TIGHT BOUNDS ON ONE-PASS MAP-REDUCE ALGORITHMS FOR MATRIX MULTIPLICATION

A Thesis by

Ashita Nagar

Bachelor of Engineering, Rajiv Gandhi Technical University, 2008

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

December 2015

.

@ Copyright 2015 by Ashita Nagar

All Rights Reserved

LOWER BOUNDS ON THE COST OF MAP-REDUCE FOR MATRIX MULTIPLICATION

The following faculty members have examined the fi copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science, with a major in Computer Science.

Prakash Ramanan, Committee Chair

Vinod Namboodiri, Committee Member

Suresh Keshavanarayana, Committee Member

DEDICATION

To my parents, my husband, my sister,
and my dear friends

ACKNOWLEDGEMENTS

Foremost, I would like to express my deep and sincere gratitude to my supervisor, Dr. Prakash Ramanan of Electrical Engineering and Computer Science Department, Wichita State University. His understanding, wide knowledge, logical way of thinking, insightfulness and personal guidance have provided a good basis for the thesis. His personal guidance, kind support and encouraging attitude helped constantly while working on this research and writing of my thesis. I would also like to extend my gratitude to members of my committee, Vinod Namboodiri, and Suresh Keshavanarayana for all the support and help.

I gratefully acknowledge my mother, Anita Nagar, father, Devendra Nagar, and husband, Swapnil Nagar for their endless love, affection and concern. At the end, I also owe loving thank to my sister, and all my friends who trusted me and made this thesis possible and an unforgettable experience for me.

ABSTRACT

MapReduce is an effi t parallel computation model introduced by Google, for performing many large-scale computations, including matrix multiplication. Matrix multiplication can be done using either an one-pass or a two-pass MapReduce algorithm; these algorithms have been extensively studied. In this thesis, we studied the tradeoff between communication cost and parallelism, for one-pass algorithms for matrix multiplication. We measured communication cost using the replication rate r, as in the literature. We measured parallelism either by reducer size q as in the literature, or by a new parameter, namely, reducer workload w. First, we provided matching upper and lower bounds on qr, for the multiplication of sparse rectangular matrices; this extends a previously-known result for dense square matrices. Then, we provided matching upper and lower bounds on wr^2, for the multiplication of sparse rectangular matrices.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In today's world, the storage and processing of massive datasets in a cost-efficient manner is one of the major concerns in many large-scale scientific problems. Numerous high performance computing applications require collection and processing of ultra-large datasets of the size of terabytes or petabytes. This includes applications such as digital image processing, web search engines, real-time computing, recommendation engines, and so on. Most of these scientific or engineering applications use matrices to store and manage the data, and matrix multiplication plays an important role in operating on that data.

Matrix multiplication is often used as a building block for solving several data-intensive problems including maximum matching [1], fi connected components [2], large-scale numerical analysis, colored intersection searching [3], and many other graph problems. Multiplication of large matrices is a time-consuming task that can affect the performance of the overall algorithm. Many popular Information Technology companies have come up with several innovative solutions and products to speedup matrix multiplication.

One way to speedup these algorithms is to execute their operations in parallel. In parallel processing, tasks are divided into sub-tasks that are distributed among several nodes to be processed together. Many diff t models of parallel computation have been studied in the literature. Traditional models include the distributed memory model, and the shared-memory model. [4] presented communication lower bounds for matrix multiplication in the distributed memory model. They showed a tradeoff between communication among the processors, and memory size of the individual processors. [5] presented a parallel algorithm that uses shared memory and remote memory access communication.

The shared-memory and distributed memory models mentioned above involve special-purpose architectures. Several companies have developed more economical distributed architectures using commodity hardware. These include Microsoft's Dryad [6], Hadoop [7], Bigtable[8], and Google's File System [9].

Google [9,10] introduced a distributed framework called MapReduce. Apache's Hadoop [7] is one successful open source implementation of the MapReduce framework. MapReduce is an efficient model for processing large datasets in parallel, on huge clusters of commodity compute nodes that are connected together by ethernet or switches. The MapReduce framework provides high fl y and well-organized resource utilization, to provide solutions for large-sized problems in reasonable time. The framework also coordinates the parallel processes in a specific way, and excels in handling complicated tasks like load balancing, data distribution, and fault tolerance. [10] show why MapReduce is a better framework compared to other parallel models for management of large volume of data.

Because of the rapid growth of social networks, mobile networks, and search engines, cloud computing using MapReduce has become popular. MapReduce has been widely adopted in several data-intensive applications, including GPU and multicore processing [11], document clustering [12], and graph analysis [13]. Massive graphs are often used to capture big data, and MapReduce is used to solve many such graph based problems. [14] provides algorithms for the problems of minimum spanning tree, maximal matching, and approximate weighted matchings. More than 10,000 programs for large-scale fault-tolerant data analysis have been implemented using MapReduce.

Ullman [15] shows that for the efficient implementation of MapReduce, it is important to understand the various complexity factors affecting the performance. MapReduce algorithms consist of two phases: The map phase and the reduce phase. The *map phase* consists of several *mappers*, and the *reduce phase* consists of several *reducers*. *Communication cost* is one of the most dominating costs of the MapReduce framework; it measures the amount of communication from the map phase to the reduce phase. *Parallelism* is essentially the number of reducers; it is inversely related to the size of each reducer (processor memory), or the amount of work done (workload) by each reducer (see below). There is a tradeoff between communication cost and parallelism; smaller the reducers, higher is the number of reducers (and parallelism), and higher is the communication cost. It is important to have a balance between these two costs. [16] gives such tradeoff for the problems of fi triangles in large graphs, and fi pairs of strings with a particular hamming distance.

MapReduce is an efficient model for performing matrix multiplication. Matrix multiplication can be done using either an one-pass or a two-pass MapReduce algorithm; these algorithms have been extensively studied. Chapter 2 of [17] gives details of matrix multiplication using MapReduce. First, it explains the large-scale distributed fi system, and then provides the implementation details and architecture of MapReduce. The details of the map phase and reduce phase for matrix multiplication are well explained in the book. It also describes the complexity theory of MapReduce. [16] (also see [17]) presents methods for fi upper and lower bounds for the tradeoff between communication cost and parallelism. They studied the multiplication of dense square matrices. They showed that, for a given reducer size, the two-pass algorithm has less communication than the one-pass algorithm.

[18] also studied multiplication of dense and sparse matrices using MapReduce. They presented upper and lower bounds on the number of MapReduce passes required as a function of reducer sizes and the total memory in the computing cluster.

In this thesis, we study the tradeoff between communication cost and parallelism, for one-pass MapReduce algorithms for matrix multiplication. We measure parallelism either by *reducer size* or *reducer workload* (these terms are defi in Chapter 2). We provide the following:

1. *Tradeoff between communication cost and reducer size*: [16] provided tight upper and lower bounds for multiplication of dense square matrices. But in many real-world applications, the matrices to be multiplied are sparse; i.e., most of the elements in the matrices are zero. We extend the results of [16] to sparse rectangular matrices.

2. *Tradeoff between communication cost and reducer workload*: We provide tight upper

and lower bounds for multiplication of sparse rectangular matrices.

The outline of this thesis is as follows. In Chapter 2, we introduce the MapReduce framework. In Chapter 3, we present simple one-pass and two-pass algorithms for matrix multiplication from [17]. In Chapters 4 and 5, we present tight bounds for tradeoff of communication cost vs. reducer size and reducer workload, respectively. In Chapter 6, we present our conclusions.

CHAPTER 2

THE MAP-REDUCE FRAMEWORK

In this chapter, we fi present an overview of the MapReduce framework [17], and then introduce its complexity factors.

2.1Overview of MapReduce

Google's MapReduce is a programming model used for parallel computing in many data-mining or big-data analysis applications. In the MapReduce architecture, large number of economical compute nodes are connected by Ethernet cables or switches. Compute nodes operate on fi of enormous size, using a distributed fi system [9]. Files are divided into chunks and these chunks are processed in parallel.

A MapReduce algorithm consists of two functions: *map function* and *reduce function*. Along with these two functions, an important role is played by the Master Controller in the middle. An overview of the MapReduce framework is as follows:

1. **Map Function:** This is the function executed by each map task. Each map task gets as input a chunk of the input fi For each input element, the map task outputs zero or more (key, value) pairs. The keys of the generated (key, value) pairs need not be unique. The code for the map function is written by the user, tailored to the specific problem to be solved.

2. **Master Controller:** The output of all the map tasks is collected by the master controller. The controller groups the (key, value) pairs and, for each distinct key, generates a pair consisting of the key and a list of all the values associated with that key.

3. **Reduce Function:** This is the function executed by each reducer. Each reducer

receives a (key, list of values) pair from the master controller. The function combines all the values associated with that key in some specifi way. The code for combining the values is written by the user, tailored to the specific problem to be solved. The output of all the reducers is the output (i.e., result) for the entire MapReduce algorithm.

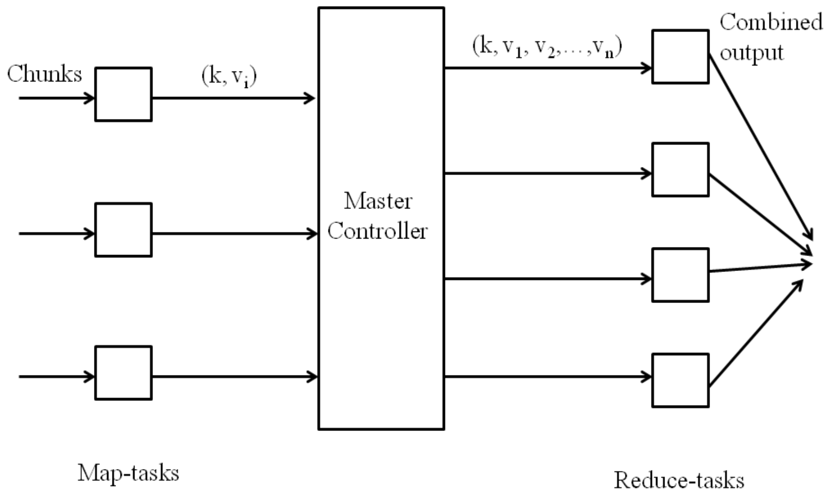

Figure 2.1: *Schematic of MapReduce Architecture*

2.2 MapReduce Complexity Factors

Communication cost and the degree of parallelism are the two most important factors for a MapReduce algorithm. We defi the following three measures. The fi measures communication cost, and the latter two measure the degree of parallelism.

1. **Replication Rate** r**:** This measures communication cost, that is the cost of sending data from the map phase to the reduce phase. Replication rate r [16] is the average communication from the map phase to the reduce phase, per each item input to the map phase. It is the average number of (key, value) pairs generated by the map tasks for each input item. The replication rate r is computed as follows.

$$r = \frac{\sum_{M\text{-}tasks} \textit{Output size of the map task}}{\sum_{M\text{-}tasks} \textit{Input size of the map task}}$$

2. **Reducer Size** *q***:** This is a measure of the degree of parallelism. Reducer size [16] is the size of each reducer: The size of the list of values sent to each reducer. It measures the amount of memory each reducer needs. Smaller the reducer size, higher is the degree of parallelism.

3. **Reducer Workload** *w***:** This is also a measure of the degree of parallelism. It is the amount of work done by each reducer. Smaller the reducer workload, higher is the degree of parallelism.

Having more reducers increases the replication rate (i.e., communication cost), because the same input item has to be sent to more reducers. But having more reducers decreases reducer size and workload. If reducer size is small, then the reduce function can be executed in the main memory. So, having a smaller reducer size or workload lowers the wall-clock time for each reducer; this increases parallelism. So, we study the tradeoff between replication rate r and each of reducer size q (Chapter 4) and workload w (Chapter 5).

CHAPTER 3

MATRIX MULTIPLICATION

In this chapter, we describe simple one-pass and two-pass algorithms for matrix multiplication, and determine their replication rate, and reducer size and workload. For the sake of simplicity, we consider square matrices. These algorithms are from [17].

Consider two $n \times n$ matrices M and N. Their product $P = MN$ is an $n \times n$ matrix. Let m_{ij} denote the i^{th} row, j^{th} column element of M. Similarly, let n_{jk} denote the j^{th} row, k^{th} column element of N. $p_{ik} = \sum_j m_{ij}n_{jk}$ is the i^{th} row, k^{th} column element of P.

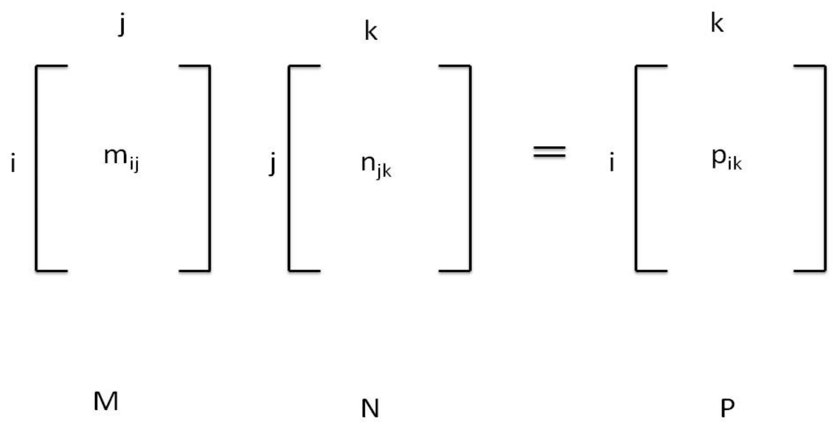

Figure 3.1: *Matrix Multiplication*

3.1One-Pass Algorithm

Recall, from Chapter 2, that a MapReduce algorithm consists of a map function and reduce function. For one-pass matrix multiplication, these functions are described as follow.

Map Function:

Input: Each map task gets a chunk of M or a chunk of N.

Output: For each element m_{ij} of M, generate the n (key, value) pairs:

$$m_{ij} \in M \rightarrow ((i,k),(M,j,m_{ij})), \forall k, 1 \leq k \leq$$

n. For each element n_{jk} of N, generate the n (key, value) pairs:

$$n_{jk} \in N \rightarrow ((i,k),(N,j,n_{jk})), \forall i, 1 \leq i \leq n.$$

Reduce Function:

Input: Input is a (key, list of values) pair, where key (i, k) has associated list of values (M, j, m_{ij}) and (N, j, n_{jk}), for $1 \leq j \leq n$. So, the list of values consists of the entire i^{th} row of M, and the k^{th} column of N.

Output: Reduce function computes p_{ik} as the product of the i^{th} row of M and the j^{th} column of N.

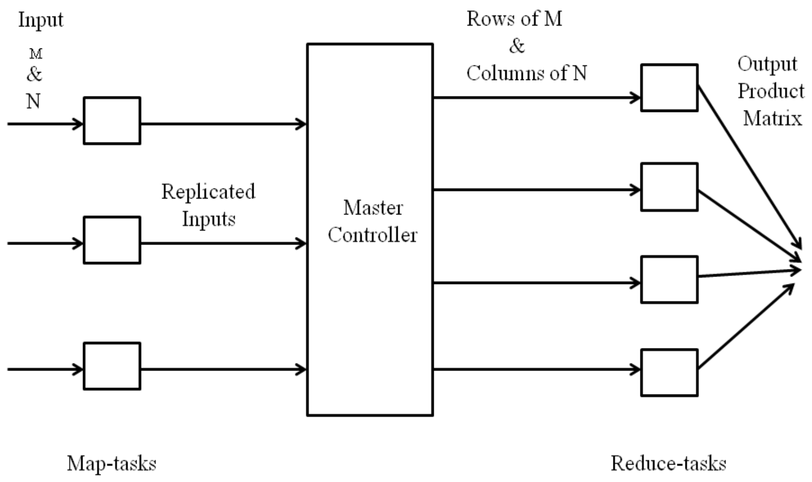

Figure 3.2: *One-Pass Algorithm*

Now, let us determine the replication rate r, reducer size q and workload w, for this algorithm. For each element $m_{ij} \in M$ and $n_{jk} \in N$, the map function produces n

(key,

value) pairs. So, the replication rate is $r = n$; communication cost is $2n^2r = 2n^3$. Each reducer receives a row of M and a column of N; so $q = 2n$. The reducer computes the product of a row of M and a column of N; so $w = n$. The algorithm uses n^2 reducers.

3.2 Two-Pass Algorithm

This algorithm uses a cascade of two MapReduce passes.

Pass I

Map Function:

Input: Each map task gets a chunk of M or a chunk of N.

Output: For each element m_{ij} of M, generate the (key, value) pair:

$$m_{ij} \in M \rightarrow (j, (M, i, m_{ij}))$$

For each element n_{jk} of N, generate the (key, value) pair:

$$n_{jk} \in N \rightarrow (j, (N, k, n_{jk}))$$

Reduce Function:

Input: Input is a (key, list of values) pair, where key j has associated list of values (M, i, m_{ij}) and (N, k, n_{jk}), for $1 \leq i, k \leq n$. So, the list of values consists of the entire j^{th} column of M, and the j^{th} row of N.

Output: For each j, the reduce function generates the n^2 (key, value) pairs $((i, k), m_{ij}n_{jk})$, $1 \leq i, k \leq n$; the key is (i, k), and the value is $m_{ij}n_{jk}$.

Pass II

Map Function: The map function is identity:

$$((i, k), m_{ij}n_{jk}) \rightarrow ((i, k), m_{ij}n_{jk})$$

Reduce Function:

Input: Input is a (key, list of values) pair, where each key (i, k) has associated list of values

$$[m_{i1}n_{1k}, m_{i2}n_{2k}, \ldots, m_{in}n_{nk}]$$

Output: For each key (i, k), generate p_{ik} as the sum of values in the list: $p_{ik} = \sum_j m_{ij}n_{jk}$.

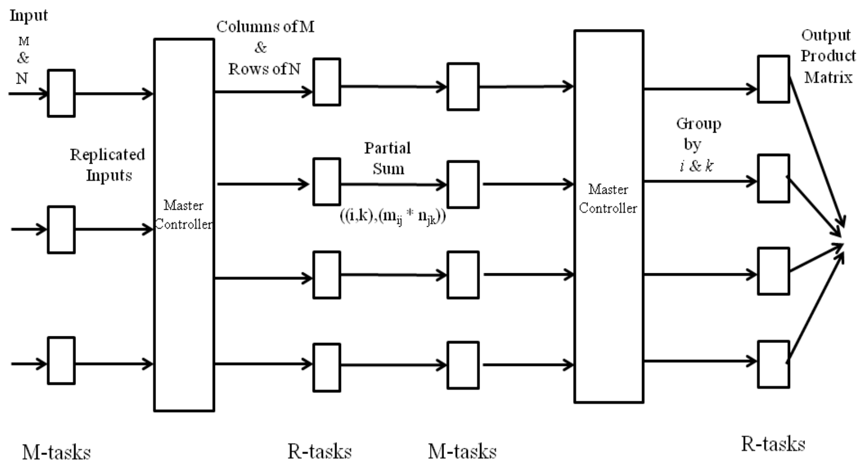

Figure 3.3: *Two-Pass Algorithm*

Now, let us determine the replication rate r, reducer size q and workload w, for this algorithm. In the fi pass, for each element $m_{ij} \in M$ and $n_{jk} \in N$, the map function produces only one (key, value) pair. So, the replication rate is $r_1 = 1$; communication cost is $2n^2r_1 = 2n^2$. Each reducer receives a column of M and a row of N; so $q_1 = 2n$. The reducer computes the product of a column of M and a row of N; so $w_1 = n^2$. The fi pass uses n reducers.

Now, consider the second pass. For each input $((i, k), m_{ij}n_{jk})$, the map function produces only one (key, value) pair. So, the replication rate is $r_2 = 1$; communication cost is $n^3r_2 = n^3$. The reducer for key (i, k) receives a list of n values $m_{ij}n_{jk}$, $1 \leq j \leq n$; so $q_2 = n$. The reducer computes the sum of these n values; so $w_2 = n$. The second pass uses n^2 reducers. The overall communication cost is $2n^2 + n^3 \approx n^3$, and replication rate is $r = n^3/(2n^2) = n/2$. Further $q = \max(q_1, q_2) = 2n$ and $w = \max(w_1, w_2) \approx n^2$.

CHAPTER 4

TIGHT BOUND ON *qr*

In this chapter we provide a tight bound on qr for one-pass matrix multiplication; recall that r is the replication rate, and q is the reducer size. [16] presented tight bounds on qr for multiplying dense square matrices. Using their technique, we extend their result to sparse rectangular matrices. We provide matching upper and lower bounds; so our algorithm has optimal tradeoff between r and q.

We have two rectangular matrices M and N. M is of size $m \times n$, and N of size $n \times p$. We want to compute the product matrix $P = MN$ of size $m \times p$. The matrices M and N could be sparse; i.e., most of their elements could be zero. Let M have a non-zero entries per row, and N have b non-zero entries per column. So, there are ma non-zero elements in M and pb non-zero elements in N.

4.1The Algorithm and the Upper Bound

As in [16], we group the rows of M and the columns of N into bands. Each reducer computes a rectangular block of P, using a band of rows from M and a band of columns from N. To get the best possible tradeoff between r and q, we need to fi the best division of M and N into bands.

Group the m rows of M into g_1 bands, where each band has m/g_1 rows. Similarly, group the p columns of N into g_2 bands, where each band has p/g_2 columns. The map and reduce functions are as follow.

Map Function

Input: Each map task gets a chunk of M or a chunk of N.
Output: For a non-zero element $m_{ij} \in M$, let row i of M be in band x_i. Output g_2 (key,

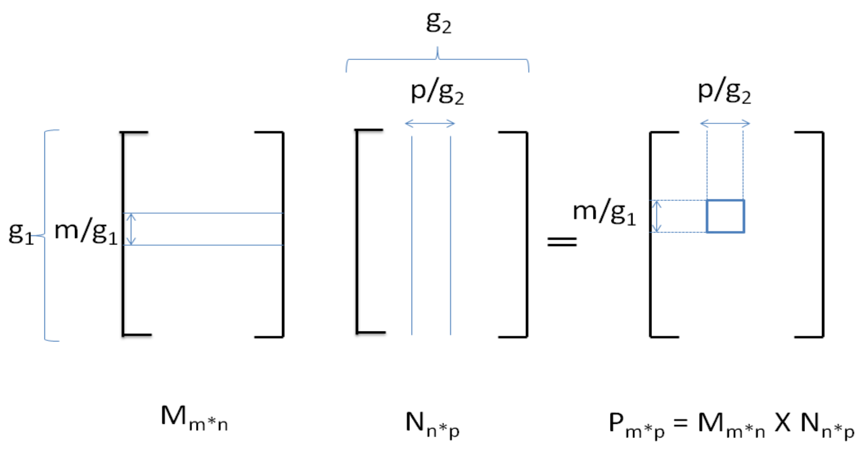

Figure 4.1: *One-Pass MapReduce: Division into Bands*

value) pairs:

$$m_{ij} \rightarrow ((x_i, y), (M, i, j, m_{ij})), \text{ for all } y, 1 \leq y \leq g_2.$$

The key is (x_i, y), and the value is (M, i, j, m_{ij}).

Similarly, for a non-zero element $n_{jk} \in N$, let column k of N be in band y_k. Output g_1 (key, value) pairs:

$$n_{jk} \rightarrow ((x, y_k), (N, j, k, n_{jk})), \text{ for all } x, 1 \leq x \leq g_1.$$

The key is (x, y_k), and the value is (N, j, k, n_{jk}).

Reduce Function

Input: Each reducer gets a band of rows from M, and a band of columns from N. The reducer with key (x, y) has all the non-zero elements of the x^{th} band of M, and the y^{th} band of N.

Output: The reducer has all the elements of M and N needed to compute a block of P. It computes a sub-matrix of P of size $m/g_1 \times p/g_2$.

Finding the optimal subdivision of *M* and *N* into bands

Number of reducers required to compute P is $k = g_1 g_2$.

Replication rate for M is g_2, but only for the ma non-zero entries.

Replication rate for N is g_1, but only for the pb non-zero entries.

Overall Replication Rate is

$$r = \frac{\sum_{M-tasks} \textit{Output size of the Map task}}{\sum_{M-tasks} \textit{Input size of the Map task}} = \frac{mag_2 + pbg_1}{ma + pb} \tag{4.1}$$

Each reducer gets all the non-zero entries of a band of M and a band of N. So, the reducer size is

$$q = \frac{ma}{g_1} + \frac{pb}{g_2} = \frac{mag_2 + pbg_1}{g_1 g_2} \tag{4.2}$$

$$qr = \frac{1}{(ma + pb)}(mag_2 + pbg_1)^2 (g_1 g_2)^{-1} \tag{4.3}$$

Note that m, n, p, a, b are constants outside the control of our algorithm. We want to minimize qr by choosing optimal values for g_1 and g_2. For this, we compute the partial derivatives of qr with respect to g_1 and g_2, and set them to 0.

$$\frac{\partial(qr)}{\partial g_1} = \frac{1}{ma + pb}[2(mag_2 + pbg_1)(pb)(g_1 g_2)^{-1} - (mag_2 + pbg_1)^2 g_2^{-1} g_1^{-2}]$$

Setting this to 0 gives $mag_2 = pbg_1$, or

$$\frac{ma}{g_1} = \frac{pb}{g_2} \tag{4.4}$$

Setting $\partial(qr)/\partial g_2 = 0$ gives the same result. So, to minimize qr, the number of non-zero entries in a band of M should equal the number of non-zero entries in a band of N.

From Eqns. 4.1–4.3, the resulting values of q, r and qr are as follow:

$q = 2ma/g_1 = 2pb/g_2$

$r = 2mag_2/(ma + pb) = 2pbg_1/(ma + pb)$

$$qr = \frac{4mapb}{(ma + pb)} \tag{4.5}$$

So, the algorithm we described at the beginning of this section has a tradeoff between the values of q and r specified above, such that qr has the value in Eqn. 4.5.

4.2 The Lower Bound

Our lower bound proof is based on the technique in [16]. We fi fi an upper bound on the number of entries of P that can be computed by a reducer of a given size q (Subsection 4.2.1). Then, using this, we derive a lower bound on qr, for any one-pass algorithm (Subsection 4.2.2).

4.2.1Upper Bound on Entries Computed by a Reducer

As pointed out in [17], if a reducer does not get all the entries in a row i of M, that reducer cannot compute any entry in row i of P. So, we assume that each reducer gets a certain number r_1 of full rows of M. For a similar reason, we assume that each reducer gets a certain number r_2 of full columns of N. Since the reducer needs only the nonzero entries in these rows/columns, the reducer size is:

$$q = r_1a + r_2b \tag{4.6}$$

We will be using the following fact in this subsection and in Chapter 5.

Fact 4.1. Consider positive real values x, y such that $x + y = c$, where c is a constant. The function $f(x, y) = xy$ has a unique maximum at $x = y = c/2$, and the maximum value of f is $c^2/4$.

Each reducer R computes a $r_1 \times r_2$ submatrix of P. So, the number of entries of P computed by R is r_1r_2. By Fact 4.1, the maximum value of $(r_1a)(r_2b)$, subject to Eqn. 4.6, is $q^2/4$. So, the maximum number r_1r_2 of entries of P that can be computed by a reducer of size q is

$$h(q) = \frac{q^2}{4ab} \tag{4.7}$$

4.2.2 Lower Bound on qr

Consider any one-pass algorithm; suppose that it has k reducers. The i^{th} reducer R_i has size q_i. By Eqn. 4.7, the maximum number of entries of P that R_i can compute is $h(q_i) = q_i^2/4ab$. Since the k reducers together must compute all the mp entries of P, we have $\sum_{i=1}^{k} h(q_i) \geq mp$. So, $\sum_{i=1}^{k} q_i^2/4ab \geq mp$, or

$$\sum_{i=1}^{k} q_i^2 \geq 4mpab$$

Since q is the largest size of any reducer, $q_i \leq q$ for all i. So,

$$q \sum_{i=1}^{k} q_i \geq 4mpab$$

The communication cost is

$$\sum_{i=1}^{k} q_i = r(ma + pb)$$

Substituting this in the previous inequality gives $qr(ma + pb) \geq 4mpab$, or

$$qr \geq \frac{4mpab}{(ma + pb)}$$

This lower bound matches the value of qr for our algorithm given in Eqn. 4.5. So, our algorithm in Section 4.1 has optimal tradeoff.

Theorem 4.1. For any one-pass MapReduce algorithm for matrix multiplication, $qr \geq 4mpab/(ma + pb)$. Also, there exists a family of algorithms that have a tradeoff between q and r such that $qr = 4mpab/(ma + pb)$.

CHAPTER 5

TIGHT BOUND ON wr^2

In this chapter, we provide a tight bound on wr^2 for one-pass matrix multiplication; recall that r is the replication rate, and w is the reducer workload. First, we prove the bound for dense square matrices; in addition to illustrating our method in a simple setting, this shows why we chose the measure wr^2. Then, we extend it to sparse rectangular matrices. We provide matching upper and lower bounds; so, our algorithm has optimal tradeoff.

5.1Tight Bound for Dense Square Matrices

In this section, we prove a tight bound on wr^2 of n^3, for multiplying dense square matrices. We have two square matrices M and N; each is of size $n \times n$. We want to compute the product matrix $P = MN$ of size $n \times n$.

To prove the upper bound on wr^2, we consider the algorithm from Chapter 4, specialized to $m = p = n$ and $g_1 = g_2 = g$. M is divided into g bands; each band has n/g rows. Similarly, N is divided into g bands; each band has n/g columns.

Each reducer computes one $(n/g) \times (n/g)$ block of P.

The number of reducers used by the algorithm is $k = g^2$.

Replication rate is $r = g$.

Work done by each reducer is $w = (n/g)^2 n = n^3/g^2 = n^3/r^2$.

So, $wr^2 = n^3$.

For the lower bound, consider any one-pass algorithm with k reducers; let w_i be the work done by the i^{th} reducer. The total work needed to compute P is $W = n^3$. So, the total work done by all the reducers is $\sum_i w_i \geq W = n^3$. Letting $w = max_i\, w_i$, we have

$$wk \geq n^3 \tag{5.1}$$

As explained in Chapter 4, we can assume that each reducer gets a band of adjacent rows of M, and a band of adjacent columns of N. Let M be divided into g_1 bands of rows, and N be divided into g_2 bands of columns. Then $k = g_1g_2$.

Average replication rate is $r = (g_1 + g_2)/2$. So, by Fact 4.1, the maximum value of $k = g_1g_2$ is r^2. Substituting this in Eqn. 5.1, we have $wr^2 \geq n^3$. This lower bound matches our upper bound above.

5.2 Tight Bound for Sparse Rectangular Matrices

In this section, we prove a tight bound on wr^2 for multiplying sparse rectangular matrices. The setting is as in Chapter 4. We have two rectangular matrices M and N. M is of size $m \times n$, and N of size $n \times p$. We want to compute the product matrix $P = MN$ of size $m \times p$. Let M have a non-zero entries per row, and N have b non-zero entries per column. So, there are ma non-zero elements in M and pb non-zero elements in N.

To prove the upper bound on wr^2, we consider the algorithm from Chapter 4. M is divided into g_1 bands; each band has m/g_1 rows. Similarly, N is divided into g_2 bands; each band has p/g_2 columns. Each reducer computes one $(m/g_1) \times (p/g_2)$ block of P. The number of reducers used by the algorithm is $k = g_1g_2$. Replication rate is

$$r = \frac{mag_2 + pbg_1}{ma + pb} \quad (5.2)$$

Computing each entry of P involves multiplying a row of M (with a nonzero entries) with a column of N (with b nonzero entries); we assume that this takes work $(a + b)$. So, work done by each reducer is

$$w = \frac{m\,p}{g_1g_2}(a+b)(5.3)$$

So,

$$wr^2 = \frac{mp(a+b)}{(ma+pb)^2}(mag_2 + pbg_1)^2 (g_1g_2)^{-1} \quad (5.4)$$

We want to minimize wr^2 by choosing optimal values for g_1 and g_2. For this, we compute the partial derivatives of wr^2 with respect to g_1 and g_2, and set them to 0. As in Chapter 4, this gives $ma/g_1 = pb/g_2$. Substituting this in Eqns. 5.2–5.4 gives:

$w = m^2a(a + b)/(bg_1^2) = p^2b(a + b)/(ag_2^2)$

$r = 2mag_2/(ma + pb) = 2pbg_1/(ma + pb)$

$$wr^2 = \frac{4m^2p^2ab(a+b)}{(ma+pb)^2} \tag{5.5}$$

For the lower bound, consider any one-pass algorithm with k reducers; let w_i be the work done by the i^{th} reducer. The total work needed to compute P is $W = mp(a + b)$. So, the total work done by all the reducers is $\sum_i w_i \geq W = mp(a + b)$. Letting $w = max_i w_i$, we have

$$wk \geq mp(a+b) \tag{5.6}$$

As explained in Chapter 4, we can assume that each reducer gets a band of adjacent rows of M, and a band of adjacent columns of N. Let M be divided into g_1 bands of rows, and N be divided into g_2 bands of columns. Then $k = g_1g_2$.

Average replication rate is $r = (mag_2 + pbg_1)/(ma + pb)$. So, by Fact 4.1, the maximum value of $(mag_2)(pbg_1)$ is $(ma + pb)^2r^2/4$. So, maximum value of $k = g_1g_2$ is $(ma + pb)^2r^2/(4mapb)$. Substituting this in Eqn. 5.6, we have

$$wr^2 \geq \frac{4m^2p^2ab(a+b)}{(ma+pb)^2}$$

This lower bound matches our upper bound in Eqn. 5.5.

Theorem 5.1. For any one-pass MapReduce algorithm for matrix multiplication, $wr^2 \geq 4m^2p^2ab(a + b)/(ma + pb)^2$. Also, there exists a family of algorithms that have a tradeoff between w and r such that $wr^2 = 4m^2p^2ab(a + b)/(ma + pb)^2$.

CHAPTER 6

CONCLUSION

MapReduce is an efficient parallel computation model introduced by Google for performing many large-scale computations, including matrix multiplication. Matrix multiplication can be done using either an one-pass or a two-pass MapReduce algorithm; these algorithms have been extensively studied. [16] (also see [17]) studied the multiplication of dense square matrices. For one-pass algorithms, they presented a method for fi upper and lower bounds for the tradeoff between communication cost and parallelism. Then, they showed that, for a given reducer size, the two-pass algorithm has less communication than the one-pass algorithm.

In this thesis, we studied the tradeoff between communication cost and parallelism, for one-pass MapReduce algorithms for matrix multiplication. We measured communication cost using the replication rate r, as in [16]. We measured parallelism either by reducer size q as in [16], or by a new parameter, namely, reducer workload w.

In Chapter 4, we provided matching upper and lower bounds on qr, for the multiplication of sparse rectangular matrices; this extends the result of [16] for dense square matrices. In Chapter 5, we provided matching upper and lower bounds on wr^2, for the multiplication of sparse rectangular matrices.

Extending our results to two-pass algorithms remains an interesting open problem.

.

REFERENCES

REFERENCES

[1]Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1987.

[2]Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 254–260. Society for Industrial and Applied Mathematics, 2004.

[3]Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 52–60. ACM, 2006.

[4]Dror Irony, Sivan Toledo, and Alexander Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

[5]Manojkumar Krishnan and Jarek Nieplocha. Srumma: a matrix multiplication algorithm suitable for clusters and scalable shared memory systems. In *Proceedings of the Parallel and Distributed Processing Symposium,* pages 70–71. IEEE, 2004.

[6]Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[7]Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed fi system. In *26th Symposium on Mass Storage Systems and Technologies (MSST), 2010 IEEE*, pages 1–10. IEEE, 2010.

[8]Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):205-218, 2008.

[9]Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google fi system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[10]Ralf Lämmel. Googles mapreduce programming model–revisited. *Science of computer programming*, 70(1):1–30, 2008.

[11] Jimmy Lin and Chris Dyer. Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177, 2010.

[12] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997.

[13] Foto N Afrati, Dimitris Fotakis, and Jeffrey D Ullman. Enumerating subgraph instances using map-reduce. In *29th International Conference on Data Engineering (ICDE), 2013 IEEE*, pages 62–73. IEEE, 2013.

[14] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 85–94. ACM, 2011.

[15] Jeffrey D Ullman. Designing good mapreduce algorithms. *XRDS: Crossroads, The ACM Magazine for Students*, 19(1):30–34, 2012.

[16] Anish Das Sarma, Foto N Afrati, Semih Salihoglu, and Jeffrey D Ullman. Upper and lower bounds on the cost of a map-reduce computation. In *Proceedings of the VLDB Endowment*, volume 6, pages 277–288. VLDB Endowment, 2013.

[17] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[18] Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. Space-round tradeoffs for mapreduce computations. In *Proceedings of the 26th ACM international conference on Supercomputing*, pages 235–244. ACM, 2012.