 2.2 shows such an operation [7, 8].

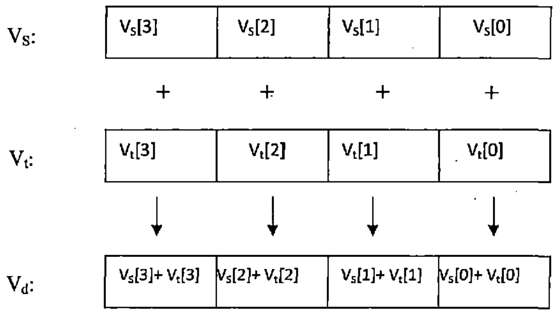


Figure 2.2 SIMD operations in Cell BE

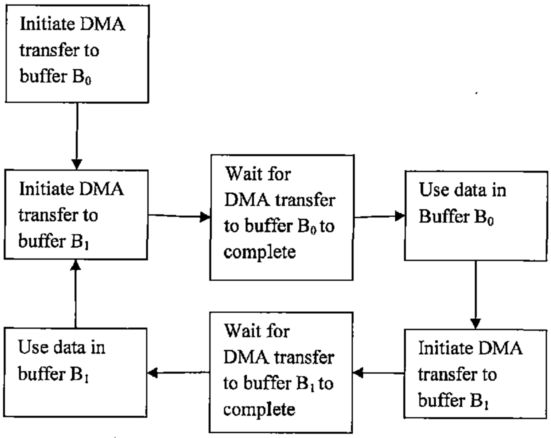
2.3.2 DMA and Double Buffering

MFC supports a set of DMA commands which provide the main mechanism for data transfer between the LS and main storage. It supports a set of synchronization commands which are used to control the order in which storage accesses are performed. Consider an SPE program that requires large amounts of data from main storage. The following is a simple scheme to achieve that data transfer:

1. Start a DMA data transfer from main storage to buffer B in the LS.
2. Wait for the transfer to complete.
3. Use the data in buffer B.
4. Repeat.

There is a wastage of large amount of waiting time for DMA transfers using this method. We can speedup the process significantly by allocating two buffers, B0 and

B1, and overlapping computation on one buffer with data transfer in the other. This technique is called double buffering. The below figure 2.3 shows a flow diagram for this double buffering scheme. Double buffering is a form of multi-buffering, which is the method of using multiple buffers in a circular queue to overlap processing and data transfer [6].



Initiate DMA transfer to buffer B_0
Initiate DMA transfer to buffer B_1
Wait for DMA transfer to buffer B_0 to complete
Use data in Buffer B_0
Initiate DMA transfer to buffer B_1
Wait for DMA transfer to buffer B_1 to complete
Use data in buffer B_1

Figure 2.3 Double buffering

2.3.3 Mailbox

While DMA transfer allows transfer of up to 16K bytes of data between the main memory and each SPE's LS, mailboxes are designed for transfer of 32-bit data between the PPE and the SPE. Structurally, mailboxes are FIFO queues [10]. The MFC provides three types of mailbox queues, each with a different behavior and data transfer direction as shown in Figure 2.4.

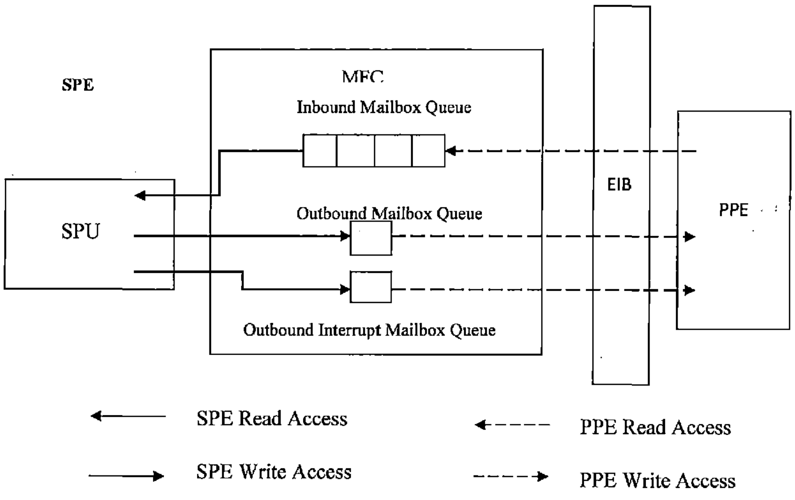


Figure 2.4 Mailbox communication mechanism in Cell BE [10]

1. ***SPU Inbound Mailbox***: This is used to send data from the PPE to the SPE. This mailbox has space for storing up to four 32-bit messages at a time. If no message is found when the SPE program accessed the queue, the SPE stalls until data is written by the PPE program.

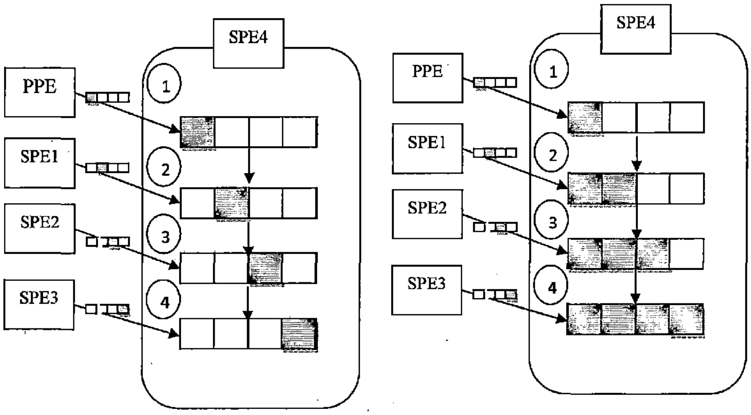
2. ***SPU Outbound Mailbox***: This is used to pass data from the SPE to the PPE. This mailbox has the capacity to accept only one 32-bit message. If the SPU outbound mailbox is full, writing of the next data is suspended until the PPE reads the data from the queue.

3. ***SPU Outbound Interrupt Mailbox***: Like the SPU outbound mailbox, this is used to send data from the SPE to the PPE. When this mailbox is written, however, an interrupt event is generated to notify the PPE when to read the data.

These mailboxes can be accessed either from the SPE or PPE programs.

2.3.4 Signal Notification Register

This is one more mechanism provided for SPE-PPE communication. Signal Notification Registers (SNRs) are 32-bit registers used to send signals, such as control messages and events, to an SPE from other SPEs or the PPE. There are two SNRs for each SPE. The sending processor (either PPE or SPE) writes the signal value in the form of 32-bit data into the SNR of the receiving processor (one of other SPEs). When the value is read by the receiving processor, all bits in the SNR are reset to zero. If the SNR is empty when it is read, the receiving processor stops execution until the signal is written [10].



(a) Overwrite Mode (b) Logical OR Mode

Figure 2.5 Signal Notification Registers in Cell BE [10]

NRs can be configured for overwrite mode or logical OR mode. The overwrite is useful in a one-to-one signaling environment, whereas the logical OR mode ›s many-to-one signaling. Either of these modes can be selected for each SNR, independently of the other. Figure 2.5 shows the difference between the SNR's two operating modes. The SNRs also make barrier synchronization of multiple SPE programs possible by configuring them into logical OR mode.

3.1. Latent Semantic Indexing

Latent Semantic Indexing is an information retrieval technique that projects queries and documents into a space with "latent" or hidden semantic dimensions. This space is called as Latent Semantic Space. In the latent semantic space, a query and a document can have high similarity even if they do not share any terms - as long as their terms are semantically similar. Also they can be distant from each other, even if they share some common terms. This latent semantic space has fewer dimensions than the original space. LSI is a method for dimensionality reduction [1]. In short, LSI projects the queries and documents into space with smaller number of dimension (k) from the space with very large number of dimensions (n) where n > k.

LSI uses a method from linear algebra, the Singular Value Decomposition (SVD) for dimensionality reduction. SVD takes a matrix A and represents it as A' in a lower dimensional space [11].

The SVD projection is computed by decomposing the Term-Document Matrix $A_{m \times n}$ into the product of three matrices, $U_{m \times n}$, $S_{n \times n}$ and $V^T_{n \times n}$ as

$$A = U \Sigma V^T$$

Where,

- U is an m x n orthogonal matrix
- Σ is an n x n diagonal matrix. The diagonal entries are known as the singular values.
- V^T is an n x n orthogonal matrix

We can view SVD as a method for rotating the axes of the n-dimensional space such that the first axis runs along the direction of largest variation among the documents, the second dimension runs along the direction with the second largest variation and so forth. The matrices U and V represent terms and documents in this new space. The diagonal matrix Σ contains the singular values of A in descending order. The ith singular value indicates the amount of variation along the ith axis [1, 2].

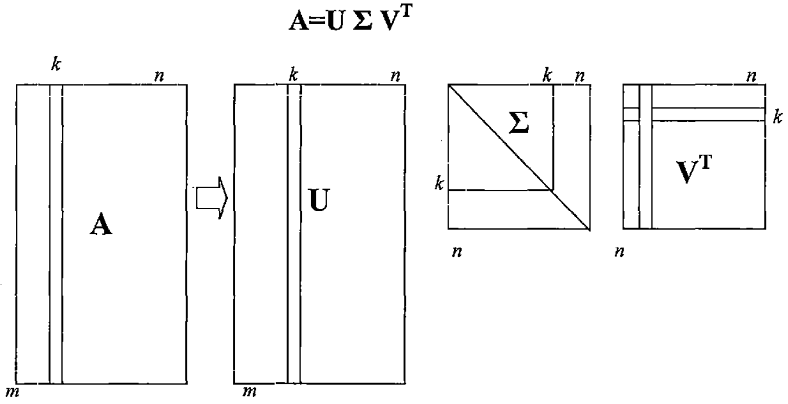


Figure 3.1 Matrix Decomposition

The best square approximation of matrix A of rank k (Figure 3.1) can be calculated by restricting the matrices U, Σ and V to their first k < n rows, as shown in below,

$$A' = U_{m\,x\,k}\,\Sigma_{k\,x\,k}V^T{}_{m\,x\,k}$$

To choose the number of dimensions (k) for A′ is an interesting problem. Reduction in k can remove much of the noise but keeping too few dimensions or factors may loose important information. LSI performance can improve considerably after 10 or 20 dimensions, peaks between 70 and 100 dimensions, and then begins to diminish slowly [4].

3.2 Singular Value Decomposition

3.2.1 Literature Review

Several algorithms have been developed for mathematical computation of singular value decomposition. We discuss here shortly some of the algorithms for computing the SVD. A number of additional algorithms exist, but they are not discussed here because they appear to be less suited for parallelization in our judgment. It cannot be predicted a clear winner among parallel SVD algorithms that can provide the single best trade-off between numerical stability [12], algorithmic complexity, parallelizability, efficiency, and ease of programming on CBE multicore architecture.

➢ **Classical Jacobi Method**

The classical Jacobi method applies to symmetric matrices. It transforms a symmetric N X N matrix A into a diagonal matrix by means of a sequence of Jacobi transformations [13]. Each of the Jacobi transformations is chosen to find the off-diagonal elements of largest absolute value. The classical Jacobi method exhibits high numerical stability and convergence but computationally it is very slow [13, 14].

➢ **Cyclic Jacobi Method**

The main disadvantage of the classical Jacobi method is the computational cost required to determine the largest off-diagonal element. Rather than searching for the largest element, a cyclic Jacobi method uses approach which applies Jacobi transformations in a data-independent fashion. This method is computationally faster than the classic Jacobi Transformation.

The primary problem with the cyclic Jacobi method is the two-sidedness of the Jacobi rotation. Matrices are stored either in row-major or column-major format. The two-sided Jacobi method traverses both. One of the two traversals might give poor performance conventional memory architectures [15].

➢ **Hestenes-Jacobi Method**

Hestenes [16] introduced the one-sided Jacobi method by discovering the equivalence between orthogonalization of two vectors and annihilating a matrix element by means of orthogonal plane rotations. Hestenes algorithm gives low performance for multicore architecture and hence not popular.

➢ **Golub-Kahan-Reinsch SVD**

The SVD algorithm suggested by Golub, Kahan, and Reinsch [17] has become the standard method for computation of SVD. This SVD algorithm is based on bidiagonalization. It consists of two phases, bidiagonalization and subsequent diagonalization.

3.2.2 Our Approach: Golub-Kahan-Reinsch SVD

The fastest available iterative Jacobi algorithms are slower than the fastest algorithms based on bidiagonalization. We follow the Golub-Kahan-Reinsch approach to calculate SVD as it is simple and compact, and maps well to the CBE multicore architecture. Algorithm 3.1 describes the SVD algorithm for an input matrix A.

Algorithm 3.1 Singular Value Decomposition

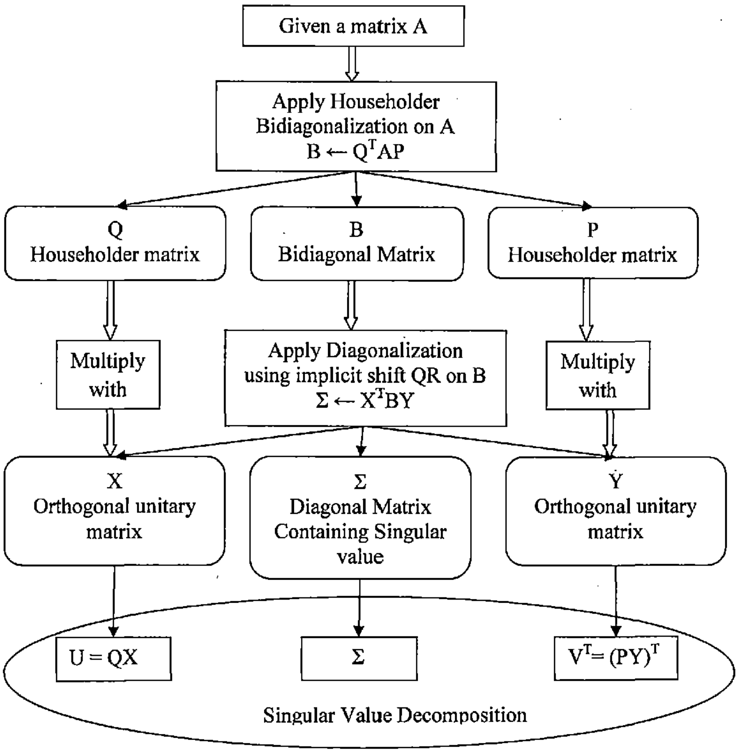
```
1. $B \leftarrow Q^TAP$                {Bidiagonalization of A to B}

2. $\Sigma \leftarrow X^TBY$                {Diagonalization of B to $\Sigma$}

3. $U \leftarrow QX$          }    {Compute orthogonal matrices U and $V^T$
4. $V^T \leftarrow (PY)^T$    }     and SVD of $A = U\Sigma V^T$}
```

Figure 3.2 Golub – Kahan - Reinsch SVD algorithm

The SVD computation suggested by Golub-Kahan-Reinsch consists of two phases, bidiagonalization and subsequent diagonalization [18]. Bidiagonalization can be achieved by means of alternating QR and QL factorizations to annihilate column and row blocks or Householder bidiagonalization [18]. We use the latter method for bidiagonalization. The matrix is first reduced to a bidiagonal matrix using a series of householder transformations. The bidiagonal matrix is then diagonalized by performing implicitly shifted QR iterations [19].



Given a matrix A
Apply Householder Bidiagonalization on A $B \leftarrow Q^TAP$
Q Householder matrix
B Bidiagonal Matrix
P Householder matrix
Multiply with
Apply Diagonalization using implicit shift QR on B $\Sigma \leftarrow X^TBY$
Multiply with
X Orthogonal unitary matrix
Σ Diagonal Matrix Containing Singular value
Y Orthogonal unitary matrix
$U = QX$
Σ
$V^T = (PY)^T$
Singular Value Decomposition

Figure 3.3 Flow of Singular Value Decomposition Algorithm

Figure 3.3 shows the flow of Golub - Kahan - Reinsch SVD algorithm-how the matrix is reduced to first bidiagonal and then to diagonal form and also how matrices are generated to calculate the singular value decomposition.

The subsequent chapters focus on Householder bidiagonalization algorithm (Chapter 4) and Diagonalization of bidiagonal matrix algorithm using implicit QR shifting (Chapter 5).

The implementation of the Singular Value Decomposition requires that the matrix be reduced to a simpler form. In our case it is an upper bidiagonal form [20], which is given as follows,

$$\begin{pmatrix} D_1 & e_1 & & & & \\ & D_2 & e_2 & & & \\ & & D_3 & e_3 & & \\ & & \cdots & \cdots & & \\ & & & D_{n-1} & e_{n-1} \\ & & & & D_n \end{pmatrix}$$

The basic idea to reduce the matrix in bidiagonal form is to decompose the matrix A in the following form:

$$A = QBP^T$$

This reduction of matrix A is achieved by applying a series of householder transformations where B is a bidiagonal matrix and Q and P are unitary householder matrices. The householder matrix acts on given vector to zero all its elements except the first one [18].

4.1. Householder Matrix

The Householder Matrix P has the form:

$$P = 1 - 2W.W^T \tag{4.1}$$

Where W is real vector with $|W|^2 = 1$.The matrix P is orthogonal because

$$P^2 = (1 - 2W.W^T) . (1 - 2W.W^T) = 1 \tag{4.2}$$

Therefore, $P = P^{-1}$. But P^T=P, and so $P^T = P^{-1}$ proving orthogonality. Rewrite P as

$$P = 1 - \frac{u.u^T}{H} \tag{4.3}$$

Where the scalar H is

$$H := \frac{1}{2}|u|^2 \tag{4.4}$$

And now *u* can be any vector. Suppose *x* is vector consists of first column of A. Choose

$$u = x \pm |x| e_1 \tag{4.5}$$

Where e_1 is unit vector [1, 0, 0....0] ^{T}. Then we have

$$P.x = \pm |x| e_1 \tag{4.6}$$

This shows that the householder matrix P acts on a given vector *x* to zero all its elements except the first one [18].

4.2 Sequential Algorithm for Householder Bidiagonalization

The standard way to reduce the matrix into bidiagonal form is to alternately pre and post multiply A with Householder transformations in order to introduce zeros in the :olumns and rows of the matrix [21, 22]. Each transformation annihilates the required)art of a whole column and whole corresponding row. To obtain the matrices P nd Q, one may either explicitly accumulate the transformations "on the fly" or store ıe Householder vectors "in-place" in the zeroed parts of A for later backward- :cumulation [21]. We follow latter strategy.

the first step we choose Householder transformation P so as to introduce zeros in hatted elements in matrix A.

$$\begin{pmatrix} X & X & X & X & X \\ \hat{X} & X & X & X & X \\ \hat{X} & X & X & X & X \\ \hat{X} & X & X & X & X \\ \hat{X} & X & X & X & X \end{pmatrix}$$

After which P · A has the form:

$$\begin{pmatrix} X & X & \hat{X} & \hat{X} & \hat{X} \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \end{pmatrix}$$

Next we choose K such that P · A · K has zeroes in the hatted elements to get a matrix of the form,

$$\begin{pmatrix} X & X & 0 & 0 & 0 \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \end{pmatrix}.$$

The reduction proceeds recursively on the matrix A to finally form a matrix B of diagonal form.

Instead of actually carrying out the matrix multiplication, we compute a vector

$$P := \frac{A.u}{H} \qquad (4.7)$$

Then we have (see [18] for detailed description)

$$A' = A - q.u^T - u.q^T \qquad (4.8)$$

This is computationally useful formula. In detail, at any stage m (m=1, 2...), the equations are as follows:

$$u^T = [0,0, \dots\dots a_{m,i} + \sqrt{\sigma}, a_{m+1,i}, \dots\dots, a_{n,i}] \tag{4.9}$$

And the quantity σ is

$$\sigma = ((a_{i,m})^2 + \cdots + (a_{i,n})^2) \tag{4.10}$$

Variables are computed in the following order σ, *u*, H, *p*, *q*, A′. At any stage m, matrix A is bidiagonal in its 0 to m-1 columns. Figure 4.1 gives the sequential algorithm for Householder Bidiagonalization.

Algorithm 4.1: Householder Bidiagonalization

```
Input:      A:[1...m][1....n]
Output:     D:[1....n] diagonal elements
            E:[1....n] Offdiagonal elements
            P:[1....n] Householder Matrix
            Q:[1....n] Householder Matrix

For i=1 to n
Do
        Calculate scale. Use scaled a's for transformation.
        Form σ. Eq. (4.10)
        Calculate u Eq. (4.9) and Store it in the A
        Calculate H given in Eq. (4.4)
        Form an element of P.(4.7)
        Form an element of Q.
        Reduce A Eq. (4.8)
End for
```

Figure 4.1 Sequential algorithm for Householder Bidiagonalization

4.3 Parallel Formulation of Algorithm

The CBE architecture has very good architectural potentials like 8 SIMD, compute intensive cores and fast communication mechanisms between the cores. But the main challenging task while developing the parallel algorithm for any application on CBE, is very limited (i.e. 256KB) memory available with each SPE core. The existing algorithm has to be restructured in order to utilize this memory efficiently and also to minimize the communication between different cores.

Bidiagonalization consumes a significant fraction of the total time to compute SVD. It has a drawback of having a lot of matrix-vector multiplication operations. Matrix-vector multiplication requires O (n^2) operations on O (n^2) data, which results in poor computation-to-communication ratio. The parallel algorithm was developed after studying the pattern in the computations and the results. In the limit of large n, the operation count of the Householder reduction is $4n^3$ / 3 [18]. It is a very difficult algorithm to parallelize due to following reasons:

1. Symmetrical nature of computations
2. The diminishing size of the matrix and computations with the iterations. (Figure 4.2)

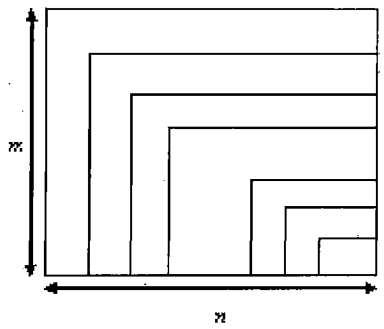


Figure 4.2 Diminishing size of Matrix

The sequential algorithm reduces one row and one column in each iteration. It was observed that the computations in each column could not be done independently of each other. We cannot distribute the whole matrix to each SPE for parallelization of outer loop due to limited memory available with each SPE (i.e. 256KB). So the variables σ, *u*, H, *p*, *q*, A′ are computed in parallel with distribution of part of column to each SPE in each iteration (Figure 4.3). For calculation of each of variable, the part of column is brought onto the SPE memory through DMA.

The DMA does transfer continuous memory data to and from main memory. Thus to distribute the column in different SPEs, the total number of DMA's required is equal to the size of part of column. In order to minimize the DMAs, the matrix is stored in column major order, as shown in Figure 4.3, so that only one DMA will be required in order to transfer part of column.

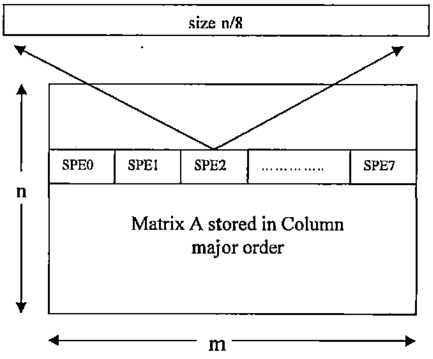


Figure 4.3 Distribution of part of column to each SPE

4.4 Implementation on Cell BE architecture

Figure 4.4 lists the parallel algorithm for sequential Householder Bidiagonalization. The algorithm processes the input matrix by small blocks, providing for great data locality and fine granularity of parallelization. In the case of the Cell processor, it also readily solves the problem of limited size of private memory associated with each computational core.

4.4.1 SPE centric model

The implementation follows the SPE-centric model in which most of the application code is distributed among SPEs. PPE core runs little more than a resource manager for the SPEs. SPE fetches next work item (what function to execute, pointer to data, etc.) from main memory or from its own memory, when it completes current work item [7]. The PPE manages the execution of the overall algorithm relying on the SPEs to deliver computational services. The PPE is responsible for launching and terminating the SPEs. The SPE execution cycle consists of waiting for a request, performing the requested task and sending back a response.

PPE creates 8 SPE threads and assigns the thread ID to each SPE. The effective address of global control block is sent to each SPE thread at the time of creation, which is pulled by each SPE to its local store by a DMA transfer. The control block contains effective address of Matrix A, thread ID of SPE and other synchronization information. This effective address of Matrix A is used by each SPE to fetch the part of Matrix A from main memory through DMA. After this initial exchange of information the execution of algorithm is carried out by SPEs, PPE manages it.

4.4.2 SPE-PPE Communication and barrier operation

To transfer the data from main memory to SPE local store, the DMA communication mechanism [6, 7, 8] has been used. Generally this data is more than 4 bytes. There are special MFC commands available which provide the DMA mechanism. This mechanism enables SPE to access main storage. SPE fetches the part of matrix A from main memory through DMA. Also it transfers the updated part of Matrix A to main emory through DMA. The commands that transfer data from main storage to SPE e referred as get commands and commands that transfer data from SPE to main orage are referred as put commands [8].

```
Algorithm: Parallel Householder Bidiagonalization Implementation

Input:      A: [1...m][1....n]
Output:     D: [1....n] diagonal elements
            E: [1....n] Offdiagonal elements
            P: [1....n][1...n] Householder Matrices
            Q: [1...n][1...n] Householder Matrices

Create 8 SPE threads with threadID [0...7]
Initialize Matrix A and store it into column major order

For i := 0 to n
        SPE fetches the respective part of i^th column of matrix A through DMA.
        SPE calculates the scale and send it to PPE
        PPE calculates global_scale and sends in to each SPE through mailbox
        SPE performs scaling on Matrix A and forms σ on respective part of
column
        SPE sends calculated σ to PPE. PPE calculates global σ.
        SPE updates the respective part of column through DMA
        Synchronize all SPEs
        PPE calculates H in Eq. (4.4)
        SPE receives signal from PPE to start next work.
        SPE fetches the respective part of i^th column of Matrix A through DMA
        SPE forms A.u in respective part of column of A.
        SPE receives calculated H from PPE through Mailbox.
        SPE forms an element of P.
        SPE updates the respective part of column through DMA.
        Synchronize all SPEs.
        PPE forms an element of Q.
        SPE receives signal from PPE to start next work.
        SPE fetches part of column through DMA.
        SPE reduces respective part of column
        SPE updates the respective part of column through DMA
        Synchronize all SPEs
End for
```

Figure 4.4 Parallel algorithm for Householder Bidiagonalization

To send data of 4 bytes, mailbox communication [7, 8] has been used. Two mailboxes (the SPE Write Outbound Mailbox and the SPE Write Outbound Interrupt Mailbox) are provided for sending messages from the SPE to the PPE. One mailbox (the SPE Read Inbound Mailbox) is provided for sending messages to the SPE [8]. Mailbox communication is used to send the control information.

To achieve a barrier operation of only SPE threads (i.e., the PPE is not participating the barrier), signal notification registers [8] provide the most efficient method. The concept is:

- One SPE (i.e. SPE 0) is assigned to be the master, others are slaves.
- The master SPE's signal notification register is configured in logical OR mode [7].
- The slave SPE's signal notification register is configured in overwrite mode.
- Each slave SPE is assigned a bit in signal notification register.

➢ **Synchronization Algorithm:**

- Slave SPE:
  - Writes a single bit to the master's signal notification register.
  - Waits for a message written to its signal notification register.
- Master SPE:
  - Reads the signal notification register until all participant bits are read non-zero.
  - Writes to all the slave SPU's signal notification register.

Figure 4.5 shows the flow of algorithm and PPE-SPE communication at various stages.

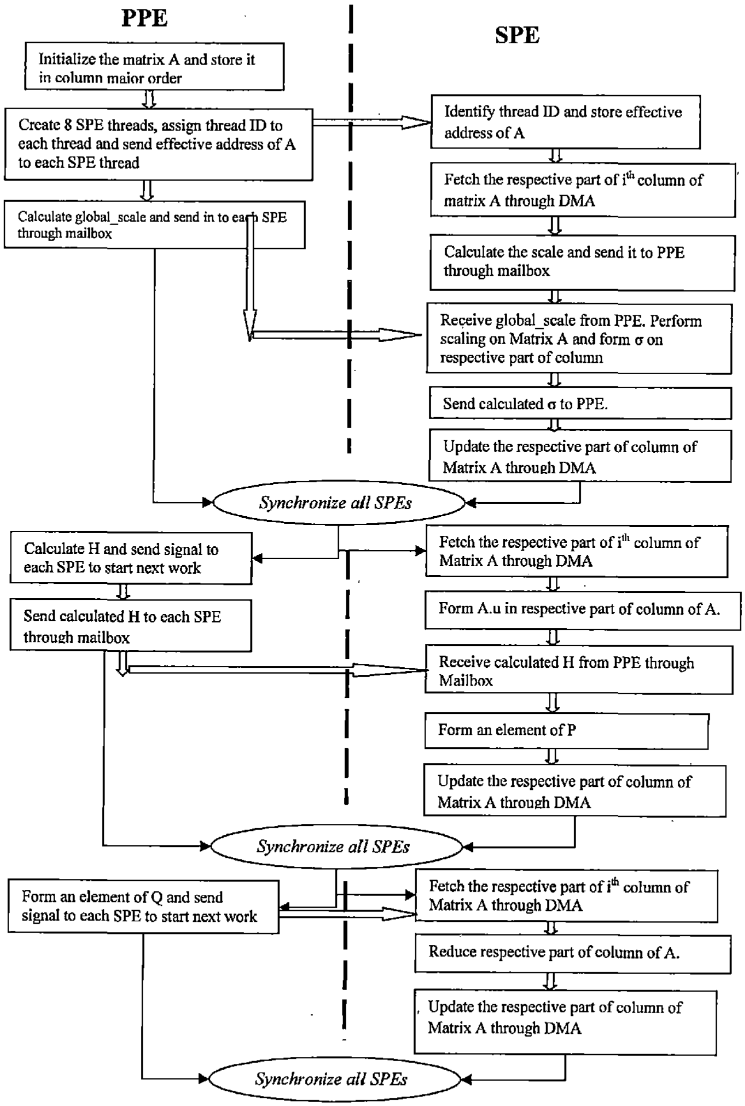

PPE
SPE
Initialize the matrix A and store it in column major order
Create 8 SPE threads, assign thread ID to each thread and send effective address of A to each SPE thread
Calculate global_scale and send in to each SPE through mailbox
Identify thread ID and store effective address of A
Fetch the respective part of ith column of matrix A through DMA
Calculate the scale and send it to PPE through mailbox
Receive global_scale from PPE. Perform scaling on Matrix A and form σ on respective part of column
Send calculated σ to PPE.
Update the respective part of column of Matrix A through DMA
Synchronize all SPEs
Calculate H and send signal to each SPE to start next work
Send calculated H to each SPE through mailbox
Fetch the respective part of ith column of Matrix A through DMA
Form A.u in respective part of column of A.
Receive calculated H from PPE through Mailbox
Form an element of P
Update the respective part of column of Matrix A through DMA
Synchronize all SPEs
Form an element of Q and send signal to each SPE to start next work
Fetch the respective part of ith column of Matrix A through DMA
Reduce respective part of column of A.
Update the respective part of column of Matrix A through DMA
Synchronize all SPEs

Figure 4.5 Flow of Parallel Bidiagonalization Algorithm

DIAGONALIZATION OF BIDIAGONAL MATRIX AND COMPLETE SVD CHAPTER 5

5.1 Diagonalization of Bidiagonal Matrix

Next step in SVD is to reduce the bidiagonal matrix in diagonal form [23]. The bidiagonal matrix can be reduced to a diagonal matrix by iteratively applying the implicitly shifted QR algorithm [19]. The matrix B obtained in the first step is decomposed as

$$\Sigma = X^T B Y$$

Where Σ is a diagonal matrix, X and Y are orthogonal unitary matrices. Σ contains the singular values of matrix B. Each iteration updates the diagonal and super diagonal elements such that the values of the super diagonal elements become less than their previous values. On convergence of the algorithm the superdiagonal elements are reduced to zero and only the diagonal elements of the matrix are left in the matrix X.

5.1.1 QR Algorithm

The basic idea behind the QR algorithm is that any real matrix can be decomposed in the form

$$A = Q \cdot R \qquad (5.1)$$

Where Q is orthogonal and R is upper triangular. For a general matrix, the decomposition is constructed by applying Householder transformations to annihilate successive columns of A below the diagonal. Now consider the matrix formed by writing the factors in (5.1) in the opposite order:

$$A' = R \cdot Q \qquad (5.2)$$

! is orthogonal, equation (5.1) gives $R = Q^T \cdot A$. Thus equation (5.2) becomes

$$A' = Q^T \cdot A \cdot Q \qquad (5.3)$$

We see that A′ is an orthogonal transformation of A.

The workload in the QR algorithm is O (n^3) per iteration for a general matrix, which is prohibitive. However, the workload is only O (n) per iteration for a bidiagonal matrix, which makes it highly efficient on this form [18].

5.1.2 Sequential Diagonalization algorithm

The standard algorithm for finding singular values of a bidiagonal matrix B is the QR algorithm applied implicitly to B^TB. The algorithm computes a sequence B_i of bidiagonal matrices starting from B_0= B as follows. From B_i the algorithm computes a shift σ^2, which is usually taken to be the smallest eigenvalue of the bottom 2 by 2 block of $B_iB_i^T$.

Then the algorithm does an implicit QR factorization of the shifted matrix $B_i^TB_i - \sigma^2 I = QR$, where Q is orthogonal and R upper triangular, from which it computes a bidiagonal B_{i+1} such that $B^T_{i+1}B_{i+1} = RQ + \sigma^2 I$. As i increases, B converges to a diagonal matrix with the singular values on the diagonal [24].

Figure 5.1 lists the sequential algorithm for Diagonalization of bidiagonal matrix (which is obtained in previous step) using the implicit QR shifting technique.

5.1.3 Parallel formulation of algorithm

The diagonalization of the bidiagonal matrix is sequential in nature; the convergence of every superdiagonal element depends on the convergence of the element before it. Hence it is not possible to run parallel threads for convergence. So all the work for Diagonalization has been carried out on PPE only. Some architectural features have been used to gain the performance:

```
Algorithm 5.1 Diagonalization of bidiagonal matrix

Input:      B[1.....n][1....n] is bidiagonal matrix
Output:     D[1....n] contains singular values
            X[1....n] and Y[1.....n] contains the unitary orthogonal matrices

For i :=0 to n
    Look for a single small subdiagonal element to split the matrix.

    Form shift.

    Perform a plane rotation followed by Givens rotations to restore bidiagonal
    form.

    Store calculated eigen value in D.

    Recover from underflow.

    Form eigenvector corresponding to ith diagonal element of B and store it into
    Z [1...n].
End for
```

Figure 5.1 Algorithm for Diagonalization of bidiagonal matrix

1. ***SIMD Vectorization***: The SIMD works on the multiple data performing a single instruction. Parallelization has been achieved through the use of SIMD Math Library functions for solving compute intensive equations involving reciprocal of square root and multiplication. This led to significant performance gain [6, 7, 8].

2. ***Loop Unrolling***: In PPEs, branches are very expensive, and when mis-predicted, results in a loss of 18 cycles. The branches can be reduced by unrolling loops. By unrolling, a long stretch of instructions can be executed on the PPEs without any branch instruction. The PPEs have sufficient number of registers to allow deep unrolling. 4-way loop unrolling has been used in order to gain the performance in some extent [6].

5.2 Complete SVD

We perform following two matrix-matrix multiplications at the end to compute orthogonal matrices as given in Algorithm 3.1.

$$U = QX$$
$$V^T = (PY)^T$$

Where, matrices Q and P are obtained the bidiagonalization step and matrices X and Y are obtained in diagonalization step.

For the matrix-matrix multiplication, each SPE will do the $(n/8)^{th}$ calculation. The distribution of matrices is as shown in the Figure 5.2. Following are the programming features of CBE used for matrix multiplication implementation.

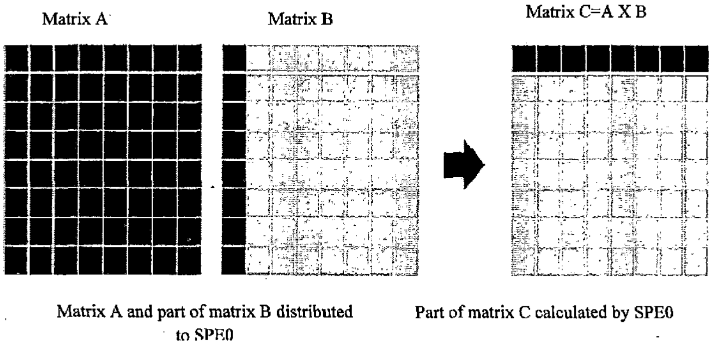

Matrix A
Matrix B
Matrix C=A X B
Matrix A and part of matrix B distributed to SPE0
Part of matrix C calculated by SPE0

Figure 5.2 Matrix multiplications on Cell BE

1. ***SIMD Vectorization***: Parallelization has been achieved through the use of SIMD Math Library functions for addition and multiplication.

2. ***DMA list***: A DMA list is a sequence of transfer elements (or list elements) that, together with an initiating DMA-list command, specifies a sequence of DMA transfers between a single area of LS and possibly discontinuous areas in main

storage. Such lists are stored in an SPE's LS, and the sequence of transfers is initiated with a DMA-list command.

3. ***Double buffering***: We can speedup the DMA process significantly by allocating two buffers, B0 and B1, and overlapping computation on one buffer with data transfer in the other. This method is called double buffering.

6.1 Cell simulator

The whole dissertation work is carried out in simulation environment using IBM's Full System Cell Simulator. It supports full functional simulation, including the PPE, SPEs, MFCs, PPE caches, bus, and memory controller. It can simulate and capture many levels of operational details on instruction execution, cache and memory subsystems, interrupt subsystems, communications and other important system functions [25]. The results are verified on Georgia Tech CBE server which provides a publicly accessible front-end cell-user.cc.gatech.edu.

6.2 Results

In the problem, two parameters are taken as input from the user, M (the number of rows) and N (the number of columns) giving rise to a M x N matrix. Therefore the parallel algorithm was tested for varying M and N. The matrix of size M x N is generated with random values which is sparse in nature. Here the values of M and N are chosen to be multiples of number of SPE's in order to assign equal computational load to each SPE. The processor specifications used in the experiment are shown in Table 6.1.

Processor	Intel Core2 Duo	Cell Broadband Engine
Cores	2	1+8
Clock	2.00 GHz	3.2 GHz
Memory	2GB DDR2	512 GB XDRAM
OS	Fedora Linux Kernel	Red Hat Enterprise Linux® 5.2
Compiler	gcc 4.3.1	spu-gcc

Table 6.1 Processor specifications

6.2.1 Speedup for Parallel Householder Bidiagonalization

Table 6.2 provides the CPU vs. CBE execution time for parallel householder bidiagonalization as a function of the size of the randomly generated input matrix A. The difference in runtime between the serial and the parallel versions on Intel Core 2 Duo and CBE respectively increases significantly by increasing the size of the input matrix, and it reaches a maximum for M=1024:N=1024, where the parallel version is about 21 times faster than the serial version.

Size of Matrix (MxN)	**Sequential Algorithm (Time in sec)**	**Parallel Algorithm (Time in sec)**	**Speedup**
64 *x* 64	0.21	0.24	0.86
128 *x* 256	5.45	2.48	2.20
256 *x* 512	48.20	6.47	7.44
512 *x* 512	151.64	14.45	10.56
512 *x* 768	264.07	21.39	12.17
768 *x* 1024	793.78	45.83	17.32
1024 *x* 1024	1236.48	58.16	21.26

Table 6.2: Timing analysis of serial Householder Reduction on Intel Core2 Duo and parallel Householder Reduction on Cell BE

The increase in speedup with increase in M was due to the fact that, in Householder transformations, for smaller dimensions the number of iterations after which to redistribute the resultant matrices were comparable to M which lead to a large idle time for some SPU's. The problem of synchronization in this algorithm was dealt with the use of the efficient Signal Notification Registers giving much better performance than the use of mailboxes.

The results show that the performance of parallel SVD on CBE degrades for the matrix size 64 *x* 64. This is due to the communication cost between SPE and main memory is greater than the computation cost. This can be dealt by deploying less number of SPE threads for computation.

Figure 6.1 compares the time required for sequential and parallel Householder bidiagonalization. Figure 6.2 suggests an increase in speedup as the number of rows M and the number of columns N were increased.

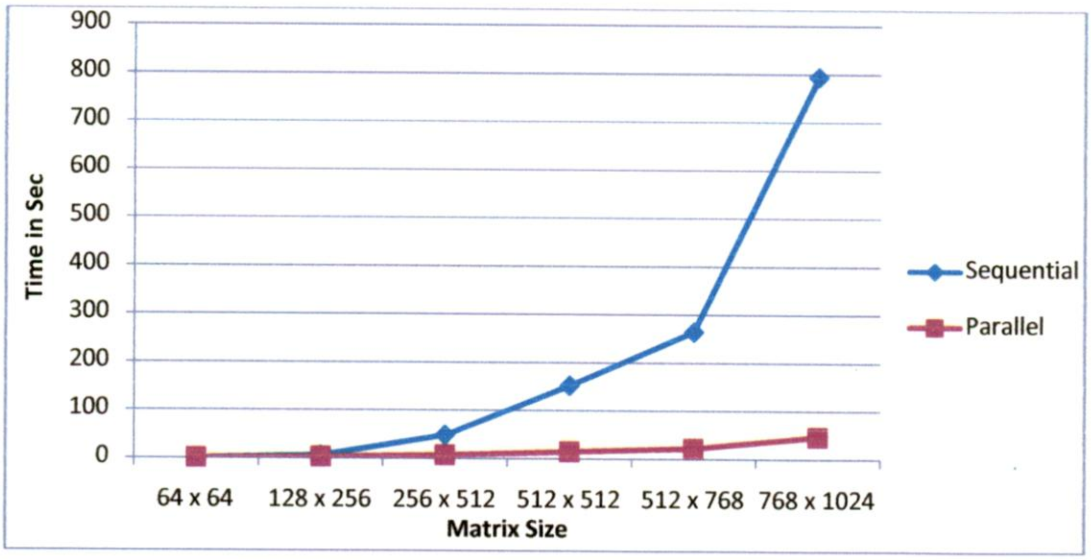


Figure 6.1: Time required for serial and parallel Householder Bidiagonalization

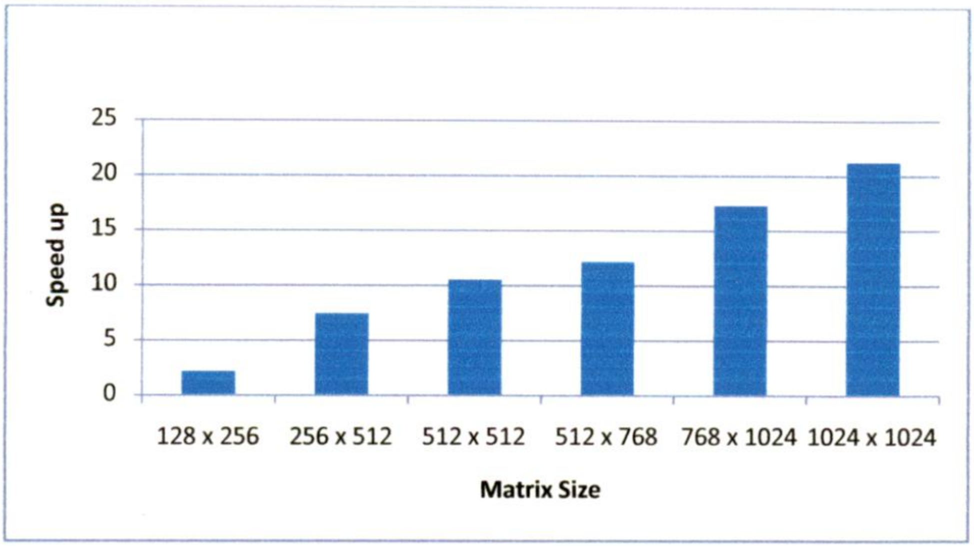


Figure 6.2: Speedup of cell BE vs. Intel Core2 Duo for varying Matrix sizes

6.2.2 Speedup for complete SVD algorithm

Table 6.2 provides the CPU vs. CBE execution time for parallel SVD as for randomly generated input matrix A. The difference in runtime between the serial and the parallel versions on Intel Core 2 Duo and CBE respectively increases significantly by increasing the size of the input matrix, and it reaches a maximum for M=1024:N=1024, where the parallel version is about 8 times faster than the serial version.

Size of Matrix (MxN)	Sequential SVD (Time in sec)	Parallel SVD (Time in sec)	Speedup
64 x 64	0.561	0.636	0.88
128 x 128	4.75	3.044	1.56
256 x 256	38.57	12.362	3.12
512 x 512	317.64	72.68	4.37
768 x 768	1149.47	189.99	6.05
1024 x 1024	2802.33	323.22	8.67

Table 6.2: Timing analysis of serial SVD on Intel Core2 Duo and parallel SVD on Cell BE

Figure 6.3 compares the time required for Sequential and parallel SVD. Figure 6.4 suggests an increase in speedup as the number of dimensions M and the number of samples N were increased.

It can be observed that the increase in speedup saturates at higher values of M because the data size in the SPEs exceeds 256KB and the computations are carried out by accessing the PPU memory multiple times which increases the memory latency due to greater DMA stalls.

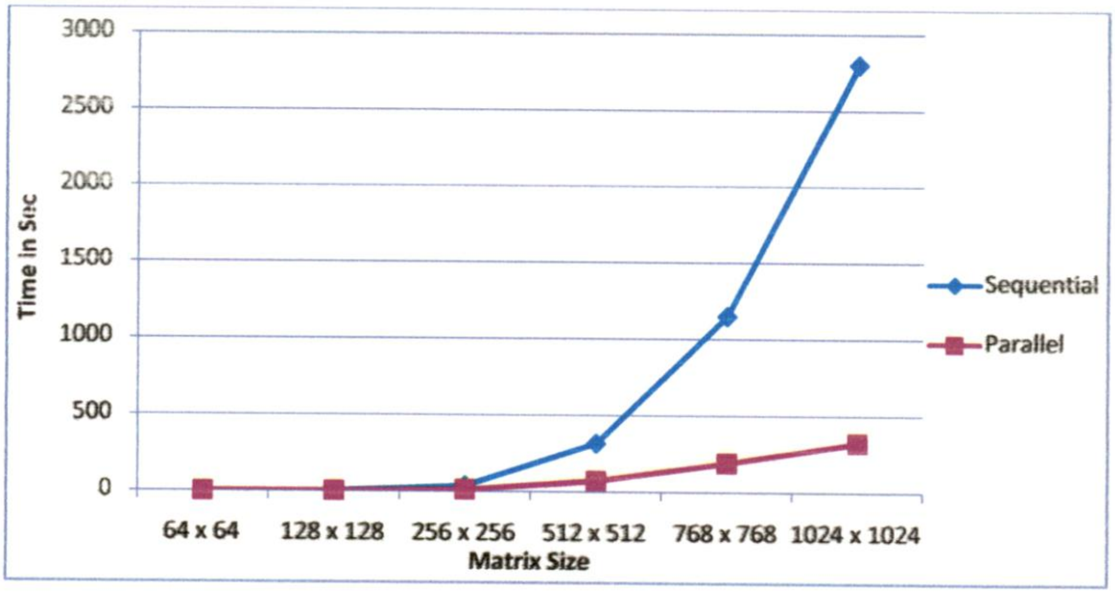


Figure 6.3: Time required for serial and parallel SVD

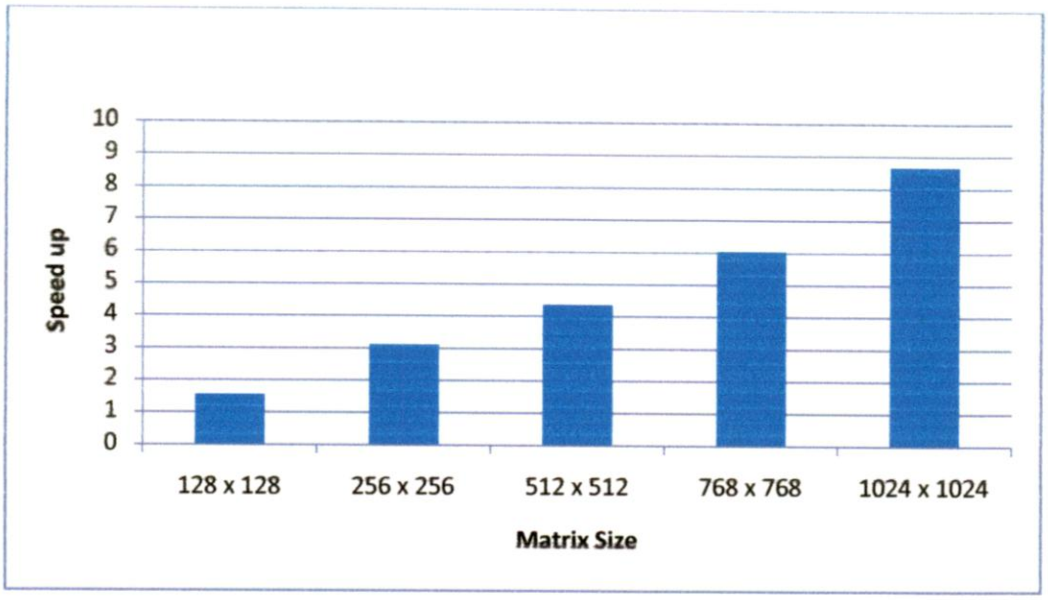


Figure 6.2: Speedup of cell BE vs. Intel Core2 Duo for varying Matrix sizes

CONCLUSION CHAPTER 7

In this work, we have proposed a parallel model for implementation of SVD computation on the CBE architecture. This model could be efficaciously used for LSI technique which is very popular method for information retrieval.

Parallelizing SVD was the most interesting and challenging task. We tried to resolve the various issues related to implementation of the algorithm and then proposed solutions to them. Due to the high level of task parallelism achieved, and through the use of SIMD vectorization, significant performance improvement is obtained by the multicore implementation on CBE processor over Intel Core 2 Duo. This leads to give about 8 times speedup over sequential SVD.

In comparison to a distributed system having multiple processing elements, the Parallel SVD implementation on the CBE processor achieves better results because of greater memory bandwidth and decreased time for interprocessor communication. In addition, CBE based implementation is quite cheap in comparison to distributed implementation.

Though the current implementation of SVD on cell is resulting significantly well yet there is scope of better performance. As an extension to this work, the better parallel strategy can be defined for diagonalization of bidiagonal matrix to achieve the maximum speedup.

REFERENCES

[1] Rosario, B, Latent Semantic Indexing: An overview. *INFOSYS* 240 (Spring 2000)

[2] Kathryn Parsons, Agata McCormac, Marcus Butavicius, Simon Dennis and Lael Ferguson "The Use of a Context-Based Information Retrieval Technique" *Technical Report*, DSTO, 2009

[3] Wall, Michael E., Andreas Rechtsteiner, Luis M. Rocha."Singular value decomposition and principal component analysis" in *A Practical Approach to Microarray Data Analysis*. D.P. Berrar, W. Dubitzky, M. Granzow, eds. pp. 91-109, Kluwer: Norwell, MA (2003). LANL LA-UR-02-4001.

[4] Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A. "Indexing by latent semantic analysis." *Journal of the Society for Information Science, 41(6)*, 391-407, 1990.

[5] http://en.wikipedia.org/wiki/Singular_value_decomposition

[6] "Cell Broadband Engine - An Introduction", Cell Programming Workshop, *IBM Systems and Technology Group*, April 14-18, 2007

[7] Cell Broadband Engine Programming Tutorial v2.0 http://moss.csc.ncsu.edu/~mueller/cluster/ps3/CBE_Tutorial_v2.0_15December2006.pdf

[8] Cell Broadband Engine programming handbook, Version 3.0, April 2006. htts://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/FC857AE550F7EB83872571A80061F788/$file/CBE_Programming_Tutorial_v3.0.pdf

[9] http://cellspe-tasklib.sourceforge.net/

[10] http://www.kernel.org/pub/linux/kernel/people/geoff/cell/ps3-linux-docs/ps3-linux-docs-08.06.09/CellProgrammingTutorial/AdvancedCellProgramming.html

[11] Christopher D. Manning and Hinrich Schütze, "Foundations of Statistical Natural Language Processing", ISBN 0262133601, 620 pp, Jun 1999.

[12] virginia c. Klema, alan j. Laub, "The Singular Value Decomposition: Its Computation and Some Applications", *IEEE transactions on automatic control*, vol. Ac-25, no. 2, April 1980

[13] Forsythe, G. E., and P. Henrici: "The cyclic Jacobi method". Trans. Amer. Math. Soc. 94, 1–23 (1960).

[14] Gene H. Golub and Charles F. Van Loan. Matrix Computations. John Hopkins University Press, Baltimore and London, 2nd edition, 1993.

[15] Volker Strumpen, Henry Hoffmann, and Anant Agarwal, "A Stream Algorithm for the SVD", *Technical Memo*, MIT-LCS-TM-64, October 22, 2003

[16] Magnus R. Hestenes, "Inversion of Matrices by Biorthogonalization and Related Results" *Journal of the Society for Industrial and Applied Mathematics*, 6(1):51-90, March 1958.

[17] Gene H. Golub and Christian Reinsch. "Singular Value Decomposition and Least Square Solutions" In J. H. Wilkinson and C. Reinsch, editors, *Linear Algebra,* volume II of Handbook for Automatic Computations, chapter I/10, pages 134-151. Springer Verlag, 1971.

[18] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannary: "Numerical Recipes in C", Cambridge University Press, New York, 1997.

[19] J. Kurzak and J.J. Dongarra, "QR Factorization for the CELL Processor," *J. Scientific Programming,* special issue on high performance computing on CELL B.E. processors, pp. 1-12, 2008.

[20] http://en.wikipedia.org/wiki/Bidiagonal_matrix

[21] B. Grosser and B. Lang, "Efficient Parallel Reduction to Bidiagonal Form," *Parallel Computing,* vol. 25, no. 8, pp. 969-986, 1999.

[22] N. Bosner and J.L. Barlow, "Block and Parallel Versions of One-Sided bidiagonalization," *SIAM J. Matrix Analysis and Applications,* vol. 29, no. 3, pp. 927-953, 2007.

[23] http://en.wikipedia.org/wiki/Diagonal_matrix

[24] James Demmel, W. Kahan, "Accurate Singular Values of Bidiagonal Matrices", *SIAM Journal on Scientific and Statistical Computing*, vol. 11, no. 5, pp. 873 – 912, Sept. 1990.

APPENDIX A: DEFINITIONS OF SOME MATHEMATICAL TERMS

1. Orthogonal Vectors

Two vectors u, v ϵ R^n are orthogonal if

$$u \cdot v = u^T v = u_1v_1 + \cdot\cdot\cdot + u_nv_n = 0$$

Note that $u \cdot u = u_1u_1 + \cdot\cdot\cdot + u_nu_n = ||u||^2$. If u is normalized, then $u \cdot u = 1$.

2. Orthogonal Matrix

A matrix Q is orthogonal if all row vectors are pair wise orthogonal, in other words

$$QQ^T = Q^TQ = I$$

In this case, $Q^T = Q^{-1}$

The product of two orthogonal matrices Q_1, Q_2 is also an orthogonal matrix

$$(Q_1Q_2)(Q_1Q_2)^T = Q_1Q_2Q_2^TQ_1^T = Q_1IQ_1^T = Q_1Q_1^T = I.$$

3. Bidiagonal Matrix

Matrix A is upper bidiagonal if a (i , j)=0 unless i=j or i=j-1. Matrix A is lower bidiagonal if a(i,j)=0 unless i=j or i=j+1.

4. Diagonal Matrix

An m x n matrix M is diagonal if $M_{ij} = 0$ for all $i \neq j$. The remaining entries may or may not be non-zero.

5. Numerical Stability

Suppose we have some mathematically defined problem represented by *f* which acts on data $d \in \Phi$ = some set of data to produce a solution $f(d) \in \Psi$ =some set of solutions. An algorithm to determine *f(d)* is numerically stable if the computed solution is near the solution of a slightly perturbed problem. More precisely, let f^* denote an algorithm used to implement or approximate *f*, then it is stable if for all $d \in \Phi$ there exists d^* near *d* such that $f(d^*)$ is near $f^*(d)$.

6. Convergent

Matrix A is convergent if A^k tends to 0 as k tends to infinity.

7. Givens Rotation

A *Givens Rotation* is a n*n matrix of the form P^T [Q 0 ; 0 I] P where P is a permutation matrix and Q is a matrix of the form [cos(x) sin(x); -sin(x) cos(x)].