

**T.C.  
BAHÇEŞEHİR UNIVERSITY**

**DESIGN OF PARALLEL PAGERANK ALGORITHM**

**M.S. Thesis**

**Murat HAKSAL**

**Istanbul, 2011**



**T.C.**  
**BAHCESEHIR UNIVERSITY**  
**The Graduate School of Natural and Applied Sciences**  
**Computer Engineering**

Title of the Master's Thesis : Design of Parallel PageRank Algorithm

Name/Last Name of the Student : Murat HAKSAL

Date of Thesis Defense : 09.09.2011

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc. Prof. F. Tunç BOZBURA  
Acting Director

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members:

Assoc.Prof. ADEM KARAHOCA (Supervisor) :

Asst.Prof. Y.BATU SALMAN :

Asst.Prof. ALPER TUNGA :

# ABSTRACT

## DESIGN OF PARALLEL PAGERANK ALGORITHM

Haksal, Murat

Computer Engineering Graduate Program

Supervisor: Assoc. Prof. Adem Karahoca

May 2011, 30 pages

The purpose of this thesis is to encode a parallel software which will compute PageRank values efficiently for large scaled networks. Such a parallel computation will be able to perform computation for multi-core and multi-processor hardware structures.

The purpose of this thesis, encoding and actual results of parallel PageRank computations are presented: CPU, memory, and I/O characteristics of the computations are presented for .Net platform. A computational profile of various steps of the algorithm is provided: Resource consumption during data-read, PageRank computations, and results persistence is presented. The performance of parallel computation is presented by contrasting various figures between sequential computation and parallel computations with various numbers of Windows threads.

Microsoft .NET and specifically C# programming language were used for software implementation of Parallel PageRank Algorithm. A new library arranging programming operation in multi-core hardware structures was supplied for the latest version of Microsoft .NET platform (4.0). This library was used for parallelization works in software.

Microsoft has recognized the importance of parallelism, and has incorporated various technologies into its solutions spectrum: Task Parallel Library for multi-core computation, and Message Passing Interface employed under HPC Server can be utilized to parallelize computations. This thesis used Task Parallel Library to implement a multi-threaded extension of PageRank algorithm. The language of choice was C#.

**Key Words:** Link Analysis, Parallelization for multi-core, .Net Thread Parallel Library

# ÖZET

Paralel PageRank Algoritma Tasarımı

Haksal, Murat

Bilgisayar Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Doç. Dr. Adem Karahoca

Mayıs 2011, 30 Sayfa

Bu tezin amacı, büyük ölçekli şebekeler(network) için, PageRank değerlerini verimli olarak hesaplayacak paralel bir yazılımı kodlamaktır. Bu tür bir paralel hesaplama, çok-çekirdekli ve çok-işlemcili donanım mimarileri için hesaplama yapabilecektir.

Bu tez kapsamında paralel PageRank algoritma tasarımı, paralel PageRank hesaplamaları kodlaması ve hesaplama için gerçek sonuçlar sunulmaktadır. Kaynak tüketimi ve veri okuma, PageRank hesaplamaları sırasında sonuçları detaylı sunulmuştur: farklı yaklaşımlar ile algoritmanın hesaplama profili sağlanmıştır. .Net platformu için CPU, bellek ve hesaplamaların I/O özellikleri sunulmaktadır: Sıralı ve paralel hesaplamalar için arasında hesaplama performansı Microsoft windows işletim sistemi kanallarının (Microsoft Windows threads) sayıları ile sunulmaktadır.

Paralel PageRank Algoritmasının yazılım implementasyonu için Microsoft .NET platformu, ve spesifik olarak da C# programlama dili kullanılmıştır. Microsoft, .NET platformunun en son versiyonunda (4.0), çok-çekirdekli donanım mimarilerinde programlamayı düzenleyen yeni bir kütüphane tedarik etmiştir. Bu kütüphane, yazılımdaki paralelizasyon işleri için kullanılmıştır.

Microsoft paralellik önemini kabul etmektedir ve paralellik için çeşitli teknolojiler geliştirmiştir: çok-çekirdekli hesaplamalar için Task Parallel Library ve HPC Server altında çalışan Message Passing Interface (MPI) ile hesaplamaları paralelizasyonu için kullanılabilir. Bu tez, PageRank algoritması multi-threaded bir uzantısı uygulamak için Task Parallel Library kullanılmıştır. Tercih edilen programlama dili C # 'tır.

**Anahtar Kelimeler:** Link Analizi, Çok çekirdekli ve çok işlemcili donanım mimarileri için paralelizasyon, .Net Thread Parallel Library

## TABLE OF CONTENTS

<b>LIST OF TABLES</b> .....	v
<b>LIST OF FIGURES</b> .....	vi
<b>ABSTRACT</b> .....	ii
<b>ÖZET</b> .....	iii
<b>1. INTRODUCTION</b> .....	<b>1</b>
<b>2. BACKGROUND</b> .....	<b>3</b>
2.1 WHAT IS PAGERANK? .....	3
2.2 PARALLEL COMPUTING .....	4
2.2.1 Data Partioning (Data Parallelism) .....	5
2.2.2 Functional Partitioning/Decomposing (Task Parallelism) .....	6
2.3 PREVIOUS STUDIES .....	7
2.4 SOCIAL RELATION REPRESENTATION .....	10
2.5 APPLICATION REQUIREMENTS .....	11
<b>3. MATERIAL AND METHODS</b> .....	<b>13</b>
3.1 DATASET .....	13
3.2 ALGORITHM .....	16
<b>4. FINDINGS</b> .....	<b>26</b>
<b>5. CONCLUSION</b> .....	<b>29</b>

**REFERENCES**

**CV**

## LIST OF TABLES

<b>Table 2.1:</b> Binary link data structure of Twitter dataset.....	7
<b>Table 2.2:</b> Block based data structure of Twitter dataset.....	8
<b>Table 2.3:</b> Split-Accumulate Data structure of Twitter dataset. ....	8
<b>Table 3.1:</b> Data structure of Twitter dataset.....	12
<b>Table 3.2:</b> Data structure of Twitter dataset adapted for algorithm.....	14
<b>Table 4.1:</b> Sequential And Parallel PageRank Calculation Time Cost.....	28
<b>Table 4.2:</b> PageRank Calculation Memory Cost.....	28
<b>Table 5.1:</b> Sequential And Parallel PageRank Calculation Time Cost.....	30

## LIST OF FIGURES

<b>Figure 2.1:</b> Data Partitioning.....	5
<b>Figure 2.2:</b> Task Partitioning.....	6
<b>Figure 2.3:</b> Social relation between Twitter user.....	10
<b>Figure 3.1:</b> PageRank calculation workflow using sqquential.....	15
<b>Figure 3.2:</b> PageRank calculation workflow using parallel.....	19
<b>Figure 4.1:</b> Cost Distribution.....	27
<b>Figure 4.2:</b> Thread Workload.....	27
<b>Figure 4.3:</b> Thread Workload Contentions.....	28



## 1. INTRODUCTION

The goal of this thesis is to provide a parallel extension of PageRank algorithm that is used to compute the relative values of nodes in a large network. Such an extension would allow efficient and scalable computation for very large datasets.

Applications of PageRank algorithm and its extension are numerous: Google uses it to rank the webpages in internet for efficient information retrieval. Meanwhile, it can be virtually used in every domain where there is a network of entities, and attaching some sort of relative importance to each node carries some value. A striking example is attaching “network values” to subscribers in a Telecom network. Churn management and cross-selling are important applications in Telecom industry (Ahn H. et al, 2010). If a network value can be attached to each subscriber (a node), it can be utilized to determine valuable subscribers, or it can be utilized to communicate a marketing message to most valuable subscribers in the network. Hence, network values a computed by PageRank algorithm and its variants have a number of practical applications in various industries where a set of entities heavily interact with each other.

PageRank algorithm is based on the key heuristic that the “value” of a node in a network is proportional to the sum of the node values that are connected. This heuristic eventually gives rise to an eigen-vector computation of a large sparse matrix. In a large network, storage of “connection matrix”, and iterative computation of eigen-vector can be a daunting task. If the number of new entrants into the network is large (in-flow) or the number of deserters is large (out-flow), or the number of emerging/dying connections is large; efficient computation of network values become indispensable. “Efficient computation” in this sense equals to parallel computation of network values in various parallelism scenarios; data-parallelism, multicore parallelism and distributed parallelism. This thesis presents algorithms and implementations of data parallelism and multicore parallelism by employing Microsoft technologies.

Microsoft has recognized the importance of parallelism, and has incorporated various technologies into its solutions spectrum: Task Parallel Library for multi-core

computation, and Message Passing Interface employed under HPC Server can be utilized to parallelize computations. This thesis used Task Parallel Library to implement a multi-threaded extension of PageRank algorithm. The language of choice was C#.

The methods are the PageRank algorithm implementations of data parallelism and multicore parallelism by employing Microsoft technologies.

The thesis is structured as follows: The first section presents the basic concepts revolving around the parallelization of PageRank algorithm. It describes the “vanilla” PageRank algorithm, the basic heuristic behind the algorithm, the set of mathematical equations derived from this heuristic, the computational requirements (CPU, I/O, storage), and a literature review of PageRank computations.

In the second sections, data structures that were used in the computations are described in detail. Partitioning of data for parallel computation lies at the core of large scale parallelization of many algorithms, and PageRank is no exception. This section describes the way data is partitioned before it is read into the memory for subsequent computation.

The third section presents the details of the PageRank algorithm that has been actually employed to compute the network values. The methodology behind parallelization, and the technologies supporting this methodology is provided in detail

In the fourth section, actual results of parallel PageRank computations are presented: CPU, memory, and I/O characteristics of the computations are presented. A computational profile of various steps of the algorithm is provided: Resource consumption during data-read, PageRank computations, and results persistence is presented. The performance of parallel computation is presented by contrasting various figures between sequential computation and parallel computations with various numbers of Windows threads.

Finally, the conclusion section presents various future extensions of the algorithms and methodologies presented in this thesis: The distributed, and temporal extensions are of primary importance.

## 2. BACKGROUND

### 2.1 WHAT IS PAGERANK?

The PageRank algorithm defines the PageRank value of a node in a network as a linear combination of the PageRank values of the nodes related to that particular node. The algebraic formulation of such a definition is as follows:

$$PR(u) = (1 - d) + d \sum_{v \in B(u)} \frac{PR(v)}{N_v} \quad (1.1)$$

In the Formula 1.1,  $u$  represents a node in the network; whereas  $B(u)$  represents the node set related to that particular node.  $PR(u)$  is the PageRank value of the node  $u$ . The  $d$  parameter in the formula is defined as a damping factor.

The Formula 1.1 equation system can be briefly stated as follows:

$$Ap = p \quad (1.2)$$

In the Formula 1.2 equation, the  $A$  matrix represents the connection between nodes, whereas  $p$  is the vector containing the PageRank values. Such computation of the PageRank value is the computation of the eigen-vector (Marcus & Minc 1988) value corresponding to a specific eigen-value (value of "1"). For a large-scale network, this

computation is performed iteratively: A random PageRank value is assigned, and the following iteration is repeated until the norm difference between consecutive PageRank vectors is omissible:

$$p_{n+1} = Ap_n \quad (1.3)$$

The damping factor ensures convergence of the PageRank vector.

## 2.2 PARALLEL COMPUTING

Parallel computing depends on the premise that large-scaled computations can be divided into small computations and calculated simultaneously (Almasi, G.S. & Gottlieb A. 1989). In recent years, parallel computing has become an important issue thanks to the multi-core and/or multi-processor hardware structures which are a direct result of the developments in hardware structures. (Asanovic et al 2006). Parallel computing uses the data parallel model (data) parallelization and task parallel model (task) approaches for multi-processor and/or multi-core hardware structures. There is one task and multiple data for the data parallel model while there are multiple tasks and multiple data. The first step of the parallel program design is to separate the problem into chunks. These chunks should be distributable to multiple processes. This process is known as decomposition/partitioning.

Parallel computing consists of four basic steps(Trobec R et al 2009, p.13):

- partitioning (decomposition to a maximum number of concurrently executable tasks)
- communication analysis (evaluating the amount of communication among tasks)
- granularity control (reduction of communication requirements by agglomeration)
- mapping (assigning tasks to processors of the model)

Matching the processes with real processors (mapping) Load balancing is the most important issue in parallel computations. It is important to ensure that after the work

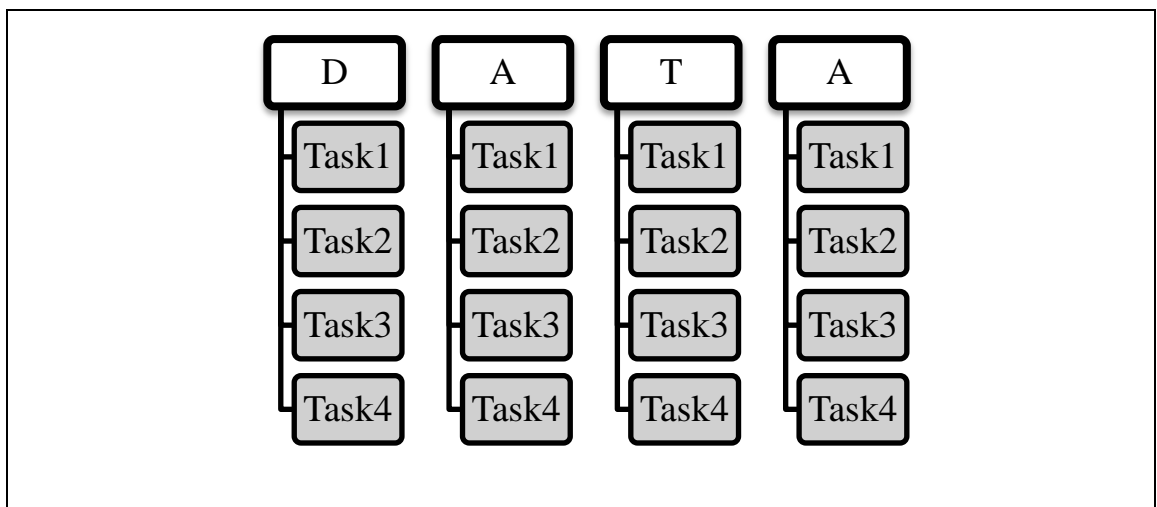
has been separated into smaller tasks, that these tasks are the same as or similar to each other in terms of work load. Otherwise, unwanted results may be detected due to poor parallelization, unequal processes and idle process(es).

There are two ways to make decomposed computation in parallel processes:

- Data set (domain) partitioning/decomposition
- Functional partitioning/decomposition

### 2.2.1 Data Partioning (Data Parallelism)

Data belonging to the problem is separated into chunks. Therefore, each parallel process works on a certain chunk of data.

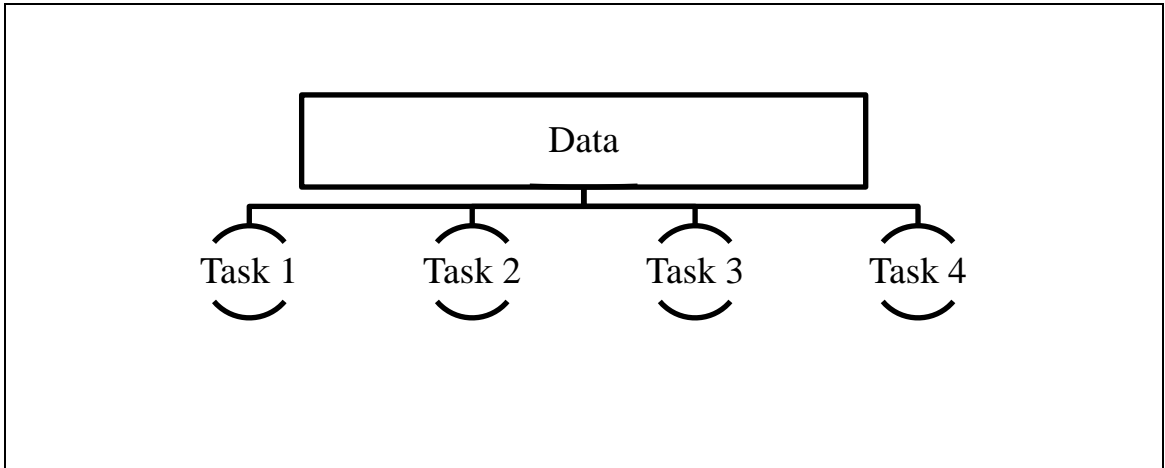


**Figure 2.1: Data Partioning**

We can decompose the data set for data partitioning horizontally and/or vertically. In a matrix data set, we can make line based, column based and line and column based decompositions. This choice depends on the problem to be computed.

### 2.2.2 Functional Partitioning/Decomposing (Task Parallelism)

Work belonging to the problem is separated into chunks. Therefore, each parallel process works on a certain chunk of the work.



**Figure 2.2: Task Partitioning**

The approach to be used for decomposing the work into chunks relates to the problem to be computed.

### 2.3 PREVIOUS STUDIES

The distributed computation (PC-Cluster) method for the computation of the PageRank algorithm is detailed. Memory utilization, I/O utilization and synchronization costs are compared regarding data-partitioning for parallelization, using 3 different techniques, and the findings are presented. An Efficient Partition-Based Parallel PageRank Algorithm (Manaskasemsak B. & Rungsawang A. 2005)

Binary Link Structure file was used for data structure.

- src\_id (4 bytes)
- out\_degree (4 bytes)
- dest\_id (4 bytes each)

**Table 2.1: Binary link data structure of Twitter dataset.**

Src_id	Out_degree	Dest_id
1	4	9 102 256 324
2	5	3 178 203 278 345
3	5	5 10 196 313 335
4	3	2 285 299
...	...	...

Three detailed approaches for data partitioning;

Partition-based parallel PageRank algorithm: All network data are equally partitioned based on the number of processors, and then sequentially assigned to each processor based on their respective source id's (src\_id). PageRank values of all Destination Id's (dest\_id) that are assigned a Source Id (src\_id) are computed. (Table 1)

PageRank values of each associated node (dest\_id) are computed and assigned to the relevant PageRank vector. These values are re-computed in each iteration.

Block Based Algorithm: The entire network is equally partitioned, and then Source Id (src\_id) and Destination Id (Dest\_id) orders are held with the number(num) value of this order. Then, these are assigned to the processor relevant to this order. Destination Id's are computed with a Source Id (src\_id) dependent order.

**Table 2.2: Block based data structure of Twitter dataset.**

Src_id	Out_degree	Num	Dest_id
1	4	1	9
2	5	1	3
...	...	...	..

Src_id	Out_degree	Num	Dest_id
1	4	1	102
2	5	1	178
...	...	...	..

Split-Accumulate algorithm: The entire network is equally partitioned, and a Destination Id is sequentially assigned to each relevant processor. Source Id's (src\_id) are computed with a Destination Id dependent order.

**Table 2.3: Split-Accumulate data structure of Twitter dataset.**

Dest_id	In_degree	Src_id
1	2	11 12
2	3	4 8 13
3	1	2
...	...	...



As a result, processor number-dependent findings are presented for I/O cost, synchronization and memory utilization, using Partition-Base, Block-Based and Split-Accumulate approaches. Memory utilization, I/O utilization and synchronization findings for Partition-base yielded the best results.

Asynchronous PageRank computation in an interactive multithreading environment The thread level parallelism method for the computation of the PageRank algorithm is detailed. The asynchronous approach was used to minimize synchronization costs, it was compared with synchronous computations and the findings were presented. (Kollias G.& Gallopoulos E. 2007).

Adjacency matrix, transition matrix and stochastic matrix were used as data structures.

The Jhyton framework was used for computations, instead of Java Virtual Machine. Jylab Framework was used for asynchronous computations. The threading domain model for pagerank computations was detailed.

The results yielded that the synchronization cost for the synchronous computation of the PageRank value (thread-join) was greater than the cost of asynchronous computation. Asynchronous implementations were detailed in order to reduce the mentioned costs.

Two basic approaches for PageRank computation, the EigenSystem and LinearSystem, were detailed; parallelization approaches were presented. Computations were performed using the PageRank Iterations method under EigenSystem approach, as well as Jacobi Iteration, GMRES (Generalize Minimum Residual), BICG (Biconjugate Gradient), BICGSTAB (Biconjugate Gradient Stabilized) methods under the Linear System approach, and the findings were presented for each of the mentioned five methods. (Gleich D. & Zhukov L. & Berkhin P. Fast Parallel PageRank: A *Linear System Approach* 2004).

Sparse matrix was used as the data structure.

The Portable, Extensible Toolkit for Scientific Computation (PETSc) was used for the computation of the abovementioned methods. Data-partitioning technique was used for

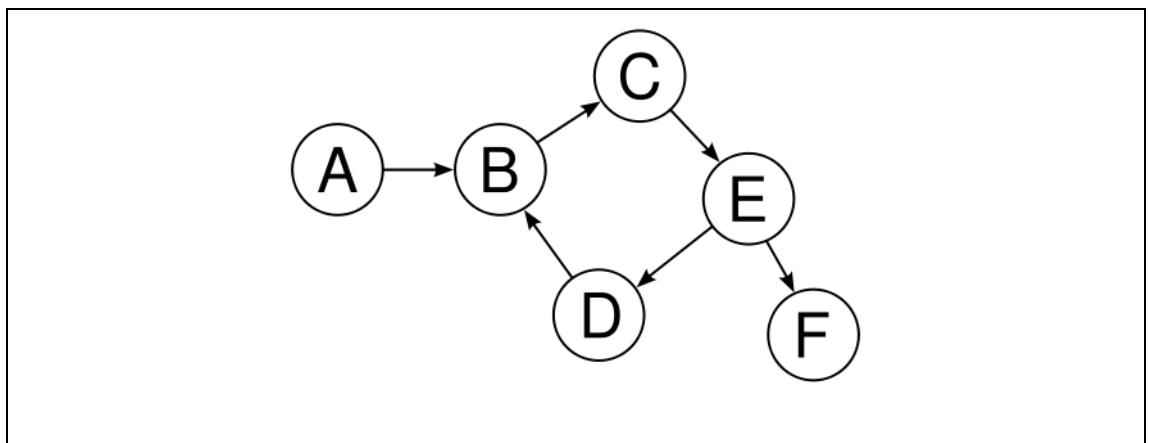
parallelization, the matrix was equally partitioned based on the number of processors, and computation of each equal partition was performed by PETSc.

The BiCGSTAB and GMRES approaches yielded the best results based on the iteration number and time cost criteria.

Focused Page Rank in Scientific Papers Ranking, The implementation of the PageRank algorithm was detailed for other scientific paper citations within the content of the scientific papers. The links given in the PR algorithm were re-formulated as citation counts. 266788 publications were used as the dataset. The findings revealed that there is a relationship between the quality of the given publication citations and the publication itself.

## 2.4 SOCIAL RELATION REPRESENTATION

Figure 2.3 demonstrates the relation between users of Twitter, one of the social networks. This relation is demonstrated from A node to B node ( $A \rightarrow B$ ).



**Figure 2.3: Social relation between Twitter user**

Demonstration of these nodes is a Directed Graph demonstration. Each node has a relation with another node for Directed Graph. This relation (edge) was processed in the data model as FromNode, ToNode. For PR(u), PR(v) values in the PageRank algorithm, PR(u) indicates PageRank value for FromNode and PR(v) indicates PageRank value for ToNode.

## **2.5 APPLICATION REQUIREMENTS**

### **2.5.1 Computing Requirements**

The most obvious requirement, particularly for scientific and technical applications, is the number of floating-point processes required for realizing the computation. Estimating this number for simple computations is relatively easy. Even a rough estimate is generally possible in complicated processes. It is possible to make predictions using books on numerical analyzes.

An example of computing requirements is given below. A 2 GHz processor can run a  $2 \times 10^9$  floating-point process in a second. For this processor, computing which requires 1 billion floating-point processes will take just half a second (Gropp W., Lusk E., and Sterling T., 2003). However, the aim in terms of general system performance should be considered when thinking about computing requirements.

*Memory:* Memory requirements of the computation influences the performance cost of application significantly. Memory should have a capacity which is sufficient to store all data required for the application. Cache memory is very important for the performance of some applications. In virtual memories, a part of disk space is used as memory. The whole of the virtual memory space can be accessed by the application. Virtual memories increase the capacity of memory with low costs. However, access time is much more than main memory since data is stored on disk. Therefore, virtual memories are not used for high performance computations.

*Input/ Output:* Results of the processes should be stored in permanent storage spaces, such as the files on disk. Parallel execution enables the processes to be made very rapidly. The performance of input / output system is predicted in proportion to this.

### 3. MATERIAL AND METHODS

#### 3.1 DATASET

##### 3.1.1 Datasource

Twitter dataset was used for the computation of the algorithm. Twitter dataset was obtained from Stanford Large Network Dataset Collection, and consists of 1.47 billion users (node/vertex) and 41.17 billion follower relations (node relation/edge). This dataset was chosen because it has the biggest network data in the data source. The large size of the data is significant in terms of providing details for computing performance matrixes. The size of the Twitter dataset is approximately 30 gigabyte in a 64 bit file system (File System).

##### 3.1.2 Data structure

Data in the dataset consists of two basic data: these are User and Follower data. User data indicates the identity value of the Twitter user. It demonstrates each node for algorithm. Follower is the identity value of the user who follows Twitter. For algorithm, data of node relating to each node represents a meaning. Data structure in the dataset is detailed in Table 2.1.

**Table 3.1: Data structure of Twitter dataset.**

Column Name	Data Type
User	Integer
Follower	Integer

### **3.1.3 Database**

The data structure received from the datasource was not suitable for the computation of algorithm and therefore had to be made suitable. The dataset received from the datasource was partitioned to provide scalability of dataset computation. The process of indexation of data arose from a need to provide access to data after partitioning(Powell, G. 2006, s.193-213). The database was used for indexation of dataset and performance optimizations of data access. Microsoft Sql Server 2008 was assigned as the database. Microsoft Sql Server Management Studio was used to transfer the said dataset to the database and to index it.

### **3.1.4 Data Preprocessing**

Data structure required by algorithm is the number of nodes, relating nodes and relating nodes of relating nodes. The data structure in the datasource was not suitable for algorithm. ETL techniques were used to make the data suitable for the data structure required by algorithm. Microsoft Sql Server 2008 Query Editor was used to apply ETL techniques.

The dataset was grouped in accordance with User data and the number of relating Follower datasets was received. Grouped dataset was associated with the Follower data of the present dataset, and the dataset was scanned and saved as a new dataset.

This process was costly in terms of memory, processor and time. Execution of this process during computation of algorithm influences the computation cost negatively due to large size of data. Therefore, it was executed only once. The data structure of the dataset is detailed in Table 3.2. Said structured data was used for the computation of the algorithm. Thus, data access for the computation of algorithm was reduced to data reading only.

**Table 3.2: Data structure of Twitter dataset adapted for algorithm**

<b>Column Name</b>	<b>Data Type</b>
FromNodeId	Integer
ToNodeId	Integer
OutGoing	Integer

### **3.1.5 Dataset Indexing**

The dataset received from the datasource was partitioned to provide scalability of dataset computing. The process of indexation of data arose from a need to provide access to data after partitioning (Powell, G. 2006, s.193-213). There are two basic approaches for indexing methodology: Clustered Index and NonClustered Index.

Clustered Index enables the dataset to be indexed physically. NonClustered Index is the indexation of physical address of data. Clustered Index was preferred for indexation of dataset. This is because, in the computation of algorithm, partitioned data is scanned sequentially and physically sorted data is required. The indexing process was configured according to User data. Indexation of large scaled data is a costly process in terms of time and source usage. Optimization of these costs was carried out by Microsoft Sql Server. For the Indexing process, parallelization techniques were used by Microsoft Sql Server and usage of memory and processor was optimized.

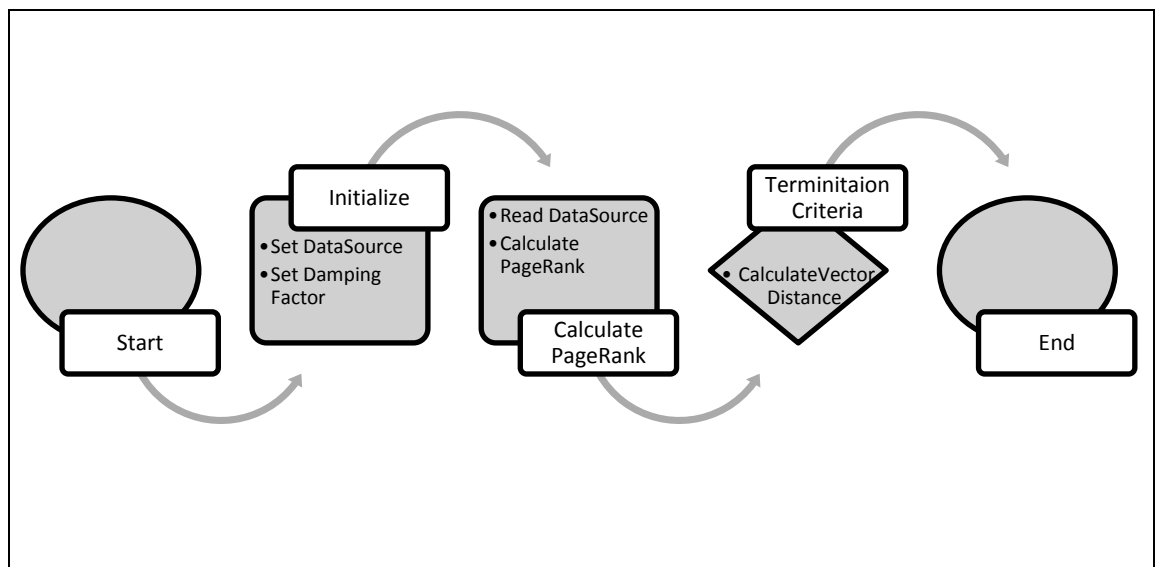
## 3.2 ALGORITHM

In the computation of PageRank, two algorithms were developed, namely sequential and parallel; and the algorithms were detailed with pseudo codes and work flows.

### 3.2.1 Sequential Algorithm

A sequential PageRank algorithm consists of four basic stages. These have been detailed as work flow in Figure 2.3 diagram. The initialize stage is the stage when algorithm parameters are received and parameters required for computation are prepared. In this stage, datasource appropriate to database connection, database and relating table (connection string) information is prepared. PageRankVector to be used in calculation content is created; data structure is Dictionary<int,float> type.

The stage after the initialize stage is ComputePageRank. In this stage, the PageRank value is calculated for each node and this value is assigned to the value of the relating node of *AfterPageRankVector*. After computations are completed for all nodes, distance value between *AfterPageRankVector* and *BeforePageRankVector* is shifted until it becomes lower than 0.001. This stage is the termination criteria of the algorithm.



**Figure 3.1:** PageRank Calculation Workflow Using Sequential



### Algorithm 2.1: Sequential PageRank Pseudo Code

1. Initialize
  - 1.1. SET Algorithm Parameters
    - 1.1.1. SET Algorithm Data Source=*TwitterRepository*
    - 1.1.2. SET Damping Factor
  - 1.2. SET PageRank Vector =*BeforePageRankVector*
2. Compute PageRank
  - 2.1. Do
    - 2.1.1. CALL *TwitterRepository.Read Grouped by FromNode* RETURNING *GroupedNodes*
    - 2.1.2. FOR each *groupNodes* of *GroupedNodes*
      - 2.1.2.1. For each *node* of *groupNodes*
        - 2.1.2.1.1. IF *firstIteration*
          - 2.1.2.1.1.1. SET *node* of *BeforePageRankVector*=1.0
          - 2.1.2.1.1.2. End If
        - 2.1.2.1.2. End For
      - 2.1.2.2. End For
      - 2.1.2.3. CALL *CalculatePageRankNode* with *BeforePageRankVector*, *groupNodes* and *DampingFactor* RETURNING *NewPageRankValue*
      - 2.1.2.4. SET *node* of *AfterPageRankVector* = *NewPageRankValue*
    - 2.1.3. End For
    - 2.2. WHILE CALL *CalculateVectorDistance* with *BeforePageRankVector* and *AfterPageRankVector* RETURNING *distanceValue*<0.001
    - 2.3. Enddo

#### 3.2.1.1 Initialize

Pseudo codes of the Algorithm 2.1 Sequential PageRank computation algorithm are given. The algorithm receives *datasource* and *damping factor* values as parameters. *BeforePageRankVector* is created in the initialize stage of the algorithm. Data structure is Dictionary<int,float> type for C# language.

### 3.2.1.2 Compute PageRank & Termination Criteria

In computation stage, an iteration is generated with Do-While (Algorithm 2.1: 2.1-2.3). PageRank values are computed for each node in iteration. For termination of iteration, distance of iteration PageRankVector values continuing with PageRankVector that is the result of previous iteration are computed; this value is shifted until it becomes lower than 0.001. The Formula 2.1 was used for computation of PageRank Vector distance.

$$D = \sqrt{\sum_{i=1}^N Y1_i - Y2_i^2} \quad (2.1)$$

Algorithm 2.1: Sequential PageRank computation algorithm is read from datasource database in each iteration content between lines 2.1 and 2.3. Read data is grouped in accordance with FromNode(User) and stored in memory (*GroupedNodes Line:2.1.1*). Groped data (*GroupedNodes*) is taken to iteration(*Line:2.1.2*). Iteration is started again for each node in accordance with each grouped data within iteration(*Line:2.1.2.1*). In initial iteration(*Line:2.1.2.1.1*), default value (1.0) is assigned for each node for *BeforePageRankVector* created in initialize stage(*Line:2.1.2.1.1.1*).

### 3.2.1.3 Calculate PageRank For Each Node

#### Algorithm 2.2: Calculate PageRank For Each Node Pseudo Code

1. METHOD *CalculatePageRankNode* with PARAMETERS: *BeforePageRankVector, GroupedNodes* and *DampingFactor*
  - 1.1. SET *SumOfPageRankValue=0*
  - 1.2. FOR each *node* of *groupNodes*
    - 1.2.1. SET *ToNodePageRankValue= BeforePageRankVector[node.ToNodeId]*
    - 1.2.2. SET *tempValue= ToNodePageRankValue / node.Outgoing*

```

1.2.3. SET SumOfPageRankValue += tempValue
1.3. End FOR
1.4. RETURNING ( $1 - \text{DampingFactor} + (\text{DampingFactor} * \text{SumOfPageRankValue})$ )
2. End METHOD

```

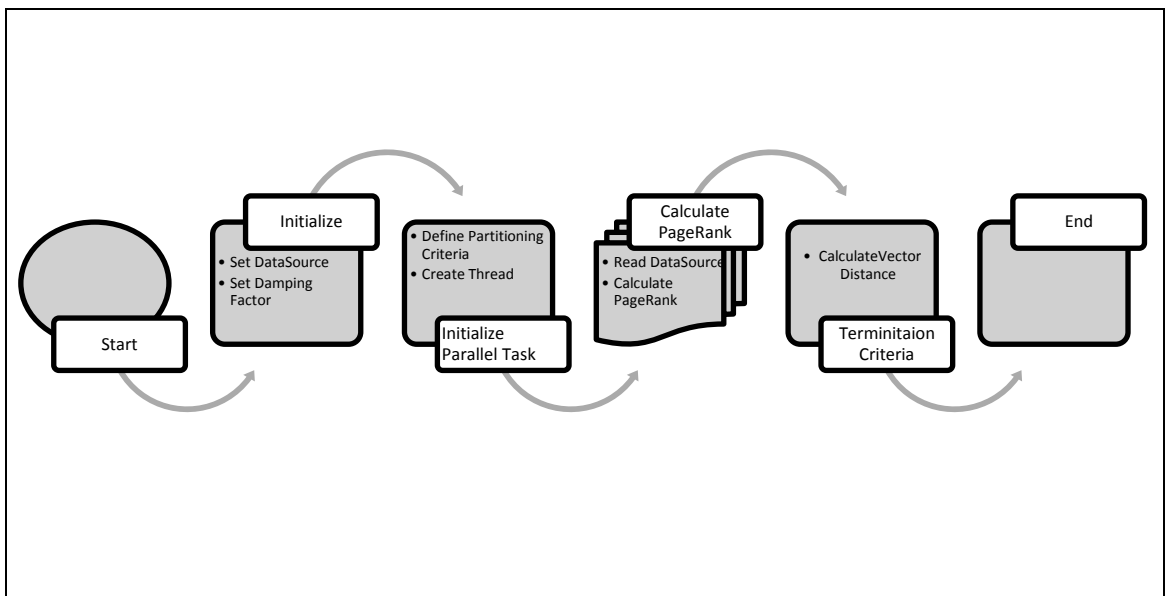
*CalculatePageRankNode* method is called for computation of PageRank value for each node (Line: 2.1.2.1.3); node in iteration is given as parameter with *BeforePageRankVector*. *CalculatePageRankNodeAlgorithm* 2.2 is detailed (Line:2.1.2.3). Result of *CalculatePageRankNode* method is assigned to PageRank value of node value related to *AfterPageRankVector* (Line:2.1.2.4).

The method receives three parameters for computation of PageRank value for each node in Algorithm 2.2. Vector retaining PageRank values computed in previous iteration (*BeforePageRankVector*) is given to node collection grouped in accordance with *FromNode(User)* and the *DampingFactor* method as a parameter. The method creates an iteration for grouped node collection (Line:1.2). In iteration content, PageRank value of relating node of each node grouped in accordance with PageRank algorithm is received from *BeforePageRankVector*(Line:1.2.1) and divided into the number of relating nodes of relating node (Line:1.2.2); result is summed for each grouped node (Line:1.2.3). Relating value for each grouped node is multiplied to *DampingFactor* value suitable for PageRank algorithm, summed with  $1 - \text{DampingFactor}$  and concluded as the result of the method (Line 1.4).

### 3.2.2 Parallel Algorithm

Parallel PageRank algorithm consists of five basic stages. These have been detailed as work flow in Figure 2.4 diagram. Initialize stage is the stage when algorithm parameters are received and parameters required for computation are prepared. In this stage, datasource appropriate to database connection, database and relating table (connection string) information is prepared. PageRankVector to be used in calculation content is

created; data structure is Dictionary<int,float> type. *DegreeOfParallelism* is taken as the parameter and this parameter demonstrates the degree of parallelism of algorithm. Different processor threads will be opened for partitioning data suitable for this value and computation. In Initialize Parallel Task stage, in accordance with the *DegreeOfParallelism* value given, the criteria regarding that data partitioning will be performed at which node intervals are determined by reading datasource. After data partitioning criteria are determined, inquiries suitable for datasource are prepared in this stage. Then, threads are opened by the processor suitable for *DegreeOfParallelism* value and *ComputePageRank* stage is started for each thread. Each thread computes a PageRank value for each node on partitioned dataset and this value is assigned to the value of relating node of *AfterPageRankVector*. In order to provide computation consistency, synchronization is provided between threads. After computations are performed for each grouped node, distance value between *AfterPageRankVector* and *BeforePageRankVector* is shifted until it becomes lower than 0.001. This stage is the termination criteria of algorithm.



**Figure 3.2:** PageRank calculation workflow using parallel

## Algorithm 2.2: Parallel PageRank Pseudo Code

1. Initialize
  - 1.1. SET Algorithm Parameters
    - 1.1.1. SET Algorithm Data Source=*TwitterRepository*
    - 1.1.2. SET Damping Factor= $0.85$
    - 1.1.3. SET Degree of Parallelism = *degreeOfParallelism*
  - 1.2. SET PageRank Vector =*BeforePageRankVector*
2. Initialize Parallel Task
  - 2.1. CALL *TwitterRepository.Read Partition by degreeIndex* RETURNING *partitioningCriteria*
  - 2.2. FOR *degreeIndex* of *degreeOfParallelism*
    - 2.2.1. CALL *newThreading of computeParallelPageRank with partitioningNodeCriteria*
  - 2.3. END FOR
3. Compute Paralel PageRank
  - 3.1. DO
    - 3.1.1. While(*INTERLOCKED.Read(IncrementalThreadingTaskValue) !=0*)
      - 3.1.1.1. Thread.Sleep(1000)
    - 3.1.2. EndWhile
    - 3.1.3. IF *IsFirst=false*
    - 3.1.4. SET *BeforePageRankVector =AfterPageRankVector*
    - 3.1.5. End IF
    - 3.1.6. CALL *TwitterRepository.PartitionRead Grouped by FromNodeWith partitioningNodeCriteria* RETURNING *GroupedNodesPointer*
    - 3.1.7. FOR each *groupNodes* of *GroupedNodesPointer*
      - 3.1.7.1. FOR each *node* of *groupNodes*
        - 3.1.7.1.1. *INTERLOCKED.INCREMENT IncrementalThreadingTaskValue*
        - 3.1.7.1.2. IF *node* of *BeforePageRankVector* Is Null
          - 3.1.7.1.2.1. SET *node* of *BeforePageRankVector*= $1.0$
        - 3.1.7.1.3. End IF
        - 3.1.7.1.4. CALL *CalculatePageRankNode* with *BeforePageRankVector* , *groupNodes* and *DampingFactor* RETURNING *NewPageRankValue*
        - 3.1.7.1.5. *INTERLOCKED.DECREMENT IncrementalThreadingTaskValue*
        - 3.1.7.1.6. WHILE *waitAllThreadsInComputation by self iterations*
          - 3.1.7.1.6.1. CALL *threadSleep with 10 milisecond*
        - 3.1.7.1.7. EndWHILE
        - 3.1.7.1.8. SET *node* of *AfterPageRankVector* = *NewPageRankValue*
      - 3.1.7.2. End FOR
    - 3.1.8. End For
  - 3.2. WHILE CALL *CalculateVectorDistance* with *BeforePageRankVector* and *AfterPageRankVector* RETURNING *distanceValue* <  $0.001$
  - 3.3. END DO

### 3.2.2.1 Initialize

Initialize stage pseudo codes of Algorithm 2.2-1. Parallel PageRank computation algorithm has been given. Algorithm receives datasource, damping factor value and parallelization number for computation parallelization as parameter. *BeforePageRankVector* is created in initialize stage of algorithm. Data structure is Dictionary<int,float> type for C# language. *DegreeOfParallelism* is taken as numerical parameter for parallelization of computation. *DegreeOfParallelism* default value is the total number of the processor core in hardware structure where computation will be performed.

### 3.2.2.2 Initialize Parallel Task

Initialize Parallel Task stage pseudo codes of Algorithm 2.2-2. Parallel PageRank computation algorithm has been given. In accordance with the *DegreeOfParallelism* value given as a parameter to algorithm, data number in datasource is taken, divided to *DegreeOfParallelism* value and it is determined approximately how many partitioned data will be received for each part. During this process, datasource is scanned; FromNode is partitioned in accordance with the number in *DegreeOfParallelism* parameter (Line:2.1). Within this stage, split is avoided for nodes(FromNode) for each partitioned data group. Data partitioning process is detailed in part 3.2.2.2.1 Data Partitioning. Outputs of data partitioning process are the starting and ending node values determining at which node intervals parts will be performed(E.g.: PartitionI: FromNodeId between 12 and 11457, PartitionII: FromNodeId 11457 and 45025,etc). Load balancing is attempted in data partitioning process. For this purpose, workloads of parts are very close to each other provided that there is no split. After data partitioning process is completed, iteration is started in accordance with the value of *DegreeOfParallelism* which is given. In each iteration content, thread is created for processor. Computation function and criteria of related partitioned dataset are given to each created thread and work assignments are identified. Threads that are created as a result of iteration assignments determining which partitioned datasets will be used for computation are carried out. Computation is started when iteration ends.

### **3.2.2.2.1 Data Partitioning**

In Manaskasemsak B. & Rungsawang A. , (2005) An Efficient Partition-Based Parallel PageRank Algorithm essay, memory usage, I/O usage and synchronization costs of three different data partitioning approaches have been compared and their findings are detailed. In Manaskasemsak B. &Rungsawang A. (2005) essay, findings for Partition-based parallel PageRank algorithm, Block Based Algorithm and Split-Accumulate algorithm techniques are provided. While the details of approaches in this essay are analyzed, problem identification appropriate to the result for which data partitioning approach gives the best result was carried out.

In accordance with the *DegreeOfParallelism* value given as a parameter to algorithm, data number in datasource is taken and divided to *DegreeOfParallelism* value, and it is determined approximately how many partitioned data will be received for each part. During this process, datasource is scanned; FromNode is partitioned in accordance with the number in *DegreeOfParallelism* parameter (Line:2.1). Within this stage, split is avoided for nodes (FromNode) for each partitioned data group. Data partitioning process has been detailed in part 3.2.2.2.1 Data Partitioning. Outputs of data partitioning process are the starting and ending node values determining at which node intervals parts will be performed (E.g.: PartitionI: FromNodeId between 12 and 11457, PartitionII: FromNodeId 11457 and 45025,etc). Load balancing was attempted in the data partitioning process. For this purpose, workloads of parts are very close to each other, provided that there is no split.

### **3.2.2.3 Compute PageRank & Termination Criteria**

In computation stage, an iteration is generated with Do-While (Algorithm 2.1: 3.1-3.3) This iteration is used for the fading function in PageRank algorithm. PageRank values are computed for each node in iteration separately. For termination of iteration, distance of iteration PageRankVector values continuing with PageRankVector that is the result of previous iteration are computed; these values are shifted until it becomes lower than 0.001. The Formula 2.1 was used for computation of PageRankVector distance.

$$D = \sqrt{\sum_{i=1}^N Y1_i - Y2_i}^2 \quad (2.1)$$

Algorithm 2.2: Parallel PageRank computation algorithm is the function which generates the enumerator pointer in order to receive data from datasource in accordance with the partitioning criteria determined in line 3.1.1 initialize stage. As a result of this function, iteration of enumerator point in line 3.1.3 is started. Data in this iteration is grouped in accordance with FromNode and given to the iteration in Line 3.1.7. In the iteration content in line 3.1.7.1, Interlocked class is used in order to provide synchronization between threads located in Microsoft.Net Threading library. Said class takes an integer statement whose data type is long and raises the present number, *IncrementalThreadingTaskValue* with Increment method by 1. This value indicates how many works have been completed at current time for threads and it is used for synchronization. Value given to Increment method starts from 0 and rises up to *DegreeOfParallelism* value. This means that there are as many threads as *DegreeOfParallelism* value in iteration content within t time. Provision of synchronization has been detailed in part 3.2.2.3.1. In Line 3.1.7.1.2, a check is carried out to determine if there is any value computed previously for node in iteration. If there is no assigned value, 1.0 assignment is carried out according to the related value of vector in Line 3.1.7.1.2.1 as default value. CalculatePageRankNode function is called in Line 3.1.7.1.4 and PageRank value is computed for each node. Assignment is carried out to the related node value of *AfterPageRankVector*. In order to provide synchronization between threads located in Microsoft.Net Threading library, Decrement function of Interlocked class is called and the number is decreased by 1 to indicate that related thread in iteration has completed the computation for that iteration. For this iteration, distance value between *AfterPageRankVector* and *BeforePageRankVector* is shifted until it becomes lower than 0.001. This stage is the termination criteria of algorithm.



### 3.2.2.3.1 Synchronization

It is a class which allows various processes to be performed on Interlocked class variables in Microsoft.Net Platform System.Threading library used by multiple threads. These variables may be value type or they can be any kind of object. Increment, Decrement and Exchange, Read methods are available in Interlocked class. Increment method takes a long data type as parameter and raises the given parameter value by one value. During this process, other threads cannot perform any change or read on these variables. Decrement method decreases the given parameter by one value. During this process, other threads cannot perform any change or read on these variables. Exchange method is used to assign the values of objects. During this process, other threads cannot perform any change or read on these variables. Read method returns the value of that parameter if there is no process performed on parameter. If there is a writing process at that moment, reading process can be pended from other threads and value result is given after the processes of other threads are completed.

During the computation of algorithm, results achieved after PageRank value in tasks partitioned for all nodes is parallel computed are assigned to the relating node value of relating PageRank vector after computation. During the computation of PageRank value of a node in PageRank algorithm, PageRank value of relating node is needed. If another iteration starts before an iteration finishes, two different PageRank values are generated in PageRank vector. Therefore, update of values belonging to the related vector is required after the computation of each partitioned task is completed. For this purpose, each task raises the number of completed tasks by one value at the beginning and decreases it by one value when it is completed with tasks numbers logic. Tasks in other threads are waited to be completed within the stage after the completion of initial iteration. Algorithm 2.2: In Parallel PageRank Pseudo Code 3.1.1 line, *IncrementalThreadingTaskValue* is expected to be 0 in case that relating tasks are completed. Related threads are put into sleep mode until it becomes 0 and checked again. After the computations of all threads are completed, computation finishes for related iteration and PageRank vector is updated. Algorithm 2.2: Parallel PageRank Pseudo Code 3.1.4 Line 3.1.4)

## 4. FINDINGS

In order to obtain the benchmarks between Parallel computation of PageRank Algorithm, Time Cost and Workload values were considered. Microsoft Visual Studio 2010 Performance Profiler was used to measure these values. Related computations detailed under efficiency Values are for the initial iteration of PageRank computation.

### 4.1 EFFICIENCY

In order to analyze the performance efficiency of algorithm, time cost, workload, cost distribution and memory approaches were detailed.

#### 4.1.1 Time cost

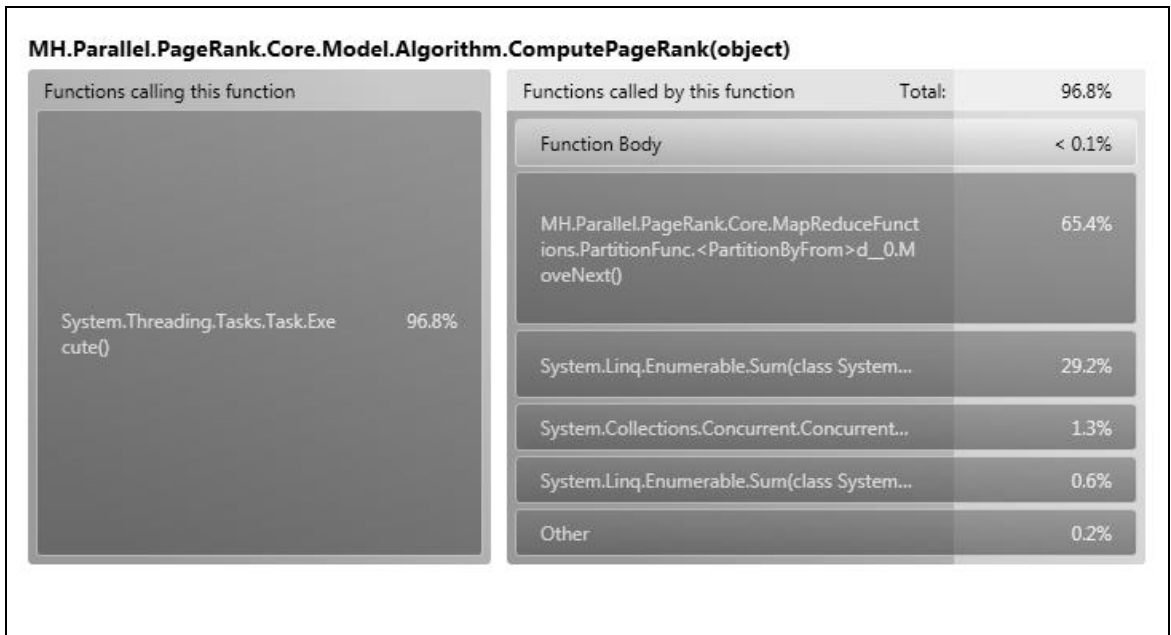
Sequential computation of PageRank algorithm achieved a 126.151 second time saving and parallel computation of algorithm achieved a 54.802 second time saving. Parallelization of algorithm realized an improvement of 43.44% time saving.

**Table 4.1 Sequential And Parallel PageRank Calculation Time Cost**

Approach	Time	Degree Of Parallelism
Sequential	126.151 Seconds	-
Parallel	54.802 Seconds	4

#### 4.1.2 Cost Distribution

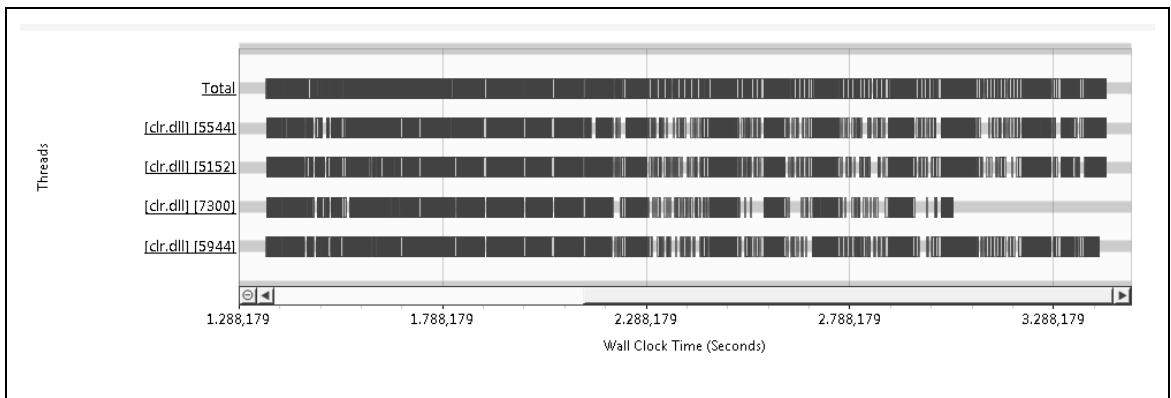
When analyzing cost distribution of Parallel PageRank algorithm, data reading and data partitioning form 65.4% of the whole cost. The part of 31.1% forms the calculation and assignment of data to thread isolation suitable for PageRank vector.



**Figure 4.1: Cost Distribution**





### 4.1.3 Workload

Workload distribution regarding 4 Threads used for Parallel algorithm was provided. Thread 1 no. 5152, Thread 2 no. 5544, Thread 3 no. 5944, and Thread 4 no. 73000. The last thread shows an indication.



**Figure 4.2: Thread Workload**

Four threads were used for computation of the Parallel PageRank algorithm. Said four threads are equal to the core number in the computation machine.

<b>Most Contended Threads</b>			
Threads with the highest number of contentions			
ID	Name	Contentions %	Contentions
5944	[clr.dll]	 26,88	7.339
5152	[clr.dll]	 26,67	7.282
5544	[clr.dll]	 26,47	7.229
7300	[clr.dll]	 19,97	5.454
7268	[MH.Parallel.PageRank.Console.exe]	0,01	2

**Figure 4.3:** Thread Workload Contentions

#### 4.1.4 Functions Allocating Most Memory

When memory usage of the Parallel PageRank algorithm is detailed, memory usage is 81.66% for data access and data partitioning, 8.12% for PageRank vector and 10.22% for PageRank vector.

**Table 4.2 PageRank Calculation Memory Cost**

Functions	Bytes %
DataAccess & Partioning	81.66%
PageRank Calculation	8.12%
PageRank Vector	10.22%

## 5. CONCLUSION

This thesis presented a parallelized extension of vanilla PageRank algorithm for multi-core architectures. Parallelism of PageRank algorithm is essential for it to be useful in large-network setting. The steps of parallelism as is applied in this thesis can be listed as follows:

1. Data partitioning
2. Updating PageRank vector at each computing unit(a single thread)
3. Updating entire PageRank vector
4. Repeat(2-3) until convergence
5. Persist the resulting PageRank vector into the database

The results in this thesis are presented with the data partitioning step included. Data partitioning step can be excluded from the computation if the data is partitioned before the computation which would normally be the case in a production environment.

The results indicate that, parallelism shows a significant increase in the performance of PageRank algorithm as it is measured in terms of computations time with everything else held constant.

When memory usage of the Parallel PageRank algorithm is detailed, memory usage is 81.66% for data access and data partitioning, 8.12% for PageRank vector and 10.22% for PageRank vector.

When analyzing cost distribution of Parallel PageRank algorithm, data reading and data partitioning form 65.4% of the whole cost. The part of 31.1% forms the calculation and assignment of data to thread isolation suitable for PageRank vector.

**Table 5.1 Sequential And Parallel PageRank Calculation Time Cost**

<b>Approach</b>	<b>Time</b>	<b>Degree Of Parallelism</b>
Sequential	126.151 Seconds	-
Parallel	54.802 Seconds	4

Sequential computation of PageRank algorithm achieved a 126.151 second time saving and parallel computation of algorithm achieved a 54.802 second time saving. Parallelization of algorithm realized an improvement of 43.44% time saving.

The next two extensions of the PageRank algorithms are in two directions: Adapting the algorithm to distributed architectures, and adapting the algorithm to temporal dynamics of the network where connections are created and destroyed in time.

## REFERENCES

### *Books*

Almasi, G.S., & Gottlieb, A., 1989. *Highly parallel computing*. Redwood City, CA: Benjamin-Cummings publishers.

Baeza-Yates R.A., & Boldi P., & Castillo. C. 2008. *Generic damping functions for propagating importance in link-based ranking*. Internet Mathematics, Volume 5 .ISSN: 1542-7951 pp:445–478.

Brin S.,& Page L. 1998.The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems, 33(3):107–117,

El-Ghazawi T., & Carlson W., & Sterling T., & Yelick K., 2005 *UPC: Distributed Shared Memory Programming*, Wiley.

Gropp W.,& Lusk E., & Sterling T., 2003. *Beowulf Cluster Computing with Linux*. The MIT Press, 2<sup>nd</sup> edition,.

Langville, A.N. & Meyer, C.D., 2006. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton, NJ,USA: Princeton University Press.

Marcus, M., & Minc, H. 1988. *Introduction to Linear Algebra*. New York, USA:Dower publications.

Powell, G. 2006. *Beginning Database Design*. IndianaPolis:Wrox.

Trobec R, Vajtersic M and Zinterhof P. , 2009.*Parallel computing: Numerics, Applications, and Trends*. ISBN 978-1-84882-409-9 Springer,

## ***Publications***

- Ahn, H., & Song, C., & , Ahn J.J., & Lee, H.Y., & Kim, T.Y., & Oh, K.J. 2010, *Using Hybrid Data Mining Techniques for Facilitating Cross-selling of a Mobile Telecom Market to develop Customer Classification Model*. Proceedings of the 43rd Hawaii International Conference on System Sciences.
- Asanovic, K. & Bodik, R. & Catanzaro, B.C. & Gebis, J.J.& Husbands, P. & Keutzer, K. & Patterson, D. & Plishker, W.L. & Shalf, J. & Williams, S.W. & Yelick, K.A. 2006. *The Landscape of Parallel Computing Research: A View from Berkeley*. University of California, Berkeley. Technical Report No. UCB/EECS-2006-183
- Boldi, P., & Santini M., & Vigna S. 2005. Pagerank as a function of the damping factor. *In WWW '05: Proceedings of the 14th international conference on World Wide Web.*, New York, NY, USA.
- Cha M., & Mislove A., & Gummadi K. 2009. A measurement-driven analysis of information propagation in the Flickr social network. *In Proc. of the 18th international conference on World Wide Web.*, Madrid, SPAIN.
- Gleich D., & Zhukov L., & Berkhin P., 2004. *A Linear System Approach*. Yahoo! Technical Report <http://www.stanford.edu/~dgleich/publications/dgleich-prlinear.pdf> [April,2011]
- Haveliwala T. 1999. *Efficient computation of pagerank*. Technical Report 1999-31, Stanford InfoLab.
- Haveliwala T., & Kamvar S.,& Klein D., & Manning C.,& Golub C. 2003. *Computing PageRank using Power Extrapolation*. <http://www.stanford.edu/taherh/papers/extrapolationII.pdf> [February 2011]
- Hwu W.,& Keutzer K.,& Mattons T. July/August 2008. *The Concurrency Challenge*. IEEE Design and Test of Computers. pp. 312-320.
- John, W. & Amy, R.G., 2007. *Parallelizing the computation of pagerank*. In Proc. 5th Workshop On Algorithms And Models For The Web-Graph (WAW), pp:202-208.
- Kollias, G. & Gallopoulos, E., 2007. Dagstuhl Seminar Proceedings 07071 Web Information Retrieval and Linear Algebra Algorithms. <http://drops.dagstuhl.de/opus/volltexte/2007/1065/>
- Krapivin, M. & Marchese, M., 2008. DOI 10.1007/978-3-540-89533-6



Langville A.N. & Meyer C.D., 2004. *Deeper Inside PageRank*. <http://mac01-wi428b.math.ncsu.edu/langville/DeeperInsidePR.pdf>

Manaskasemsak, B. & Rungsawang, A. 2005. *An Efficient Partition-Based Parallel PageRank*, (0-7695-2281-5/05 IEEE)

Marcus M., & Minc H. 1988. *Introduction to Linear Algebra*. New York: Dover

Page, L. & Brin, S., & Motwani, R., & Winograd T. 1999. *The pagerank citation ranking: Bringing order to the web*. *Technical Report 1999-66*, Stanford InfoLab, Previous number = SIDL-WP-1999-0120.

Ryoo S., & Rodrigues, C., & Stone, S., & Baghsorkhi, S., & Ueng, S., & Stratton J., & Hwu W., 2008. *Program Optimization Space Pruning for a Multithreaded GPU*. Proceedings of the 2008 International Symposium on Code Generation and Optimization. New York, USA.

Sun, E., & Rosenn I., & Marlow C., & Lento T., 2009. *Gesundheit! modeling contagion through facebook news feed*. In Proc. of International AAAI Conference on Weblogs and Social Media.

Wicks, J.R., & Greenwald, A. 2007 *More efficient parallel computation of pagerank*. in *SIGIR '07*: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. New York, NY, USA:pp:861–862.

West, J., & Althouse, B., & Bergstrom, C., & Rosvall, M., & Bergstrom, T. 2007. *Ranking and mapping scientific knowledge*. <http://www.eigenfactor.org> [June 2010]

## **CURRICULUM VITEA**

**Name/Surname** : Murat HAKSAL

**Address** : Cevizlik Mah. Kartopu sok. Kuşluk Apt. No:2 Bakırkoy/Istanbul

**Place & Birthday**: Hamburg, 1981

**Primary School** : Gaziantep Gazi İlk Öğretim Okulu, 1996

**High Schoole** : Gaziantep Lisesi, 2000

**B.S** : Çağ Üniversitesi- İşletme, 2005

**M.S.** : Bahçeşehir Üniversitesi Bilgisayar Mühendisliği, 2011

**Name of Schoole** : Fen Bilimleri Enstitüsü

**Name** : Bilgisayar Mühendisliği Yüksek Lisans Programı

**Industrial Experinces** : Eczacıbasi-Intema 2003-2005, BilgeAdam 2005-...