Parallel PageRank Computation
CS240B Project Report

Han Chen, Kirsten Meeker and Zheming Zheng
{hanchen,kmeeker,zhengzhm}@cs.ucsb.edu

March 18, 2003

1 Introduction

The World Wide Web creates many new challenges for information retrieval. It is very large, heterogeneous, and growing rapidly. However, the World Wide Web is hypertext and provides considerable auxiliary information on top of the text of the web pages, such as link structure. PageRank[1] helps search engines and users quickly make sense of the vast heterogeneity of the World Wide Web, and is used in search engines like Google to determine the "importance' of a web page based on the 'importance' of its parent pages[2]. There have been some studies investigating computing PageRanks for specific topics, and some speculation on customized PageRanks for individual users. Toward this end there have been investigations into ways to optimize the algorithm for both memory storage requirement and computation time on a single processor. This project is to develop a program to compute PageRank on a parallel cluster. If fast enough, customized PageRanks could be computed in the time a web search service takes to reply to a query.

2 Literature Survey

2.1 The PageRank Citation Ranking: Bringing Order to the Web

Page, Brin, Motwani and Winorgrad [1] describe a method for ranking web pages based on the link structure of the web. Links from a page are referred to as forward links, and links to a page as backlinks. Many search engines have used a simple backlink count, but this method does not give any weight to the importance of the page that the backlink is from. An example given is a link from Yahoo being given the same weight as a link from an obscure page. PageRank improves on simple backlink counting by using the sum of the ranks of the backlinked pages instead. So a few high-ranked pages are equivalent to many low-ranked pages.

2.1.1 Definition of PageRank

Let u be a web page, B_u the set of pages that point to u (backlinks), and F_u the set of pages u point to (forward links). Let $N_u = |F_u|$ be the number of links from u, and c be a normalization factor used to keep the total rank of all web pages constant. Then a simplified version of a page's rank, R, can be defined as

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} \tag{1}$$

This divides the rank of a page among its forward links evenly to contribute to the ranks of the pages they point to. It is a recursive equation, and can be expressed in another way a matrix equation

$$R = cAR \tag{2}$$

Where A is a square matrix with rows and columns representing web pages, and entries $A_{u,v} = 1/N_u$ if there is a link from u to v, and $A_{u,v} = 0$ if not. R is an eigenvector of A with eigenvalue c. The PageRank vector is the dominant eigenvector of A, and is found by recursively applying the equation starting from any nondegenerate vector.

Another way to look at PageRank is to think of it as the probability distribution of a random walk of the web. A corresponds to the stochastic transition matrix, and PageRank can be viewed as the stationary probability distribution over pages induced by a random walk on the web.

2.1.2 Rank Sinks

If two web pages are interconnected, but isolated from any other pages, they will form a trap called a "rank

sink" which accumulates rank but never distributes it. The way to avoid this is to add a small probability of a random jump from one web page to another (irrespective of linkage). This is modeled by adding a vector $E(u)$, which is analogous to a source of rank. Equation 1 then becomes

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} + E(u) \quad (3)$$

Or

$$R = c(AR + E) \quad (4)$$

Since $||R||_1 = 1$ this can be rewritten as

$$R = c(A + E \times \mathbf{1})R \quad (5)$$

Where $\mathbf{1}$ is the row vector consisting of all ones, and R is an eigenvector of $(A + E \times \mathbf{1})$.

2.1.3 Convergence Properties

The Convergence of PageRank is guaranteed only if A is irreducible (i.e., the directed graph of the web is strongly connected) and aperiodic [3]. The latter is true in practice for the web, while the former is guaranteed by adding the dampening factor E.

2.1.4 Implementation

The implementation of PageRank at Stanford starts with a web crawler that converts each URL into a unique integer and stores them in a link database. The database is sorted by Parent ID (making access linear), and dangling links (links to pages without any outgoing links) are removed. Memory is allocated for the current PageRank vector, (a 4 byte single precision floating point for each web page), and an initial guess for it's value are made. The PageRank from the previous iteration is stored on disk. For a database of 75 million URLs, 300 MB of memory was used and each iteration (reading A and the previous PageRank vector from disk) took 6 minutes. After the vector converged, the dangling links were added back in, and the vector was recomputed. This takes as many iterations as it took to remove the dangling links. The whole process took about 52 iterations in five hours to reach a reasonable tolerance. Convergence on half the data took about 45 iterations, so the scaling factor is approximately linear in $\log n$. This means it should scale well for larger n.

2.2 Efficient Computation of PageRank

Haveliwala [4] discusses several aspects of computing PageRank efficiently. He argues that efficiency can achieved in the following ways:

1. Use the principle underlying the block nested-loops-join strategy, that efficiently controls the cost per iteration even in low memory environments (Section 3)
2. Single-precision rank values are sufficient for the computation (Section 4)
3. Techniques for determining the number of iterations required to yield a useful PageRank assignment are investigated.

2.2.1 Efficient Memory Usage

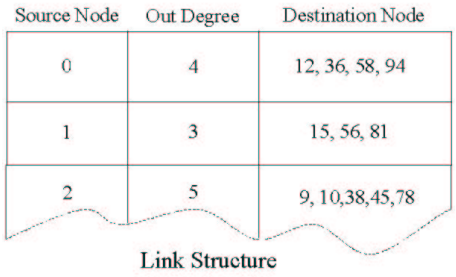

Figure 1: The Link Structure

A link structure record, Figure 1, is defined as: Source Node(32-bit id), Out Degree(16-bit integer), Destination Nodes(series of 32-bit id's). The naive technique for efficient memory usage is to store only the data of pages with outdegree greater than 0. An improvement is to use a block-based strategy. We partition the links file *Links* into β links buckets $Links_0, ..., Links_{\beta-1}$, such that the destination field in $Links_i$ contains only those destitions *dest* such that $\beta \times i \leq dest < \beta \times (i+1)$. In other words, the outgoing links of a node are bucketed according to the range that the identifier of the destination page falls into.

2.2.2 Accuracy

It was found that the use of single-precision Rank vectors does not lead to significant numerical error. A comparison of the residual value after 100 iterations using double precision and single precision showed that the difference was on the order of 10^{-7}.

2.2.3 Convergence Analysis

Though convergence can be measured in terms of the residual (the norm of the difference of the PageRank vectors from successive iterations), a more useful measure is in terms of the page ordering produced. The global ordering was analyzed using a histogram of the difference in positions of pages within two orderings, and a similarity measure between two ordered lists. The similarity measure was defined as the intersection of the two lists A and B divided by the union of the two. Histograms with a bucket size of 100 showed that the bulk of the pages occurred at similar positions for 50 iterations as 100 iterations. The similarity measure showed the ordering from 25 iterations agreed closely with the ordering from 100 iterations.

An even more useful measure is to look at rankings for a specific query. Instability in the global ranking tends to affect the relative rankings of unrelated pages. We really are only interested in the relative rankings of related pages. An analysis of the results for two queries using the similarity measure showed that the rankings from 10 iterations agreed fairly well with the rankings from 100 iterations.

2.3 Extrapolation Methods for Accelerating PageRank Computations

The original PageRank algorithm uses the Power Method to compute successive iterates that converge to the principal eigenvector of the Markov matrix representing the Web link graph. The algorithm proposed by Kamvar, Haveliwala, Manning and Golub [6], called Quadratic Extrapolation, accelerates the convergence of the Power Method by periodically subtracting off estimates of the nonprincipal eigenvectors from the current iterate of the Power Method.

2.4 Topic-Sensitive PageRank

Haveliwala [7] proposes computing a set of PageRank vectors for a set of topics. PageRanks are computed offline, then combined with query specific scores when a search is done. This approach minimizes query-time cost. The computation of topic specific PageRanks involves biasing the web link graph by adding artificial links.

A topic sensitive PageRank can be created by using a nonuniform damping vector E. In this case the URLs in categories of the Open Directory Project (ODP) were used. Let T_j be the set of URLs in the ODP category c_j. When computing the Pagerank vector for topic c_j, we use the nonuniform damping vector $E = v_j$ where

$$v_{ji} = \begin{cases} \frac{1}{|T_j|} & i \in T_j, \\ 0 & i \notin T_j, \end{cases}$$

At query time a linear combination of the topic-sensitive PageRanks, weighted by the similarity of the query to the categories is used in place of the general PageRank vector.

Two measures were used to analyze the results: the degree of overlap between the top n URLs of two rankings, and the similarity of the two rankings. Pairwise similarities of the different topic rankings were close to zero. A subjective analysis was also done, asking users to rate the search results comparing the unbiased PageRank with the topic-sensitive PageRank. Only the top three topic-sensitive vectors for the query were used when forming the linear combination PageRank. For the ten queries evaluated, in only one case was the unbiased PageRank preferred, and one case neither. In the other eight cases, users preferred the biased PageRank result.

In addition to the query term, context information can be used to further determine the sum of the weighted topic specific vectors. Some sources of this information are terms in the current page, browse history, or a hierarchical page directory. User context information such as browsing patterns, bookmarks, and email archives could also be used. This has several benefits over choosing a personalization vector directly. It is flexible, diverse sources of context information can be treated uniformly. It has an intuitive interpretation, and can be easily tuned. It can provide some measure of privacy, a client-side program could compute the user profile locally, and send only the weight vector to the server. Finally, it is efficient because the query-time cost is low because the PageRank vectors have been computed offline.

3 Methodology and Deliverables

In this project, we propose to implement parallel versions of PageRank algorithm on a distributed memory machine.

3.1 Web Graph Generation

First, we will build web graphs of different sizes by generating random links. It is assumed that the aver-

age number of link in a web page is 11[1].

3.2 Data Structures - Block Based Strategy

In a repository of the web containing roughly 25 million pages, there are over 81 million URLs in the link graph. Ignoring dangling links, pages with 0 out degree, yields a subgraph with close to 19 million nodes. Consecutive node id's are assigned to each URL starting from 0. If we use the link structure *Links*, suggested by Haveliwala [4], and store it on disk in a binary format, the size of the link structure, after the preprecessing steps above, is assumed to exceed the size of main memory of one processor.

On the parallel cluster, the memory requirement will be distributed over multiple processors. We create two arrays of floating point values representing the rank vectors, called *Source* and *Dest*. Each vector has N entries, where N is the number of nodes in our web graph. The rank values for iteration i are held in *Source*, and the rank values for iteration $i+1$ are constructed in *Dest*. In the block-oriented strategy, the *Dest* array is partitioned into D blocks each of size β pages, as illustrated in Figure 2. To reflect this

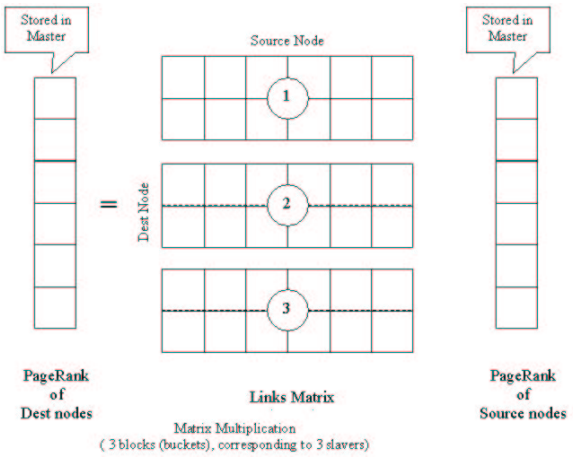

Figure 2: Blocked Multiplication

setup, the links file *Links* must be rearranged. We partition *Links* into β links buckets $Links_0, ..., Links_{\beta-1}$, such that the *destinations* field in $Links_i$ contains only those nodes *dest* such that $\beta \times i \leq dest < \beta \times (i+1)$. In other words, the outgoing links of a node are bucketed according to the range that the identifier of the destination page falls into. In each of the buckets, $Links_i$ is sorted on the *source* field, each node requires only one sequential pass through *Source* and the link information then stays in the memory. The partitioning of the links in Figure 1 for the case of three blocks is shown in Figure 3.

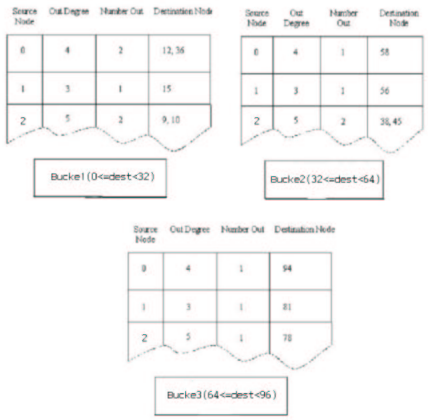

Figure 3: Partitioned Link File (bucket)

3.3 Parallel Architecture

The parallel architecture is as shown in Figure 4. First,

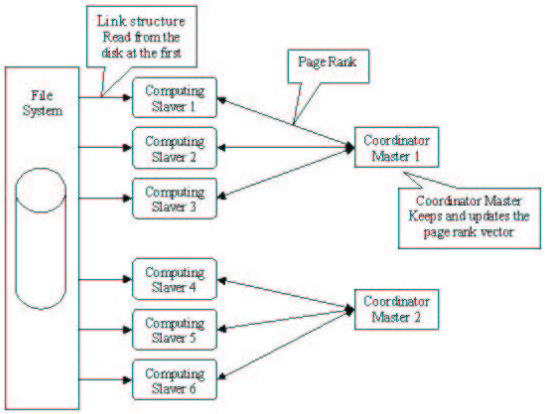

Figure 4: Parallel Architecture

since the computational task for each node is sorted by the destination id, each computing node (left hand side) reads the link structure needed from the input data on hard disk, and stores it in it's memory. The PageRank values are initialized and stored on the master nodes (right hand side). They are divided among multiple masters such that two arrays holding the partial PageRank values from the preious and current it-

erations can fit in the memory. Chen and Zhang [8] showed that parallel computation of pagerank without global synchronization eventually converges to the same equilibrium value while better speedup can be achieved. During the iteration (computation), each computing node contribute its part of the PR vector to the corresponding master node, and proceeds to the next iteration using the most currently available PageRanks.

Upon convergence, the master nodes write the final PageRank to the disk.

3.4 Algorithm

In the parallel block-oriented strategy, the algorithm for master nodes proceeds as follows:

(for master i)
initialize $PR(j)$, $j \in [\beta j, \beta(j+1))$
read the slave-source list from the disk
while $residual > \delta$
    send PR to slaves
    receive new PR from all slaves
    calculate residual
stop the slaves
write PR to disk

And the algorithm for slaves is as follows:

(for slave i)
read its sub link structure from the disk
repeat
    receive PR from masters
    for each $\beta i \leq j < \beta(i+1)$ compute $PR(j)$
    send $PR[\beta i, \cdots, \beta(i+1)]$ to masters
until 'stop' from the master

4 Implementation

4.1 Input Data Generation

The input Data files representing the link structures, as shown in Figure 1, are manually genearted. The number of outgoing links is assumed to satisfy a normal distribution with its average per page being 11[1]. By generating suedo random numbers, we write the raw data files on disk in binary format.

4.2 Programming and Test Problems

All the test problems are run on the BlueHorizon parallel machines at the San Diego Super Computer Center. Test problems range from Cleve Moler's simple example with only 6 pages[5], to a web graph containing 7 million pages. Both MPI and sequential programs are written in C.

5 Results and Analysis

5.1 Validity Verification

To check the validity of our programs, we ran both the sequential program and MPI program on Cleve's simple example with 6 pages[5], and compared the results. The sequential program and MPI program yield the same results and, as shown in Figure 5, they exactly match the result given by Cleve.

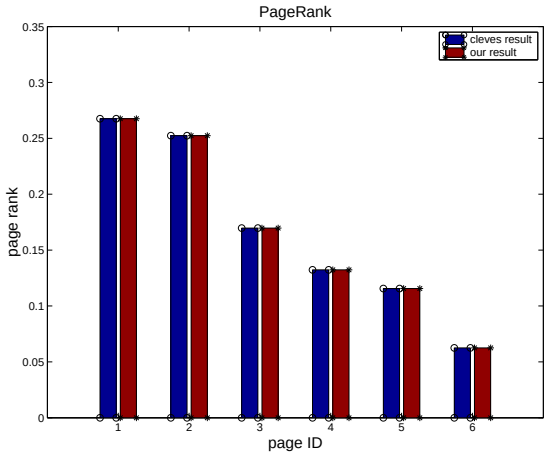

Figure 5: Validity Test

5.2 Convergence

Since the web graph we generated is irreducible and contains no rank sinks, the convergence is fast. In Figure 6, the residual (1 norm of the residual vector) decreases very fast. When the tolerance is set as 10^{-10}, the convergence is achieved after about 45 iterations.

5.3 Performance Analysis

5.3.1 Performance Metric – Memory Usage

When we fix N,the total number of pages, the memory usage by the slave dominates and decreases with increasing number of processors. Figure 7 shows the decreased memory usage with increasing N. The total number of pages is one million, and the memory usage by the master is alway $13.22MB$ (with different

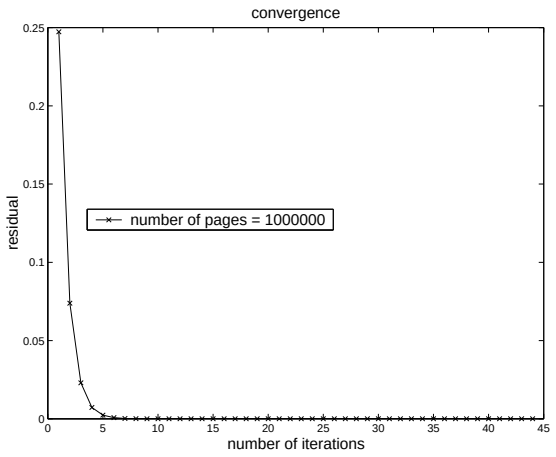

Figure 6: Convergence of PageRank with one million pages

$N's$). Note the slight increase of memory usage when we have one master and one slave. This is expected due to the extra storage for communication buffer.

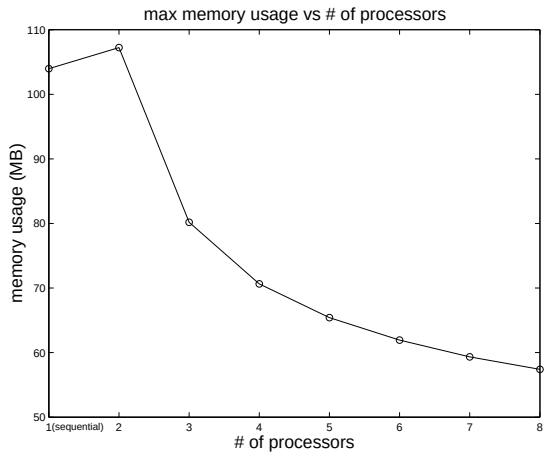

Figure 7: Memory Usage

5.3.2 Performance Metric – Running Time

The total running time for both sequential and parallel programs, when $N = 1,000,000$ is fixed, is shown in Figure 8. Figure 9 shows the speedup. We see reduced speedup when having more than 4 processors, due to the increasing communication cost.

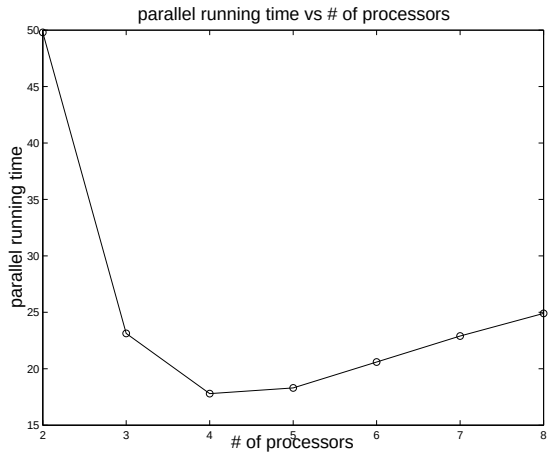

Figure 8: Running time when N=1000000

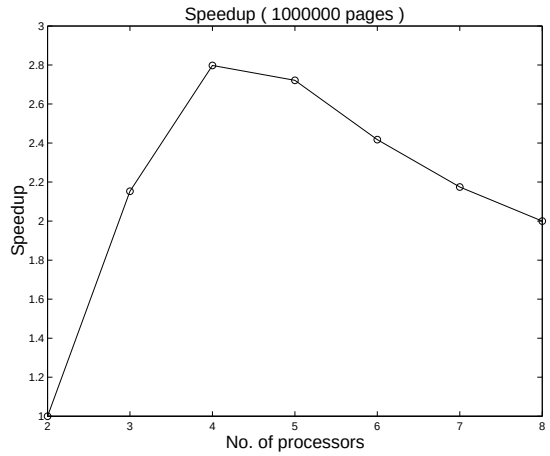

Figure 9: Speedup when N=1000000

5.3.3 Performance Metric – Floating Point Operations

Similar to the memory usage, the number of floating point operations is always the same when we increase the number of processors, but fix N, and the bottleneck is at the slaves. Figure 10 shows the decrease of the number of floating point operations, with an increasing number of processors.

5.3.4 Scalability

When the problem size (N) increases, from one million to 7 million, our program scales well. As shown in Table 1, the performance scales pretty well. Here, 5 processors are used.

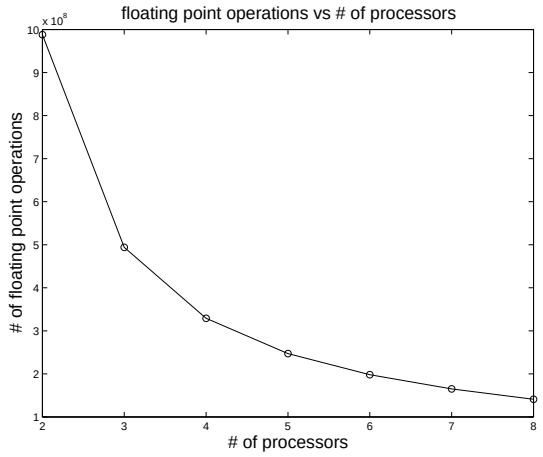

Figure 10: Floating point operations when N=1000000

N (10^6)	floating pnt ops	time (s)	memory(MB)
1	2.47×10^8	18.3	65.3
2	4.94×10^8	40.1	125.5
5	1.23×10^9	116.8	305.6
7	1.66×10^9	1070.6	421.3

Table 1: Performance with increasing data size

6 Conclusions

Although the PageRank algorithm is based on a simple idea, scaling it's implementation to operate on large graphs of the web requires careful treatment. In this project, we have demonstrated that PageRank can be computed on parallel machines with modest amount of memory, and appreciable speedup can be achieved. But to resolve memory constraint on each single processor when data size further increases, running time will be sacrificed and hence less speedup.

7 Acknowledgements

Advice from Cleve Moler is appreciated.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winorgrad. "The PageRank Citation Ranking: Bringing Order to the Web". *Stanford Digital Libraries Working Paper*, 1998.

[2] S. Brin, L. Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine". In *Proceedings of the Seventh International World Wide Web Conference*, 1998.

[3] R. Motwani and P. Raghavan. "Randomized Algorithms". Cambridge University Press, United Kingdom, 1995.

[4] T. H. Haveliwala. "Efficient Computation of PageRank". *Stanford University Technical Report*, 1999.

[5] C. B. Moler and K. A. Moler. "Numerical Computing with MATLAB". CS110A online textbook, UCSB.

[6] S. D. Kamvar, T. H. Haveliwala, C. D. Manning and G. H. Golub. "Extrapolation Methods for Accelerating PageRank Computations". *Stanford University Technical Report*, 2002.

[7] T. H. Haveliwala. "Topic-sensitive PageRank". In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[8] Y. Chen and H. Zhang. "Parallelization of the Page Ranking in the Google Search Engine". October 1999. none.