A Web-Site-Based Partitioning Technique for Reducing Preprocessing Overhead of Parallel PageRank Computation*

Ali Cevahir, Cevdet Aykanat, Ata Turk, and B. Barla Cambazoglu

Bilkent University, Department of Computer Engineering,
TR-06800 Bilkent, Ankara, Turkey
{acevahir, aykanat, atat, berkant}@cs.bilkent.edu.tr

Abstract. A power method formulation, which efficiently handles the problem of dangling pages, is investigated for parallelization of PageRank computation. Hypergraph-partitioning-based sparse matrix partitioning methods can be successfully used for efficient parallelization. However, the preprocessing overhead due to hypergraph partitioning, which must be repeated often due to the evolving nature of the Web, is quite significant compared to the duration of the PageRank computation. To alleviate this problem, we utilize the information that sites form a natural clustering on pages to propose a site-based hypergraph-partitioning technique, which does not degrade the quality of the parallelization. We also propose an efficient parallelization scheme for matrix-vector multiplies in order to avoid possible communication due to the pages without in-links. Experimental results on realistic datasets validate the effectiveness of the proposed models.

1 Introduction

PageRank is a popular algorithm used for ranking Web pages by utilizing the hyperlink structure among the pages. PageRank algorithm usually employs the random surfer model [21], which can be described as a Markov chain, where the PageRank values of pages can be computed by finding the stationary distribution of this chain. Traditionally, PageRank computation is formulated as finding the principal eigenvector of the Markov chain transition matrix and solved using the iterative power method. Recently, linear system formulations and associated iterative solution methods [3, 9, 18] are investigated for PageRank computation as well. In both types of formulations, PageRank computation can be accelerated via parallelization [9, 20] or increasing the convergence rate of the iterative methods [5, 12, 15, 16, 19].

The focus of this work is on reducing the per iteration time through parallelization. Among several formulations [13, 15, 17, 18] proposed for handling the dangling-page (pages without out-links) problem, widely used formulation

* This work is partially supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under project EEEAG-106E069.

B. Kågström et al. (Eds.): PARA 2006, LNCS 4699, pp. 908–918, 2007.

of Kamvar et al. [15] is selected for parallelization. In this formulation, which is based on the power method, the kernel operations are sparse-matrix vector multiply and linear vector operations. The partitioning scheme adopted in our parallelization is based on rowwise partitioning of the transition matrix and conformable partitioning of the linear vectors used in the iterative power method.

Recently, the hypergraph-partitioning-based sparse matrix partitioning method of Catalyurek and Aykanat [6, 7] is applied by Bradley et al. [4] for efficient parallelization of the above-mentioned power method formulation. This parallelization technique successfully reduces the communication overhead while maintaining computational load balance. However, the preprocessing overhead due to hypergraph partitioning, which must be repeated often due to constantly evolving nature of the Web, is quite significant compared to the duration of the PageRank computation.

In this work, we investigate techniques for reducing the overhead of the preprocessing step before the PageRank computation without degrading the quality of the parallelization. To this end, we propose a site-based compression on the rows of the transition matrix relying on the expectation that sites form a natural clustering on pages. Then, the conventional hypergraph model [6, 7] is applied on the compressed site-to-page transition matrix to induce a partitioning on the original page-to-page transition matrix. We also propose an efficient parallelization scheme for matrix-vector multiplies in order to avoid possible communication due to the pages without in-links. Furthermore, we extend the hypergraph-partitioning model to encapsulate both this efficient parallelization scheme and the computational load balance over the whole iterative algorithm. Experimental results on realistic Web datasets verify the validity of the proposed models. The proposed site-based partitioning scheme reduces the preprocessing time drastically compared to the page-based scheme while producing better partitions in terms of communication volume. Our implementation for the proposed parallel PageRank algorithm shows that site-based partitioning scheme leads to better speedup values compared to the page-based scheme on a 32- node PC cluster.

The rest of the paper is organized as follows. Section 2 summarizes the PageRank algorithm. The proposed parallelization scheme is discussed in Section 3. Section 4 describes the proposed page-based and site-based partitioning schemes. Experimental results are presented in Section 5. Finally, concluding remarks are given in Section 6.

2 PageRank Algorithm

PageRank can be explained with a probabilistic model, called the random surfer model. Consider a Web user randomly visiting pages by following out-links within pages. Let the surfer visit page i at a particular time step. In the next time step, the surfer chooses to visit one of the pages pointed by the out-links of page i at random. If page i is a dangling page, then the surfer jumps to a random page. Even if page i is not a dangling page, the surfer may prefer to jump to a random page with a fix probability instead of following one of the out-links of page i.

In the random surfer model, the PageRank of page i can be considered as the (steady-state) probability that the surfer is at page i at some particular time step. In the Markov chain induced by the random walk on the Web containing n pages, states correspond to the pages in the Web and the $n \times n$ transition matrix $\mathbf{P} = (p_{ij})$ is defined as $p_{ij} = 1/deg(i)$, if page i contains out-link(s) to page j, and 0, otherwise. Here, $deg(i)$ denotes the number of out-links within page i.

A row-stochastic transition matrix $\mathbf{P'}$ is constructed from $\mathbf{P}$ as $\mathbf{P'} = \mathbf{P} + \mathbf{d}\mathbf{v}^T$ via handling of dangling pages according to the random surfer model. Here, $\mathbf{d} = (d_i)$ and $\mathbf{v} = (v_i)$ are column vectors of size n. $\mathbf{d}$ identifies dangling pages, i.e., $d_i = 1$ if row i of $\mathbf{P}$ corresponds to a dangling page, and 0, otherwise. $\mathbf{v}$ is the teleportation (personalization) vector which denotes the probability distribution of destination pages for a random jump. Uniform teleportation vector $\mathbf{v}$, where $v_i = 1/n$ for all i, is used for generic PageRank computation [14]. Non-uniform teleportation vectors can be used for achieving topical or personalized PageRank computation [11, 21], or preventing link spamming [10].

Although $\mathbf{P'}$ is row-stochastic, it may not be irreducible. For example, the Web contains many pages without in-links, which disturb irreducibility. An irreducible Markov matrix $\mathbf{P''}$ is constructed as $\mathbf{P''} = \alpha\mathbf{P'} + (1-\alpha)\mathbf{e}\mathbf{v}^T$, where $\mathbf{e}$ is a column vector of size n containing all ones. Here, α represents the probability that the surfer chooses to follow one of the out-links of the current page, and $(1 - \alpha)$ represents the probability that surfer makes a random jump instead of following the out-links.

Given $\mathbf{P''}$, PageRank vector $\mathbf{r}$ can be determined by computing the stationary distribution for the Markov chain, which satisfies the equation $(\mathbf{P''})^T\mathbf{r} = \mathbf{r}$. This corresponds to finding the principal eigenvector of matrix $\mathbf{P''}$. Applying the power method directly for the solution of this eigenvector problem leads to a sequence of matrix-vector multiplies $\mathbf{p}^{k+1} = (\mathbf{P''})^T\mathbf{p}^k$, where $\mathbf{p}^k$ is the kth iterate towards the PageRank vector $\mathbf{r}$. However, matrix $\mathbf{P''}$ is completely dense, whereas original $\mathbf{P}$ is sparse. Kamvar et al. [15] propose an efficient multiplication scheme by reformulating the multiplication with dense matrix $(\mathbf{P''})^T$ in terms of sparse $\mathbf{P}^T$. This efficient PageRank algorithm is given in Fig. 1.

3 Parallel PageRank Algorithm

Two basic types of operations are performed repeatedly at each iteration of the PageRank algorithm given in Fig. 1. The first type is sparse-matrix vector multiply (i.e., $\mathbf{q} \leftarrow \alpha\mathbf{A}\mathbf{p}$), and the second type is linear vector operations, such as L_1 norm (e.g., $\|\mathbf{q}\|_1$), DAXPY (i.e., $\mathbf{q} \leftarrow \mathbf{q} + \gamma\mathbf{v}$) and vector subtraction (i.e., $\mathbf{q} - \mathbf{p}$). We consider the parallelization of the computations of the PageRank algorithm through rowwise partitioning of the $\mathbf{A}$ matrix as $\mathbf{A} = [\mathbf{A}_1^T \cdots \mathbf{A}_k^T \cdots \mathbf{A}_K^T]^T$, where processor P_k stores row stripe $\mathbf{A}_k$. All vectors (e.g., $\mathbf{p}$ and $\mathbf{q}$) used in the algorithm are partitioned conformably with the row partition of $\mathbf{A}$ to avoid communication of the vector components during linear vector operations. That is, the $\mathbf{p}$ and $\mathbf{q}$ vectors are partitioned as $[\mathbf{p}_1^T \cdots \mathbf{p}_K^T]^T$ and $[\mathbf{q}_1^T \cdots \mathbf{q}_K^T]^T$, respectively. Processor P_k is responsible for performing the local matrix-vector

PageRank($\mathbf{A}$, $\mathbf{v}$)
1. $\mathbf{p} \leftarrow \mathbf{v}$
2. **repeat**
3. $\quad \mathbf{q} \leftarrow \alpha\mathbf{A}\mathbf{p}$
4. $\quad \gamma \leftarrow ||\mathbf{p}||_1 - ||\mathbf{q}||_1$
5. $\quad \mathbf{q} \leftarrow \mathbf{q} + \gamma\mathbf{v}$
6. $\quad \delta \leftarrow ||\mathbf{q} - \mathbf{p}||_1$
7. $\quad \mathbf{p} \leftarrow \mathbf{q}$
8. **until** $\delta < \varepsilon$
9. **return** $\mathbf{p}$

Fig. 1. Efficient PageRank algorithm based on the power method: $\mathbf{A} = \mathbf{P}^T$ is the transition matrix, $\mathbf{v}$ is the teleportation vector, and ε is the convergence threshold

multiply $\mathbf{q}_k \leftarrow \alpha\mathbf{A}_k\mathbf{p}$ while holding $\mathbf{p}_k$. Processor P_k is also responsible for the linear vector operations on the kth blocks of the vectors.

In this scheme, the linear vector operations can be efficiently performed in parallel such that only the norm operations require global communication overhead. Fortunately, the volume of communication incurred due to this global communication does not increase with increasing n, and it is only $K-1$ words. On the other hand, depending on the way in which rows of $\mathbf{A}$ are partitioned among the processors, entries in $\mathbf{p}$ may need to be communicated among the processors before the local matrix-vector multiplies, hence this scheme can be considered as a pre-communication scheme. During the pre-communication phase, a processor P_k may be sending the same $\mathbf{p}_k$-vector entry to different processors according to the sparsity pattern of the respective column of $\mathbf{A}$. This multicast like operation is referred to here as *Expand* operation. Note that the communication requirement during the pre-communication may be as high as $(K-1)n$ words and $K(K-1)$ messages, and the communication occurs when each sub-matrix $\mathbf{A}_k$ has at least one nonzero in each column.

As seen in Fig. 1, PageRank algorithm requires two global communication operations in the form of all-to-all reduction due to the norm operations at steps 4 and 6 in Fig. 1. The global operations may incur high communication overhead in parallel architectures with high message latency. In this work, we propose a coarse-grain parallel PageRank algorithm, which reduces the number of global communication operations at each iteration from two to one by rearranging the computations as shown in Fig. 2. Here, two global norms are accumulated at all processors in a single all-to-all reduction operation performed at step 5(c) in Fig. 2. Hence, the proposed coarse-grain formulation halves the latency overhead while keeping the communication volume the same. The only drawback of this formulation is that it will perform an extra iteration compared to the power method formulation given in Fig. 1, because the convergence check is applied on the $\mathbf{p}$ vectors of the previous two iterations. In Fig. 2, a superscript k denotes the partial result computed by processor P_k, e.g., γ^k is the partial result for global scalar γ, where $\gamma = \sum_{k=1}^{K} \gamma^k$.

Parallel-PageRank($\mathbf{A}_k$, $\mathbf{v}_k$)
1. $\mathbf{p}_k \leftarrow \mathbf{v}_k$
2. $\mathbf{t}_k \leftarrow \mathbf{0}$
3. **repeat**
4. (a) $\mathbf{p} \leftarrow \text{Expand}(\mathbf{p}_k)$
   (b) $\mathbf{q}_k \leftarrow \alpha \mathbf{A}_k \mathbf{p}$
5. (a) $\gamma^k \leftarrow \|\mathbf{p}_k\|_1 - \|\mathbf{q}_k\|_1$
   (b) $\delta^k \leftarrow \|\mathbf{p}_k - \mathbf{t}_k\|_1$
   (c) $\langle \gamma, \delta \rangle \leftarrow \text{AllReduceSum}(\langle \gamma^k, \delta^k \rangle)$
6. $\mathbf{t}_k \leftarrow \mathbf{p}_k$
7. $\mathbf{p}_k \leftarrow \mathbf{q}_k + \gamma \mathbf{v}_k$
8. **until** $\delta < \varepsilon$
9. **return** $\mathbf{p}_k$

Fig. 2. Coarse-grain parallel PageRank algorithm (pseudocode for processor P_k)

Web data may contain many pages without in-links [3]. This property can be utilized to increase the efficiency of the parallel PageRank algorithm as follows. Since pages without in-links correspond to zero rows of matrix $\mathbf{A}$, the matrix-vector multiply at step 4 of Fig. 2 results in zero values for the respective $\mathbf{q}$-vector entries. Hence, for each page i without in-links, p_i iterate can be simply updated as γv_i instead of the DAXPY operation at step 7. Note that v_i is a constant throughout the iterations and γ is a global scalar computed and stored at all processors at each iteration of the algorithm. Hence, possible expand communications of p_i due to the sparsity pattern of column i of $\mathbf{A}$ can be totally avoided as follows: We replicate v_i among the processors that have at least one row with a non-zero at column i at the very beginning of the algorithm and then enforce each one of those processors to redundantly compute $p_i = \gamma v_i$ at each iteration of the algorithm.

4 Rowwise Partitioning

The objective in the proposed parallelization is to find a rowwise partition of $\mathbf{A}$ that minimizes the volume of communication during each sparse matrix-vector multiply while maintaining the computational load balance during each iteration.

4.1 Page-Based Partitioning

Rowwise partitioning of irregularly sparse matrices for the parallelization of matrix-vector multiplies is formulated using the hypergraph-partitioning model [6, 7]. In the column-net model proposed for rowwise partitioning [6, 7], a given matrix is represented as a hypergraph which contains a vertex for each row and a net for each column. The net corresponding to a column connects the vertices corresponding to the rows that have a non-zero at that column. The vertices connected by a net are said to be its pins. Vertices are associated with weights which are set equal to the number of non-zeros in the respective rows.

A K-way vertex partition on the hypergraph is decoded as assigning the rows corresponding to the vertices in each part of the partition to a distinct processor. Partitioning constraint on balancing the part weights corresponds to balancing the computational loads of processors, whereas partitioning objective of minimizing the cutsize corresponds to minimizing the total communication volume during a parallel matrix-vector multiply.

In this work, we adopt the hypergraph-partitioning model and extend it to encapsulate both the computational load balance over the whole iterative algorithm and the efficient parallelization scheme described in Section 3. For this purpose, we reorder the rows and columns of matrix **A** in such a way that rows and columns corresponding to the pages without in-links are permuted to the end. Then, we decompose the reordered transition matrix **A** as follows:

$$\mathbf{A} = \begin{array}{|c|c|} \hline \mathbf{C} & \mathbf{W} \\ \hline \multicolumn{2}{|c|}{\mathbf{Z}} \\ \hline \end{array}$$

Here, rows of sub-matrix **Z** and columns of sub-matrix **W** correspond to the pages without in-links. Note that **Z** is a sub-matrix containing all zeros. We compute a rowwise partition of **A** in two phases. The first and second phases respectively incorporate the partitioning of the PageRank computation for the pages with and without in-links among the processors.

In the first phase, we obtain a K-way row partition of sub-matrix $[\mathbf{C} \mid \mathbf{W}]$ by partitioning the column-net representation $\mathcal{H}(\mathbf{C})$ of the **C** sub-matrix. $\mathcal{H}(\mathbf{C})$ contains one vertex and one net for each row and non-zero column of sub-matrix **C**, respectively. Note that no nets are introduced for zero-columns which correspond to dangling pages. Vertices of $\mathcal{H}(\mathbf{C})$ are weighted to incorporate the floating point operations (flops) associated with the non-zeros of both **C** and **W** sub-matrices as well as the flops associated with the linear-vector operations. That is the weight of vertex i is set equal to: $2{\times}nnz(row\ i\ of\ [\mathbf{C} \mid \mathbf{W}])+7$. The first term accounts for the number of flops associated with row i during a matrix-vector multiply operation since each matrix non-zero incurs one multiply and one add operation. The second term accounts for the number of flops associated with the linear vector operations performed on the ith entries of the vectors.

In the second phase, rows of sub-matrix **Z** are distributed among the parts of the rowwise partition obtained at the end of the first phase. Although it may seem awkward to mention about distributing zero-rows across processors, recall that partitioning of rows also incur the assignment of the respective vector entries and the associated linear vector operations. The number of linear vector operations performed on the vector entries corresponding to the zero rows reduces from 7 to 4 flops since the respective **q**-vector entries remain as zero. The only metric considered during this distribution is to improve the balance of the partition obtained in the first phase.

4.2 Site-Based Partitioning

Experimental results show that the page-based partitioning technique proposed in Section 4.1 is quite successful in minimizing the total volume of communication during the parallel PageRank computation. However, the preprocessing overhead incurred in the first phase of the page-based scheme is quite significant due to the partitioning of the hypergraph representing the page-to-page **C** sub-matrix, which is very large in practice. For example, the time elapsed for 16-way partitioning of the `Google` and `In-2004` datasets are as high as the time elapsed for 64 and 47 iterations of the PageRank computations performed for the respective datasets.

In this section, we propose a technique to reduce this overhead by partitioning a compressed version, $\bar{\mathbf{C}}$, of the page-to-page sub-matrix **C**. We generate the site-to-page $\bar{\mathbf{C}}$ matrix exploiting the fact that Web sites form a natural clustering of pages. In matrix $\bar{\mathbf{C}}$, rows correspond to Web sites while columns corresponds to pages. The union of the non-zeros of the **C**-rows that correspond to the pages residing in a site form the non-zeros of the $\bar{\mathbf{C}}$-row corresponding to that site. Then, we apply the column-net model on $\bar{\mathbf{C}}$ to obtain the $\mathcal{H}(\bar{\mathbf{C}})$, and partition $\mathcal{H}(\bar{\mathbf{C}})$ instead of $\mathcal{H}(\mathbf{C})$. The weight of a vertex j in $\mathcal{H}(\bar{\mathbf{C}})$ corresponding to site j is set equal to

$$\sum_{\text{page i} \in \text{site j}} (2 \times nnz(\text{row i of } [\mathbf{C} \mid \mathbf{W}])) + 7 \times p_{\text{in}-\text{links}}(\text{site j}),$$

where $p_{\text{in}-\text{links}}(\text{site j})$ denotes the number of pages with at least one in-link in site j.

Although the compression of **C** reduces the number of vertices in $\mathcal{H}(\bar{\mathbf{C}})$ significantly, the compression does not reduce the number of nets. However, experimental results show that $\mathcal{H}(\bar{\mathbf{C}})$ contains many nets that connect a single vertex. These single-pin nets correspond to the pages that have out-links only to the sites they belong to. Since single-pin nets have no potential to incur cost to the cutsize, they can be discarded from $\mathcal{H}(\bar{\mathbf{C}})$ before the partitioning. Removal of single-pin nets significantly reduces the number of total nets in $\mathcal{H}(\bar{\mathbf{C}})$. The partitioning time of $\mathcal{H}(\bar{\mathbf{C}})$ is expected to be much less than that of $\mathcal{H}(\mathbf{C})$, since $\mathcal{H}(\bar{\mathbf{C}})$ contains significantly fewer vertices and nets.

5 Experimental Results

In our experiments, two datasets with different sizes are used. The `Google`[1] dataset is provided by Google and includes .edu domain pages in the US. The `In-2004`[2] dataset is crawled by UbiCrawler and includes pages from the Web of India. The properties of these datasets are given in Table 1. For the convergence threshold of $\epsilon = 10^{-8}$, the PageRank computations converge in 91 and 90

[1] `http://www.google.com/programming-contest/`
[2] `http://law.dsi.unimi.it/`

Table 1. Properties of datasets and column-net hypergraph representations of the respective $\mathbf{C}$ and $\bar{\mathbf{C}}$ matrices

		Google	In-2004
# of pages		913,569	1,347,446
# of pages w/o in-links		132,167	86
# of sites		15,819	4,376
# of links		4,480,218	13,416,945
% intra-site links		87.42	95.92
% inter-site links		12.58	4.08
Page-based hypergraph	**# of vertices**	781,402	1,347,446
	# of nets	608,769	1,065,161
	# of pins	4,741,970	14,482,106
Site-based hypergraph	**# of vertices**	15,819	4,376
	# of nets	214,659	205,106
	# of pins	600,952	555,195

iterations for the Google and In-2004 datasets, respectively. In the PageRank computations, the damping factor α is set to 0.85, conforming with the usual practice [3].

Table 1 also shows the properties of the column-net hypergraphs $\mathcal{H}(\mathbf{C})$ and $\mathcal{H}(\bar{\mathbf{C}})$, which represent the $\mathbf{C}$ and $\bar{\mathbf{C}}$ matrices, respectively. As seen in Table 1, the proposed compression scheme leads to a significant decrease in the size of the hypergraphs. For example, in the In-2004 dataset, approximately 99%, 80% and 96% reductions are obtained in the number of vertices, nets and pins, respectively. Direct K-way hypergraph partitioning tool kPaToH [2, 8] is used, with default parameters and an imbalance tolerance of 3%, for partitioning the hypergraphs. As kPaToH involves randomized heuristics, kPaToH is run ten times with different seed values for each partitioning instance and the averages of those results are reported in the following figures. The partitioning operations are performed on an Intel Pentium IV 3.0 GHz processor with 2 GB of RAM.

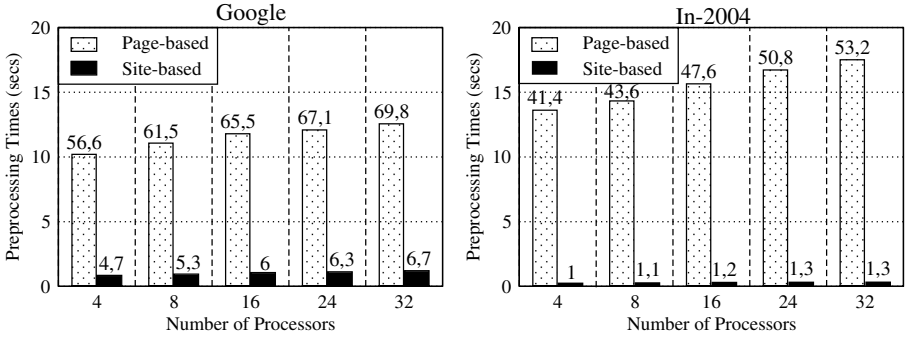

Fig. 3. Preprocessing times in seconds

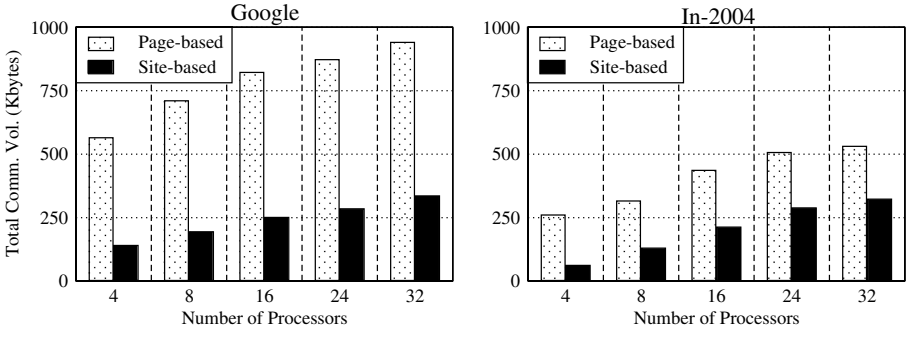

Fig. 4. Total communication volumes in Kbytes

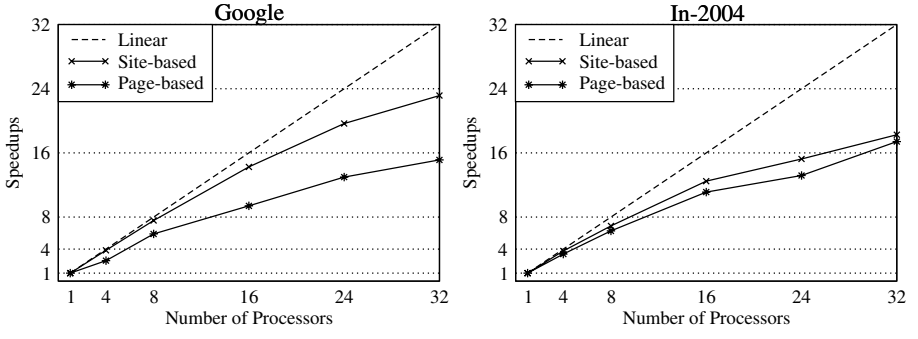

Fig. 5. Speedup curves

Fig. 3 displays the variation of the preprocessing times of page-based and site-based partitioning schemes with increasing number of processors. For the page-based scheme, the preprocessing time involves only the partitioning time, whereas for the site-based scheme, it involves both compression and partitioning times. As seen in Fig. 3, the proposed site-based partitioning scheme achieves a drastic reduction in the preprocessing time compared to the page-based scheme. For example, the site-based scheme performs the preprocessing approximately 11 and 40 times faster than the page-based scheme in partitioning the `Google` and `In-2004` datasets, on the overall average. In Fig. 3, the number annotated with each bar shows the ratio of preprocessing time to the sequential per iteration time. According to these values, in the `In-2004` dataset, the preprocessing time of the site-based scheme is approximately equal to a single iteration time of the sequential PageRank computation.

Fig. 4 displays the quality of the partitions obtained by the page-based and site-based partitioning schemes in terms of total communication volume. As seen in the figure, the partitions obtained by the site-based scheme incurs 70%

and 54% less communication volume than those of the page-based scheme for the `Google` and `In-2004` datasets, respectively. These experimental findings verify the expectation that sites constitute natural clusters of pages.

In order to compare the speedup performances of the page- and site-based partitioning schemes, the parallel PageRank algorithm proposed in Section 3 is implemented using the ParMxvLib library [22]. The parallel PageRank algorithm is run on a 32-node PC cluster interconnected by a Fast Ethernet switch, where each node contains an Intel Pentium IV 3.0 GHz processor, 1 GB of RAM. The speedup curves are given in Fig. 5. As seen in the figure, the site-based partitioning scheme leads to higher speedup values than the page-based scheme, in accordance with the reduction in the communication volumes. For example, the site-based scheme leads to a speedup of approximately 24 on 32 processors, whereas the page-based scheme achieves a speedup of 16.

6 Conclusion

An efficient parallelization technique for PageRank computation was proposed and implemented. Experimental results show that, compared to a state-of-the-art parallelization scheme, the proposed technique not only reduces the preprocessing time drastically, but also reduces the parallel per iteration time. Although the proposed parallelization scheme is applied on a particular power method formulation, the underlying ideas can be easily and effectively applied to the parallelization of other iterative method formulations investigated in the literature for PageRank computation.

References

1. Aykanat, C., Pinar, A., Catalyurek, U.V.: Permuting sparse rectangular matrices into block-diagonal form. SIAM J. Scientific Computing 25(6), 1860–1879 (2004)
2. Aykanat, C., Cambazoglu, B.B., Ucar, B.: Multilevel hypergraph partitioning with multiple constraints and fixed vertices. J. Parallel and Distributed Computing. (submitted)
3. Berkhin, P.: A survey on PageRank computing. Internet Mathematics 2(1), 73–120 (2005)
4. Bradley, J.T., Jager, D.V., Knottenbelt, W.J., Trifunovic, A.: Hypergraph partitioning for faster parallel PageRank computation. In: Bravetti, M., Kloul, L., Zavattaro, G. (eds.) Formal Techniques for Computer Systems and Business Processes. LNCS, vol. 3670, pp. 155–171. Springer, Heidelberg (2005)
5. Brezinski, C., Redivo-Zaglia, M., Serra Capizzano, S.: Extrapolation methods for PageRank computations. Comptes Rendus de l'Académie des Sciences de Paris, Series I 340, 393–397 (2005)
6. Catalyurek, U.V., Aykanat, C.: Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. In: Saad, Y., Yang, T., Ferreira, A., Rolim, J.D.P. (eds.) IRREGULAR 1996. LNCS, vol. 1117, pp. 75–86. Springer, Heidelberg (1996)
7. Catalyurek, U.V., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Transactions on Parallel and Distributed Systems 10(7), 673–693 (1999)

8. Catalyurek, U.V., Aykanat, C.: A multilevel hypergraph partitioning tool, version 3.0. Tech. Rep., Bilkent University (1999)
9. Gleich, D., Zhukov, L., Berkhin, P.: Fast parallel PageRank: A linear system approach. Tech. Rep. YRL-2004-038, Yahoo! (2004)
10. Gyöngyi, Z., Garcia-Molina, H., Pedersen, J.: Combating Web spam with TrustRank. In: Proc. 30th Int'l Conf. on VLDB, pp. 576–587 (2004)
11. Haveliwala, T.: Topic sensitive PageRank. In: Proc. 11th Int'l WWW Conf., pp. 517–526 (2002)
12. Ipsen, I.C.F., Kirkland, S.: Convergence analysis of a PageRank updating algorithm by Langville and Meyer. SIAM J. Matrix Anal. Appl. 27, 952–967 (2006)
13. Ipsen, I.C.F., Selee, T.M.: PageRank computation, with special attention to dangling nodes. SIAM J. Matrix Anal. Appl. (2007) (submitted)
14. Ipsen, I.C.F., Wills, R.S.: Mathematical properties and analysis of Google's PageRank. Bol. Soc. Exp. May. Apl. 34, 191–196 (2006)
15. Kamvar, S., Haveliwala, T., Manning, C., Golub, G.: Extrapolation methods for accelerating PageRank computations. In: Proc. 12th Int'l WWW Conf., pp. 261–270 (2003)
16. Kamvar, S., Haveliwala, T., Golub, G.: Adaptive methods for computation of PageRank. In: Proc. Int'l Conf. on the Numerical Solution of Markov Chains (2003)
17. Kamvar, S., Haveliwala, T., Manning, C., Golub, G.: Exploiting the block structure of the Web for computing PageRank. Tech. Rep., Stanford Univ. (2003)
18. Langville, A.N., Meyer, C.D.: Deeper inside PageRank. Internet Mathematics 1(3), 335–380 (2005)
19. Langville, A.N., Meyer, C.D.: A reordering for the PageRank problem. SIAM J. Scientific Computing 27(6), 2112–2120 (2006)
20. Manaskasemsak, B., Rungsawang, A.: Parallel PageRank computation on a gigabit PC cluster. In: Proc. AINA'04, pp. 273–277 (2004)
21. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the Web. Tech. Rep. 1999-66, Stanford Univ. (1999)
22. Ucar, B., Aykanat, C.: A library for parallel sparse matrix-vector multiplies. Tech. Rep. BU-CE-0506, Department of Computer Engineering, Bilkent University, Ankara, Turkey (2005)
23. Ucar, B., Aykanat, C.: Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for matrix-vector multiplies. SIAM J. Scientific Computing. 25(6), 1837–1859 (2004)