Site-Based Partitioning and Repartitioning Techniques for Parallel PageRank Computation

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

Abstract—The PageRank algorithm is an important component in effective web search. At the core of this algorithm are repeated sparse matrix-vector multiplications where the involved web matrices grow in parallel with the growth of the web and are stored in a distributed manner due to space limitations. Hence, the PageRank computation, which is frequently repeated, must be performed in parallel with high-efficiency and low-preprocessing overhead while considering the initial distributed nature of the web matrices. Our contributions in this work are twofold. We first investigate the application of state-of-the-art sparse matrix partitioning models in order to attain high efficiency in parallel PageRank computations with a particular focus on reducing the preprocessing overhead they introduce. For this purpose, we evaluate two different compression schemes on the web matrix using the site information inherently available in links. Second, we consider the more realistic scenario of starting with an initially distributed data and extend our algorithms to cover the repartitioning of such data for efficient PageRank computation. We report performance results using our parallelization of a state-of-the-art PageRank algorithm on two different PC clusters with 40 and 64 processors. Experiments show that the proposed techniques achieve considerably high speedups while incurring a preprocessing overhead of several iterations (for some instances even less than a single iteration) of the underlying sequential PageRank algorithm.

Index Terms—PageRank, sparse matrix-vector multiplication, web search, parallelization, sparse matrix partitioning, graph partitioning, hypergraph partitioning, repartitioning.

1 Introduction

PageRank [13] is a very well-known algorithm that attracted the attention of the information retrieval community in the last decade. This algorithm acts as a meaningful component in enabling accurate ranking of web search results by imposing an order on web pages according to their importance. The idea behind PageRank is basically an application of the academic citation literature to the web. This involves deriving a Markov chain matrix from the hyperlink structure of the web and computing its principle eigenvector in a series of iterations.

Although the PageRank algorithm is quite effective, it may be computationally expensive due to the following three reasons: First, the size of the web is enormous. As of July 2008, it is estimated that the web contains 1 trillion unique pages [28]. Even with the fastest computers, PageRank computations using a web matrix of this size would take unacceptably long. Second, the web is constantly evolving [21]. New pages are added, existing pages are deleted, and links within the pages are modified constantly. This essentially requires recomputation of PageRank values in a continuous manner, or otherwise, computed page importance values quickly become obsolete. Third, in some cases, it may be necessary to compute more than one PageRank vector, e.g., if there are multiple preferred views for page importance [31]. These reasons show the need for efficient PageRank computations.

Broadly, the research efforts trying to speedup the PageRank computations are based on numerical techniques or parallelization. Among numerical approaches, there are various acceleration techniques such as extrapolation [12], [38], adaptive [39], [52], block-structure [40], and aggregation-disaggregation [46], [52]. These techniques mainly aim to increase the convergence rate of the power method [30], which is the de-facto method for PageRank computation with low memory requirement. Among the recently proposed linear system approaches, there are Krylov subspace methods [22], [25], which are applied together with various preconditioners. These methods decrease the number of iterations for convergence at the expense of increased computation per iteration and increased space consumption. At the core of all these iterative PageRank algorithms, there are repeated sparse matrix-vector multiplication (SpM$\times$V) operations. Also, recently, the lumpability of the dangling pages (i.e., pages with no outlinks) has been noticed and exploited to devise several algorithms that significantly reduce the per-iteration computation time [23], [36], [45], [48]. These algorithms achieve computational savings by excluding the SpMxVs associated with the dangling pages from the iterative computations without degrading the convergence rate. Hence, the total running time becomes proportional to the number of nondangling pages.

Despite the recent and rich literature on numeric approaches, research on parallelization of PageRank is relatively rare [25], [41], [42], [50]. A parallelization of the power method is discussed in [50], and various linear system formulations are compared in terms of parallel runtime performance in [25]. Asynchronous computation models for parallel PageRank are proposed in [42] and [43]. In [42], a multithreading scheme is investigated. In [43], asynchronous schemes are investigated for parallel architectures but very low speed-up values are reported even for small numbers of

- *A. Cevahir is with the Tokyo Institute of Technology, Tokyo, Japan. E-mail: ali@matsulab.is.titech.ac.jp.*
- *C. Aykanat and A. Turk are with the Computer Engineering Department, Bilkent University, Ankara 06800, Turkey. E-mail: {aykanat, atat}@cs.bilkent.edu.tr.*
- *B.B. Cambazoglu is with Yahoo! Research, Barcelona, Spain. E-mail: barla@yahoo-inc.com.*

Manuscript received 27 Jan. 2009; revised 17 Nov. 2009; accepted 18 Mar. 2010; published online 1 June 2010.
Recommended for acceptance by M. Yamashita.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-01-0039.
Digital Object Identifier no. 10.1109/TPDS.2010.119.

processors. A Gauss-Jacobi-based parallel PageRank algorithm, which utilizes the site information for straightforward matrix partitioning, is proposed in [41]. All these studies are based on 1D rowwise partitioning of the web matrix and consider only the load balancing issue in parallel SpMxVs. None of these studies apart from [41] involve any effort for minimization of the communication overhead, whereas the use of site information in [41] provides an implicit effort in this direction. However, if the number of sites is much larger than the number of processors, the benefit of this implicit effort diminishes. Finally, there are some works [55] on approximate Page-Rank computations in a distributed setting, but they are not in the scope of our work.

Bradley et al. [11] have applied hypergraph-partitioning-based (HP-based) parallel SpMxV models of Catalyurek and Aykanat [15], [16] and Ucar and Aykanat [58], for parallelization of PageRank computations. These models are based on 1D rowwise and 2D fine-grain partitioning of the web matrix and are quite successful in formulating the load balancing constraint and the total communication volume requirement during the repeated parallel SpMxVs involved in PageRank computations. However, given the vast size of the web matrix, these techniques are not affordable in practice due to the high partitioning overhead introduced by HP. Bradley et al. [11] try to tackle this performance issue using the parallel HP tool Par$k$way [56].

In this paper, we first focus on reducing the above-mentioned partitioning overhead. For this purpose, we propose two different web matrix compression schemes, namely, 1D and 2D compression, by exploiting the site information inherently available in page links. The 1D scheme compresses the $n \times n$ web matrix along only one dimension, i.e., either along rows or columns, thus obtaining an $m \times n$ or $n \times m$ matrix, where n is the number of pages and m is the number of sites. These matrices are then partitioned using HP. 1D rowwise and 1D columnwise partitioning models are discussed under this 1D compression scheme. The 1D rowwise partitioning model has been briefly introduced in our earlier works [2], [20]. The 2D scheme compresses the matrix in both dimensions, obtaining an $m \times m$ matrix, which is then partitioned using graph partitioning (GP). 1D rowwise and 1D columnwise partitioning models are formulated and discussed under this 2D compression scheme. These partitioning models significantly decrease the preprocessing overhead of partitioning the $n \times n$ matrix, without sacrificing the parallel efficiency.

Partitioning models discussed in the literature generally assume the availability of a global web graph, possibly stored as a single file or data set in a host machine. However, in a real-world scenario, this assumption may not be valid since the initial web data set is likely to be distributed among many processors. In such a setup, the data have to be redistributed among processors for the sake of efficient parallel PageRank computations. Hence, partitioning models should encapsulate the initial data redistribution overhead as well as the communication overhead that will be incurred during the parallel PageRank computations. This problem constitutes a typical instance of the repartitioning (remapping) problem. In this paper, we adopt the recently proposed repartitioning models [3], [14], [18], which are based on GP and HP with fixed vertices, and apply them on top of our above-mentioned site-based GP and HP models in order to encapsulate the initial redistribution overhead in parallel PageRank computations.

Moreover, in this paper, we propose a simple yet effective method to handle pages with no in-links. This method avoids the SpMxVs associated with the submatrices corresponding to the pages with no in-links throughout the iterations by only performing two SpMxVs at the beginning. Also, for the power-method-based parallel PageRank algorithms, we implement an improvement, which reduces the number of global communications due to the norm operations from two to one [20]. All of our contributions are presented in the context of a state-of-the-art sequential PageRank algorithm proposed by Ipsen and Selee [36], whereas our contributions can be easily extended to other iterative PageRank algorithms. This power-method-based algorithm [36] utilizes the lumping method to handle the dangling pages efficiently via applying the power method only to the smaller lumped matrix, where the convergence rate remains the same as that of the power method applied to the full matrix. It also has the advantage of allowing the dangling node vectors and personalization vectors to be different, thus enabling the implementation of TrustRank [29]. This algorithm is parallelized and tested on two PC clusters with 40 and 64 processors in order to verify the validity of the proposed techniques. PageRank computations conducted on eight well-known large web data sets indicate the effectiveness of the proposed techniques. These techniques result in considerably high speedups while incurring a preprocessing overhead of several iterations (for some instances even less than a single iteration) of the underlying sequential PageRank algorithm.

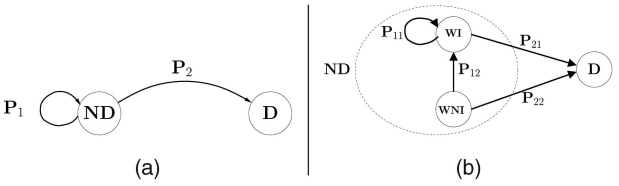


Fig. 1. (a) Decomposition of the web link structure according to dangling (D) and nondangling (ND) pages [36]. (b) Extended decomposition according to pages with in-links (WI) and with no in-links (WNI).

The organization of the paper is as follows: Section 2 provides background material. The proposed parallel PageRank algorithm is given in Section 3. The proposed compression schemes and partitioning models are given in Section 4. Section 5 presents the proposed repartitioning models. Experimental results are reported and discussed in Section 6. Section 7 concludes the paper.

2 Background

2.1 PageRank Algorithm

PageRank can be explained with a probabilistic model, called the random surfer model [54]. In this model, the PageRank of page i is defined as the steady-state probability that the surfer is at page i at some particular time step. In the Markov chain induced by a random walk on the web (containing n pages), the states of the chain 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 outlink(s) to page j, or 0, otherwise. Here, $deg(i)$ denotes the number of outlinks of page i.

In the web, there exist pages with no outlinks to other pages. Such pages are called dangling pages. We can decompose the link structure of the web, as shown in Fig. 1a, where ND and D, respectively, represent sets of n_1 nonangling and n_2 dangling pages, and $n_1 + n_2 = n$. In accordance with the link structure given in this figure, we

decompose the $\mathbf{P}$ matrix by permuting the rows and columns corresponding to the dangling pages to the end as:

$$\widehat{\mathbf{P}} = \mathbf{Q}\mathbf{P}\mathbf{Q}^{\mathrm{T}} = \begin{array}{|c|c|}\hline \mathbf{P}_1 & \mathbf{P}_2 \\ \hline \multicolumn{2}{|c|}{\mathbf{Z}} \\ \hline \end{array} \quad ,$$

where $\mathbf{Q}$ is the $n \times n$ permutation matrix. Here, $\mathbf{P}_1$ is an $n_1 \times n_1$ matrix representing the links among nondangling pages, $\mathbf{P}_2$ is an $n_1 \times n_2$ matrix representing the outlinks from nondangling to dangling pages, and $\mathbf{Z}$ is an $n_2 \times n$ zero-matrix.

A row-stochastic transition matrix $\mathbf{S}$ is constructed from $\widehat{\mathbf{P}}$ as:

$$\mathbf{S} = \widehat{\mathbf{P}} + \mathbf{d}\mathbf{u}^{\mathrm{T}} = \begin{array}{|c|c|}\hline \mathbf{P}_1 & \mathbf{P}_2 \\ \hline \multicolumn{2}{|c|}{\mathbf{d}\mathbf{u}^{\mathrm{T}}} \\ \hline \end{array} \text{, where } \mathbf{d} = \begin{array}{|c|}\hline \mathbf{0} \\ \hline \mathbf{e}_{n_2} \\ \hline \end{array}$$

via handling of dangling pages according to the random surfer model. That is, a surfer visiting a dangling page randomly jumps to another page in the next time step according to the distribution given by the dangling page vector $\mathbf{u}$, where $\|\mathbf{u}\|_1 = 1$. Here, $\mathbf{e}_{n_2}$ denotes a column vector of size n_2 containing all ones, and $\|\cdot\|_1$ denotes the L_1-norm. Although $\mathbf{S}$ is row stochastic, it may not be irreducible. An irreducible Markov matrix $\mathbf{G}$, which is also known as the Google matrix [48], is constructed as:

$$\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)\mathbf{e}_n\mathbf{t}^{\mathrm{T}}. \tag{1}$$

Here, α represents the probability that the surfer chooses to follow one of the outlinks of the current page and $(1-\alpha)$ represents the probability that the surfer makes a random jump instead of following the outlinks. $\mathbf{t}$ is the teleportation (personalization) vector, which denotes the probability distribution of destination pages for a random jump. A uniform teleportation vector $\mathbf{t}$, where $\mathbf{t}_i = 1/n$ for all i, is used for general PageRank computation. Nonuniform teleportation vectors can be used for topical or personalized PageRank computation [31], [54].

Given $\mathbf{G}$, PageRank vector $\mathbf{p}$ can be determined by computing the stationary distribution for the Markov chain that satisfies $\mathbf{G}^{\mathrm{T}} \times \mathbf{p} = \mathbf{p}$. This corresponds to finding the principal eigenvector of matrix $\mathbf{G}$. Applying the power method directly for the solution of this eigenvector problem leads to a sequence of SpMxVs $\mathbf{p}^{i+1} = \mathbf{G}^{\mathrm{T}} \times \mathbf{p}^i$, where $\mathbf{p}^i$ is the ith iterate towards the PageRank vector $\mathbf{p}$. However, unlike $\mathbf{P}$, the $\mathbf{G}$ matrix is completely dense. The power method applied to $\mathbf{G}$ can be implemented with SpMxVs on the original sparse $\widehat{\mathbf{P}}^{\mathrm{T}}$ matrix without forming dense $\mathbf{S}$ and $\mathbf{G}$ matrices by the following iterative formula [38], [45]:

$$\mathbf{p}^{i+1} = \mathbf{G}^{\mathrm{T}} \times \mathbf{p}^i = \alpha\mathbf{S}^{\mathrm{T}} \times \mathbf{p}^i + (1-\alpha)\mathbf{t}\big(\mathbf{e}_n{}^{\mathrm{T}}\mathbf{p}^i\big) \tag{2}$$

$$= \alpha\widehat{\mathbf{P}}^{\mathrm{T}} \times \mathbf{p}^i + \alpha\mathbf{u}(\mathbf{d}^{\mathrm{T}}\mathbf{p}^i) + (1-\alpha)\mathbf{t} \tag{3}$$

$$= \alpha\widehat{\mathbf{P}}^{\mathrm{T}} \times \mathbf{p}^i + \alpha\big(1 - \|\mathbf{p}_1^i\|_1\big)\mathbf{u} + (1-\alpha)\mathbf{t}. \tag{4}$$

In (2), $\mathbf{e}_n^{\mathrm{T}}\mathbf{p}^i = 1$ since $\mathbf{p}^i$ is a probability vector. In (3), $\mathbf{d}^{\mathrm{T}}\mathbf{p}^i = [\mathbf{0} \quad \mathbf{e}_{\mathbf{n_2}}^{\mathrm{T}}]\,[\mathbf{p_1^i}^{\mathrm{T}} \quad \mathbf{p_2^i}^{\mathrm{T}}]^{\mathrm{T}} = \|\mathbf{p_2^i}\|_1 = \mathbf{1} - \|\mathbf{p_1^i}\|_1$, where $\mathbf{p}_1^i$ and $\mathbf{p}_2^i$ are the ith iterates of the PageRank vectors corresponding to nondangling and dangling pages, respectively.

Herein, we choose to parallelize the PageRank algorithm given in [36], which handles dangling pages via lumping method. In the rest of the paper, we will use $\mathbf{A} = \widehat{\mathbf{P}}^{\mathrm{T}}$ since our discussions about parallelization are based on matrix-vector multiplication rather than vector-matrix multiplication. That is:

$$\mathbf{A} = \widehat{\mathbf{P}}^{\mathrm{T}} = \begin{array}{|c|c|}\hline \mathbf{P}_1^{\mathrm{T}} & \\ \hline \mathbf{P}_2^{\mathrm{T}} & \mathbf{Z}^{\mathrm{T}} \\ \hline \end{array} = \begin{array}{|c|c|}\hline \mathbf{A}_1 & \\ \hline \mathbf{A}_2 & \mathbf{Z}^{\mathrm{T}} \\ \hline \end{array} \quad ,$$

where $\mathbf{A}_1 = \mathbf{P}_1{}^{\mathrm{T}}$ and $\mathbf{A}_2 = \mathbf{P}_2{}^{\mathrm{T}}$ are submatrices of sizes $n_1 \times n_1$ and $n_2 \times n_1$, respectively. The lumping method avoids the $\mathbf{A}_2 \times \mathbf{p}_1$ SpMxV associated with the dangling pages throughout the power method iterations. After the convergence of the power method iterations, the PageRank vector for the dangling pages is computed by a single $\mathbf{A}_2 \times \mathbf{p}_1$ SpMxV.

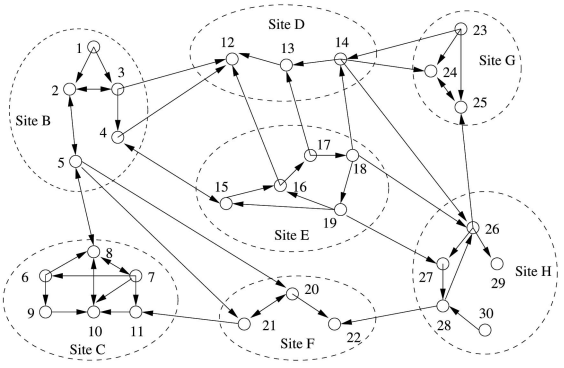


Fig. 2. A sample subset of the web with seven sites, 30 pages, and 56 links.

Fig. 2 displays a sample subset of the web, which contains three dangling pages (i.e., pages 12, 22, and 29). Fig. 3 displays sparsity patterns of $\mathbf{P}^{\mathrm{T}}$, $\mathbf{A} = \widehat{\mathbf{P}}^{\mathrm{T}}$ and $\mathbf{A}_1 = \mathbf{P}_1^{\mathrm{T}}$ matrices, nonzeros of which represent the link structure of the sample web. In Figs. 3a and 3b, gray columns, which contain no nonzeros, correspond to dangling pages. In Fig. 3b, solid horizontal and vertical lines show the decomposition of the $\mathbf{A}$ matrix into $\mathbf{A}_1$ and $\mathbf{A}_2$ submatrices. Finally, Fig. 3c shows the $\mathbf{A}_1$ matrix.

2.2 Sparse Matrix Partitioning Models

Partitioning of irregularly sparse matrices for parallelization of SpMxVs is formulated as K-way GP [33] and K-way HP [15], for a K-processor parallel system. In these models, the partitioning objective of minimizing the cutsize, which is defined over the edges or nets, relates to minimizing the total communication volume. The partitioning constraint of maintaining the balance on part weights corresponds to maintaining the computational load balance. HP models have the following advantages over GP models: First, the partitioning objective in HP models is an exact measure of the total communication volume, whereas the objective in GP models is an approximation. Second, HP models are capable of partitioning rectangular matrices, whereas GP models can only partition square matrices. Third, elegant HP models exist for 2D matrix partitioning [16].

In the GP models for 1D rowwise and columnwise partitioning, a square matrix is represented as a graph, which contains a vertex for each row/column and an edge for each nonzero. Weights of vertices are set equal to the number of nonzeros in the respective rows or columns for rowwise or columnwise partitioning, respectively. The cost of each edge is set equal to 2 for structurally symmetric

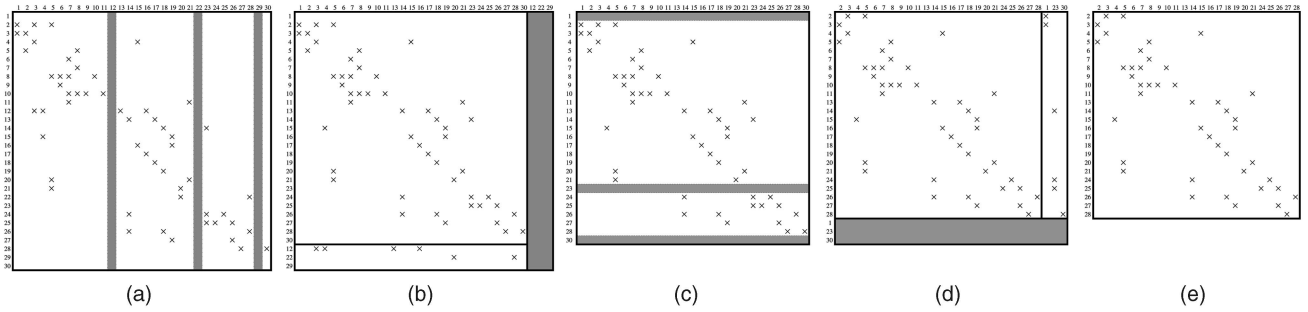


Fig. 3. Sparsity patterns of matrices (a) $\mathbf{P}^{\mathrm{T}}$, (b) $\mathbf{A} = \widehat{\mathbf{P}}^{\mathrm{T}}$, (c) $\mathbf{A}_1 = \mathbf{P}_1{}^{\mathrm{T}}$, (d) $\mathbf{A}'_1$, (e) $\mathbf{A}_{11}$.

matrices, whereas it is set to either 1 or 2 for structurally unsymmetric matrices [15].

In the column-net HP model for 1D rowwise partitioning [15], a given matrix is represented as a hypergraph, which contains a vertex for each row and a net for each column. Each net corresponds to a column and connects vertices that correspond to the rows with at least one nonzero in that column. In the row-net HP model for 1D columnwise partitioning [15], there exists a vertex for each column and a net for each row such that the net corresponding to a row connects the vertices corresponding to the columns that have a nonzero at that row. The vertex weighting schemes for rowwise and columnwise HP models are the same as the respective GP models. All nets are associated with a unit cost in both row-net and column-net models.

To enforce symmetric partitioning in the 1D framework, both input and output vectors of the SpMxV are partitioned conformally with the given rowwise or columnwise partition of the matrix. Consistency of HP models for symmetric partitioning depends on the existence of nonzero diagonals in the matrix [15], [16]. If the matrix contains zero diagonals, virtual nonzeros are inserted into the diagonal of the matrix, and then the respective HP model is applied. Although these virtual nonzeros do not have weights, they affect the topology of the constructed hypergraphs.

2.3 Repartitioning Models

In many scientific computing applications, although the initial mapping of tasks to processors may be satisfactory in terms of both computational load balance and communication overheads, the quality of this initial mapping typically tends to deteriorate in successive phases as the computational structure or the application parameters change. This has the potential to reduce the efficiency of parallelization. One solution is to rebalance the load distribution of the processors as needed by rearranging the assignment of tasks to processors via repartitioning.

Recently, a number of successful models [4], [14], [18], based on GP and HP with fixed vertices, are proposed as solutions to repartitioning problems in different applications. In these models, tasks and interactions among them are represented as an interaction graph/hypergraph, where vertices model tasks and the associated data, and edges/nets model interactions. This interaction graph/hypergraph is augmented by fixed processor vertices and edges/nets, which connect original vertices with appropriate processor vertices in order to represent the initial task and data distribution. Then, the repartitioning problem is formulated as K-way GP/HP with fixed vertices on this augmented graph/hypergraph, referred to as the repartitioning graph/hypergraph [4], [18]. In this repartitioning model, the cutsize defined over the original edges/nets shows the total communication volume due to assigning interacting tasks to different processors, whereas the cutsize defined over the newly added edges/nets shows the communication overhead due to data redistribution.

3 Parallel PageRank Algorithm

In addition to dangling pages, the web may also contain many pages with no in-links [6]. Based on this fact, the web link structure given in Fig. 1a can be further refined, as shown in Fig. 1b. In Fig. 1b, $\mathbf{P}_{11}$ represents the links among nondangling pages with in-links, $\mathbf{P}_{21}$ represents the links from nondangling pages with in-links to dangling pages, $\mathbf{P}_{12}$ represents the links from nondangling pages with no in-links to nondangling pages with in-links, and $\mathbf{P}_{22}$ represents the links from nondangling pages with no in-links to dangling pages. The extended decomposition given in Fig. 1b can be considered as a special case of the strongly connected component decomposition approach proposed in [49]. However, identifying pages with no in-links is very easy and their PageRank values can be computed very efficiently as described below.

In accordance with the link structure given in Fig. 1b, we can decompose $\mathbf{P}_1$ by permuting its rows and columns corresponding to the pages with no in-links to the end, and we can decompose $\mathbf{P}_2$ by permuting its rows corresponding to the pages with no in-links to the end as follows:

$$\mathbf{P}'_1 = \mathbf{O}\mathbf{P}_1\mathbf{O}^{\mathrm{T}} = \left[\begin{array}{c|c} \mathbf{P}_{11} & \\ \hline \mathbf{P}_{12} & \mathbf{Z} \end{array}\right], \quad \mathbf{P}'_2 = \mathbf{O}\mathbf{P}_2 = \left[\begin{array}{c} \mathbf{P}_{21} \\ \hline \mathbf{P}_{22} \end{array}\right]$$

where $\mathbf{O}$ is the $n_1 \times n_1$ permutation matrix, $\mathbf{P}'_1$ and $\mathbf{P}'_2$ are the permuted versions of $\mathbf{P}_1$ and $\mathbf{P}_2$, respectively. Here, columns of zero submatrix $\mathbf{Z}$ and rows of submatrix $\mathbf{P}_{12}$ correspond to pages with no in-links. Since $\mathbf{A}_1 = (\mathbf{P}_1)^{\mathrm{T}}$ and $\mathbf{A}_2 = (\mathbf{P}_2)^{\mathrm{T}}$, $\mathbf{A}_1$ and $\mathbf{A}_2$ can be decomposed as:

$$\mathbf{A}'_1 = \left[\begin{array}{c|c} \mathbf{P}_{11}^{\mathrm{T}} & \mathbf{P}_{12}^{\mathrm{T}} \\ \hline \multicolumn{2}{c}{\mathbf{Z}^{\mathrm{T}}} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \hline \multicolumn{2}{c}{\mathbf{Z}^{\mathrm{T}}} \end{array}\right], \quad \mathbf{A}'_2 = \left[\begin{array}{c|c} \mathbf{P}_{21}^{\mathrm{T}} & \mathbf{P}_{22}^{\mathrm{T}} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{A}_{21} & \mathbf{A}_{22} \end{array}\right]$$

leading to the following decomposition on $\mathbf{A}$:

$$\mathbf{A}' = (\mathbf{P}')^{\mathrm{T}} = \left[\begin{array}{c|c|c} \mathbf{P}_{11}^{\mathrm{T}} & \mathbf{P}_{12}^{\mathrm{T}} & \\ \hline \multicolumn{2}{c|}{\mathbf{Z}} & \mathbf{Z} \\ \hline \mathbf{P}_{21}^{\mathrm{T}} & \mathbf{P}_{22}^{\mathrm{T}} & \end{array}\right] = \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \\ \hline \multicolumn{2}{c|}{\mathbf{Z}} & \mathbf{Z} \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22} & \end{array}\right]$$

Here, $\mathbf{A}_{11}$ and $\mathbf{A}_{12}$ are submatrices of sizes $n_{11}\times n_{11}$ and $n_{11}\times n_{12}$, respectively, and $\mathbf{A}_{21}$ and $\mathbf{A}_{22}$ are submatrices of sizes $n_2\times n_{11}$ and $n_2\times n_{12}$, respectively. n_{12} denotes the number of nondangling pages with no in-links and $n_1 = n_{11} + n_{12}$. Reordering pages with no in-links to the end may cause new zero rows to appear. Hence, it is possible to investigate $\mathbf{A}_{11}$ in a recursive manner for further reductions. This recursive scheme is not investigated in this paper.

For the $\mathbf{q}_1 = \mathbf{A}_1 \times \mathbf{p}_1$ multiplication, entries of the $\mathbf{p}_1$ and $\mathbf{q}_1$ vectors are permuted according to this row/column reordering:

$$\mathbf{q}'_1 = \mathbf{O} \times \mathbf{q}_1 = \begin{bmatrix} \mathbf{q}_1(\mathbf{A}_{11}) \\ \mathbf{q}_1(\mathbf{Z}) \end{bmatrix}, \quad \mathbf{p}'_1 = \mathbf{O} \times \mathbf{p}_1 = \begin{bmatrix} \mathbf{p}_1(\mathbf{A}_{11}) \\ \mathbf{p}_1(\mathbf{Z}) \end{bmatrix}. \tag{5}$$

In further discussions, for readability, $\mathbf{q}'_1$, $\mathbf{p}'_1$, $\mathbf{A}'_1$, and $\mathbf{A}'_2$ will be referred to as $\mathbf{q}_1$, $\mathbf{p}_1$, $\mathbf{A}_1$, and $\mathbf{A}_2$, respectively. In (6), $\mathbf{q}_1(\mathbf{A}_{11})$ and $\mathbf{p}_1(\mathbf{A}_{11})$ are column vectors of size n_{11}, and $\mathbf{q}_1(\mathbf{Z})$ and $\mathbf{p}_1(\mathbf{Z})$ are column vectors of size n_{12}. As pages with no in-links correspond to zero rows of $\mathbf{A}_1$, the $\mathbf{q}_1(\mathbf{Z}) = \mathbf{Z}\times\mathbf{p}_1$ multiplication results in a zero vector. Hence, the $\mathbf{q}_1 = \mathbf{A}_1\times\mathbf{p}_1$ multiplication can be performed as the sum of the results of two SpMxVs:

$$\mathbf{q}_1(\mathbf{A}_1) = \mathbf{A}_{11} \times \mathbf{p}_1(\mathbf{A}_{11}) + \mathbf{A}_{12} \times \mathbf{p}_1(\mathbf{Z}).$$

Since $\mathbf{q}_1(\mathbf{Z}) = 0$, we have $\mathbf{p}_1(\mathbf{Z}) = (1-\alpha)\mathbf{t}_1(\mathbf{Z}) + \alpha\gamma\mathbf{u}_1(\mathbf{Z})$. Hence, the $\mathbf{A}_{12}\times\mathbf{p}_1(\mathbf{Z})$ multiplication reduces to:

$$\mathbf{A}_{12} \times \mathbf{p}_1(\mathbf{Z}) = 0 + (1-\alpha)\mathbf{A}_{12} \times \mathbf{t}_1(\mathbf{Z}) + \alpha\gamma\mathbf{A}_{12} \times \mathbf{u}_1(\mathbf{Z}).$$

Note that $\mathbf{t}_1(\mathbf{Z})$ and $\mathbf{u}_1(\mathbf{Z})$ do not change throughout the iterations. Thus, SpMxVs $\mathbf{A}_{12} \times \mathbf{t}_1(\mathbf{Z})$ and $\mathbf{A}_{12} \times \mathbf{u}_1(\mathbf{Z})$ can be avoided by computing the SpMxVs $\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z}) = \mathbf{A}_{12} \times \mathbf{t}_1(\mathbf{Z})$ and $\widehat{\mathbf{u}}_1(\mathbf{A}_{12}, \mathbf{Z}) = \mathbf{A}_{12} \times \mathbf{u}_1(\mathbf{Z})$ only once at the very beginning and computing $\mathbf{q}_1(\mathbf{A}_{11})$ as:

$$\begin{aligned}\mathbf{q}_1(\mathbf{A}_{11}) = \mathbf{A}_{11} \times \mathbf{p}_1(\mathbf{A}_{11}) + (1-\alpha)\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z}) \\ + \alpha\gamma\widehat{\mathbf{u}}_1(\mathbf{A}_{12}, \mathbf{Z})\end{aligned}$$

at every iteration. That is, SpMxV $\mathbf{A}_{12} \times \mathbf{p}_1(\mathbf{Z})$ is replaced by the less expensive DAXPY operation $(1-\alpha)\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z}) + \alpha\gamma\widehat{\mathbf{u}}_1(\mathbf{A}_{12}, \mathbf{Z})$. Since scalar $(1-\alpha)$ and vector $\widehat{\mathbf{t}}_1, (\mathbf{A}_{12}, \mathbf{Z})$ are constant, the scalar-vector multiplication $(1-\alpha)\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z})$ can also be avoided by computing $\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z}) = (1-\alpha)\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z})$ only once at the very beginning. The scalar-vector multiplies $(1-\alpha)\mathbf{t}_1(\mathbf{A}_{11})$ and $(1-\alpha)\mathbf{t}_1(\mathbf{Z})$ can also be avoided, in a similar manner. Fig. 4 displays the PageRank algorithm for efficient handling of pages with no in-links.

Three basic types of operations are performed repeatedly at each iteration: 1) SpMxV performed at step 7. 2) Linear vector operations performed on the input and output vectors $\mathbf{p}_1$ and $\mathbf{q}_1$ of the SpMxV and the teleportation and dangling page vectors $\mathbf{t}_1$ and $\mathbf{u}_1$. These operations include the DAXPY-like operations at steps 8 and 9, and the vector subtraction at step 10. 3) L_1-norm operations performed at steps 6 and 10.

As the input vector of the current iteration is obtained from the output vector of the previous iteration through linear vector operations at steps 8 and 9(a), a symmetric partitioning scheme is adopted to avoid communication of vector entries during the linear vector operations. Hence, all vectors that participate in steps 7, 8, and 9(a) (i.e., $\mathbf{p}_1(\mathbf{A}_{11})$, $\mathbf{q}_1(\mathbf{A}_{11})$, $\mathbf{t}_1(\mathbf{A}_{11})$, $\mathbf{u}_1(\mathbf{A}_{11})$, $\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z})$, $\widehat{\mathbf{u}}_1(\mathbf{A}_{12}, \mathbf{Z})$) are partitioned

PageRank($\mathbf{A}_{11}$, $\mathbf{A}_{12}$, $\mathbf{A}_2$, $\mathbf{t}$, $\mathbf{u}$, α, ε)
▷ $\mathbf{p} = [\mathbf{p}_1^{\mathrm{T}}\ \mathbf{p}_2^{\mathrm{T}}]^{\mathrm{T}}$; $\mathbf{t} = [\mathbf{t}_1^{\mathrm{T}}\ \mathbf{t}_2^{\mathrm{T}}]^{\mathrm{T}}$; $\mathbf{u} = [\mathbf{u}_1^{\mathrm{T}}\ \mathbf{u}_2^{\mathrm{T}}]^{\mathrm{T}}$
▷ $\mathbf{p}_1 = [\mathbf{p}_1^{\mathrm{T}}(\mathbf{A}_{11})\ \mathbf{p}_1^{\mathrm{T}}(Z)]^{\mathrm{T}}$; $\mathbf{t}_1 = [\mathbf{t}_1^{\mathrm{T}}(\mathbf{A}_{11})\ \mathbf{t}_1^{\mathrm{T}}(Z)]^{\mathrm{T}}$
▷ $\mathbf{u}_1 = [\mathbf{u}_1^{\mathrm{T}}(\mathbf{A}_{11})\ \mathbf{u}_1^{\mathrm{T}}(Z)]^{\mathrm{T}}$
1. (a) $\mathbf{p}_1 \leftarrow \mathbf{t}_1$
   (b) $\mathbf{p}_1^{\mathrm{old}} \leftarrow \mathbf{p}_1$
2. $\delta \leftarrow \|\mathbf{p}_1\|_1$
3. (a) $\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow \mathbf{A}_{12} \times \mathbf{t}_1(\mathbf{Z})$
   (b) $\widehat{\mathbf{u}}_1(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow \mathbf{A}_{12} \times \mathbf{u}_1(\mathbf{Z})$
4. (a) $\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow (1-\alpha)\widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z})$
   (b) $\mathbf{t}_1 \leftarrow (1-\alpha)\mathbf{t}_1$
5. **while** $\delta > \varepsilon$ **do**
6. $\gamma \leftarrow 1 - \|\mathbf{p}_1\|_1$
7. $\mathbf{q}_1(\mathbf{A}_{11}) \leftarrow \mathbf{A}_{11} \times \mathbf{p}_1(\mathbf{A}_{11})$
8. $\mathbf{q}_1(\mathbf{A}_{11}) \leftarrow \mathbf{q}_1(\mathbf{A}_{11}) + \widehat{\mathbf{t}}_1(\mathbf{A}_{12}, \mathbf{Z}) + \alpha\gamma\widehat{\mathbf{u}}_1(\mathbf{A}_{12}, \mathbf{Z})$
9. (a) $\mathbf{p}_1(\mathbf{A}_{11}) \leftarrow \alpha\mathbf{q}_1(\mathbf{A}_{11}) + \mathbf{t}_1(\mathbf{A}_{11}) + \alpha\gamma\mathbf{u}_1(\mathbf{A}_{11})$
   (b) $\mathbf{p}_1(\mathbf{Z}) \leftarrow \mathbf{t}_1(\mathbf{Z}) + \alpha\gamma\mathbf{u}_1(\mathbf{Z})$
10. $\delta \leftarrow \|\mathbf{p}_1 - \mathbf{p}_1^{\mathrm{old}}\|_1$
11. $\mathbf{p}_1^{\mathrm{old}} \leftarrow \mathbf{p}_1$
12. **end while**
13. $\gamma \leftarrow 1 - \|\mathbf{p}_1\|_1$
14. (a) $\mathbf{q}_2 \leftarrow \mathbf{A}_2 \times \mathbf{p}_1$
    (b) $\mathbf{p}_2 \leftarrow \alpha\mathbf{q}_2 + (1-\alpha)\mathbf{t}_2 + \alpha\gamma\mathbf{u}_2$
15. **return** $\mathbf{p}$

Fig. 4. PageRank algorithm, which handles dangling pages via the lumping method [36] and pages with no in-links.

conformally with the partition induced by partitioning of $\mathbf{A}_{11}$. In particular, $\mathbf{p}_1(\mathbf{A}_{11})$ and $\mathbf{q}_1(\mathbf{A}_{11})$ vectors are partitioned as $[\mathbf{p}_{11}^{\mathrm{T}}(\mathbf{A}_{11}) \cdots \mathbf{p}_{1K}^{\mathrm{T}}(\mathbf{A}_{11})]^{\mathrm{T}}$ and $[\mathbf{q}_{11}^{\mathrm{T}}(\mathbf{A}_{11}) \cdots \mathbf{q}_{1K}^{\mathrm{T}}(\mathbf{A}_{11})]^{\mathrm{T}}$, respectively, where processor P_k is also responsible for the linear vector operations on the kth blocks of the vectors. That is, P_k performs linear vector operations on $\mathbf{p}_{1k}(\mathbf{A}_{11})$, $\mathbf{q}_{1k}(\mathbf{A}_{11})$, $\mathbf{t}_{1k}(\mathbf{A}_{11})$, $\mathbf{u}_{1k}(\mathbf{A}_{11})$, $\widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z})$, $\widehat{\mathbf{u}}_{1k}(\mathbf{A}_{12}, \mathbf{Z})$. The Z-vectors $\mathbf{p}_1(\mathbf{Z})$, $\mathbf{t}_1(\mathbf{Z})$, $\mathbf{u}_1(\mathbf{Z})$ involved in the linear vector operation at step 9(b) do not participate in linear vector operations with other vectors. Hence, they can be partitioned independent of partitioning of the $\mathbf{A}_{11}$ matrix. It is sufficient to partition these vectors conformally with each other. Parallelization of the L_1-norm operations at steps 6 and 10, which compute the global scalars γ and δ, requires global communication operations in the form of all-to-all reduction. Here, we adopt our efficient parallelization scheme [2], [20] to reduce the number of global communication operations at each iteration from 2 to 1 by rearranging the computations. Note that the use of *compensated summation* scheme [61] can be considered for these norm operations to achieve better accuracy in large data sets. However, we do not use compensated summation, since we believe it will not substantially change our discussions.

Fig. 5 displays the proposed parallel PageRank algorithm. In Fig. 5, the superscript k of a matrix (e.g., $\mathbf{A}_{11}^k$) denotes the portion of that matrix stored in processor P_k. The superscript k of a global scalar denotes the partial result computed by P_k, e.g., δ^k is the partial result for the global scalar δ, where $\delta = \sum_{k=1}^{K} \delta^k$. Note that two global norms (π and δ) are accumulated at all processors by the single all-to-all reduction performed at step 9(c). The bottom part of Fig. 5 displays the row and column-parallel implementations [57] of the ParMatVecMult function. The row-parallel and column-parallel algorithms are applied to partitions obtained using the 1D rowwise and columnwise schemes, respectively. In the row-parallel algorithm, *Expand* represents the multicast-like

Parallel-PageRank($\mathbf{A}_{11}^k$, $\mathbf{A}_{12}^k$, ${\mathbf{A}_2}^k$, $\mathbf{t}_k$, $\mathbf{u}_k$, α, ε)
▷ $\mathbf{p}_k = [\mathbf{p}_{1k}^{\mathrm{T}} \ \mathbf{p}_{2k}^{\mathrm{T}}]^{\mathrm{T}}$; $\mathbf{t}_k = [\mathbf{t}_{1k}^{\mathrm{T}} \ \mathbf{t}_{2k}^{\mathrm{T}}]^{\mathrm{T}}$; $\mathbf{u}_k = [\mathbf{u}_{1k}^{\mathrm{T}} \ \mathbf{u}_{2k}^{\mathrm{T}}]^{\mathrm{T}}$
▷ $\mathbf{p}_{1k} = [\mathbf{p}_{1k}^{\mathrm{T}}(\mathbf{A}_{11}) \ \mathbf{p}_{1k}^{\mathrm{T}}(\mathbf{Z})]^{\mathrm{T}}$; $\mathbf{t}_{1k} = [\mathbf{t}_{1k}^{\mathrm{T}}(\mathbf{A}_{11}) \ \mathbf{t}_{1k}^{\mathrm{T}}(\mathbf{Z})]^{\mathrm{T}}$
▷ $\mathbf{u}_{1k} = [\mathbf{u}_{1k}^{\mathrm{T}}(\mathbf{A}_{11}) \ \mathbf{u}_{1k}^{\mathrm{T}}(\mathbf{Z})]^{\mathrm{T}}$
1. (a) $\mathbf{p}_{1k} \leftarrow \mathbf{t}_{1k}$
   (b) $\mathbf{p}_{1k}^{\mathrm{old}} \leftarrow \mathbf{p}_{1k}$
2. (a) $\pi^k \leftarrow \|\mathbf{p}_{1k}\|_1$
   (b) $\delta^k \leftarrow \|\mathbf{p}_{1k}\|_1$
   (c) $\langle \pi, \delta \rangle \leftarrow$ **AllReduceSum**$(\langle \pi^k, \delta^k \rangle)$
   (d) $\gamma \leftarrow 1 - \pi$
3. (a) $\widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow$ **ParMatVecMult**$(\mathbf{A}_{12}^k, \mathbf{t}_{1k}(\mathbf{Z}))$
   (b) $\widehat{\mathbf{u}}_{1k}(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow$ **ParMatVecMult**$(\mathbf{A}_{12}^k, \mathbf{u}_{1k}(\mathbf{Z}))$
4. (a) $\widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow (1-\alpha)\widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z})$
   (b) $\mathbf{t}_{1k} \leftarrow (1-\alpha)\mathbf{t}_{1k}$
5. **while** $\delta > \varepsilon$ **do**
6. $\mathbf{q}_{1k}(\mathbf{A}_{11}) \leftarrow$ **ParMatVecMult**$(\mathbf{A}_{11}^k, \mathbf{p}_{1k}(\mathbf{A}_{11}))$
7. $\mathbf{q}_{1k}(\mathbf{A}_{11}) \leftarrow \mathbf{q}_{1k}(\mathbf{A}_{11}) + \widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z}) + \alpha\gamma\widehat{\mathbf{u}}_{1k}(\mathbf{A}_{12}, \mathbf{Z})$
8. (a) $\mathbf{p}_{1k}(\mathbf{A}_{11}) \leftarrow \alpha\mathbf{q}_{1k}(\mathbf{A}_{11}) + \mathbf{t}_{1k}(\mathbf{A}_{11}) + \alpha\gamma\mathbf{u}_{1k}(\mathbf{A}_{11})$
   (b) $\mathbf{p}_{1k}(\mathbf{Z}) \leftarrow \mathbf{t}_{1k}(\mathbf{Z}) + \alpha\gamma\mathbf{u}_{1k}(\mathbf{Z})$
9. (a) $\pi^k \leftarrow \|\mathbf{p}_{1k}\|_1$
   (b) $\delta^k \leftarrow \|\mathbf{p}_{1k} - \mathbf{p}_{1k}^{\mathrm{old}}\|_1$
   (c) $\langle \pi, \delta \rangle \leftarrow$ **AllReduceSum**$(\langle \pi^k, \delta^k \rangle)$
   (d) $\gamma \leftarrow 1 - \pi$
10. $\mathbf{p}_{1k}^{\mathrm{old}} \leftarrow \mathbf{p}_{1k}$
11. **end while**
12. (a) $\mathbf{q}_{2k} \leftarrow$ **ParMatVecMult**$(\mathbf{A}_2^k, \mathbf{p}_{1k})$
   (b) $\mathbf{p}_{2k} \leftarrow \alpha\mathbf{q}_{2k} + (1-\alpha)\mathbf{t}_{2k} + \alpha\gamma\mathbf{u}_{2k}$
13. **return** $\mathbf{p}_k$

ParMatVecMult $(\mathbf{A}^k, \mathbf{x}_k)$

(a) $\mathbf{x}'_k \leftarrow \text{Expand}(\mathbf{x}_k)$	(a) $\mathbf{y}^k \leftarrow \mathbf{A}^k \times \mathbf{x}_k$
(b) $\mathbf{y}_k \leftarrow \mathbf{A}^k \times \mathbf{x}'_k$	(b) $\mathbf{y}_k \leftarrow \text{Fold}(\mathbf{y}^k)$
Row parallel	Column parallel

Fig. 5. Parallel PageRank algorithm (pseudocode for processor P_k).

operation performed by the processors for sending their local $\mathbf{x}$-vector entries to other processors according to sparsity patterns of respective columns of $\mathbf{A}$. $\mathbf{x}'_k$ denotes $\mathbf{x}$-vector entries that are needed by P_k for its local SpMxV. In the column-parallel algorithm, *Fold* corresponds to the multinode accumulation performed by the processors on local $\mathbf{y}$-vector entries according to sparsity patterns of respective rows of $\mathbf{A}$.

4 Partitioning Models for Parallel PageRank

We investigate two distinct frameworks for partitioning: page based and site based. In page-based partitioning, the page-by-page (PP) matrix $\mathbf{A}_{11}$ is partitioned directly without compression. This approach introduces unacceptable partitioning overhead due to size issues. However, we still present the page-based models for a better understanding of the site-based models. In site-based partitioning, the site information is utilized in order to reduce the partitioning time. For this purpose, we propose 1D page-by-site (PS), 1D site-by-page (SP), and 2D site-by-site (SS) compression schemes on $\mathbf{A}_{11}$. Fig. 6 displays the taxonomy of the partitioning models. Here, RW and CW denote 1D rowwise and 1D columnwise matrix partitioning schemes, respectively.

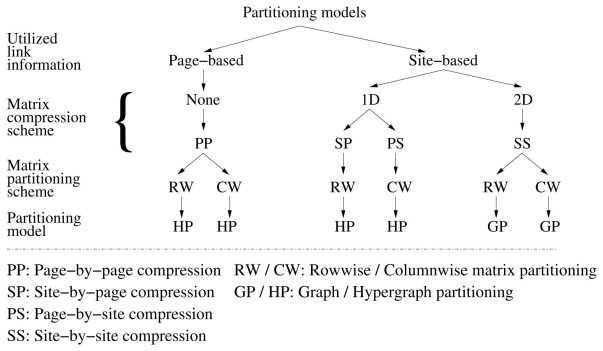


Fig. 6. A taxonomy for the partitioning models used.

4.1 Page-Based Partitioning Models

We obtain a K-way 1D rowwise and 1D columnwise partition of matrix $\mathbf{A}_{11}$ by partitioning the appropriate graph or hypergraph representation of $\mathbf{A}_{11}$. For computational load balancing, vertices of the graph/hypergraph are weighted to incorporate the floating point operations (flops) associated with the SpMxV $\mathbf{A}_{11} \times \mathbf{p}_1(\mathbf{A}_{11})$ as well as the flops associated with the linear vector and norm operations. The local linear vector operations performed at step 8(b) of Fig. 5 should be balanced separately since partitioning of the Z-vectors are independent of partitioning of $\mathbf{A}_{11}$. We achieve this balancing by adopting the best-fit decreasing heuristic used in solving the K-feasible bin-packing problem [34].

In RW, there is need to balance local computations between the local synchronization due to the point-to-point expand operation at step 6(a) and the global synchronization due to the all-reduce-sum operation at step 9(c). A single-constraint formulation becomes sufficient for load balancing as local SpMxV and linear vector operations remain as a single block between successive synchronization points throughout the iterations. Thus, the weight of vertex v_i is set equal to:

$$w(v_i) = 2 \times \text{nnz}\,(\text{row } i \text{ of } \mathbf{A}_{11}) + 10. \tag{6}$$

The first term accounts for the number of flops associated with row i during an SpMxV since each matrix nonzero incurs a scalar multiply-and-add operation. The second term accounts for the number of flops associated with the linear vector and norm operations performed on the ith entries of the vectors at steps 7, 8(a), 9(a), and 9(b).

In CW, the local SpMxV and the linear vector operations remain in separate blocks throughout the iterations. The SpMxV operations performed at the processors remain between the global all-reduce-sum operation [step 9(c)] of the previous iteration and the fold operation [step 6(b)] of the current iteration, whereas the linear vector operations remain between the fold operation [step 6(b)] of the current iteration and the global all-reduce-sum operation [step 9(c)] of the current iteration. Hence, a two-constraint formulation is needed for load balancing. The two weights associated with vertex v_i are:

$$w_1(v_i) = 2 \times \text{nnz}\,(\text{col } i \text{ of } \mathbf{A}_{11}), \quad w_2(v_i) = 10. \tag{7}$$

4.2 Site-Based Partitioning Models

For the sake of clarity of the following discussions, we assume that the $\mathbf{A}_{11}$ matrix is permuted symmetrically into an $m \times m$ block structure $(\mathbf{A}_{11})_{\mathrm{BL}}$, where rows and columns representing the pages belonging to the same site are ordered consecutively. The sample $\mathbf{A}_{11}$ matrix obtained in

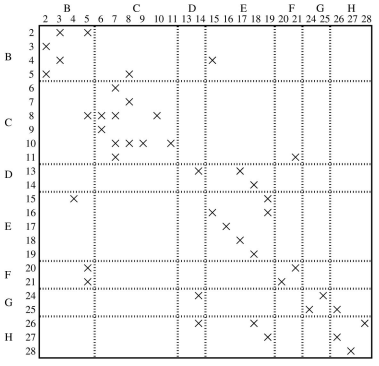


Fig. 7. A 7×7 site-based block structure $(\mathbf{A}_{11})_{\mathrm{BL}}$ of the sample $\mathbf{A}_{11}$ matrix obtained in Fig. 3e.

Fig. 3e is redrawn in Fig. 7 with vertical and horizontal dashed lines indicating the boundaries of the sites to make the site-based row/column ordering clear.

4.2.1 1D Site-by-Page and Page-by-Site Compression

We propose both an SP compression of $\mathbf{A}_{11}$ to construct the $\mathbf{A}_{11}^{\mathrm{sp}}$ matrix and a PS compression of $\mathbf{A}_{11}$ to construct the $\mathbf{A}_{11}^{\mathrm{ps}}$ matrix. The rows and columns of $\mathbf{A}_{11}^{\mathrm{sp}}$ correspond to sites and pages, respectively, whereas this relation is transposed for $\mathbf{A}_{11}^{\mathrm{ps}}$. The weights associated with the nonzeros of $\mathbf{A}_{11}^{\mathrm{sp}}$ and $\mathbf{A}_{11}^{\mathrm{ps}}$ correctly summarize the computational requirements of row-parallel and column-parallel $\mathbf{A}_{11} \times \mathbf{p}_1(\mathbf{A}_{11})$ SpMxVs, respectively. The sparsity patterns of $\mathbf{A}_{11}^{\mathrm{sp}}$ and $\mathbf{A}_{11}^{\mathrm{ps}}$ correctly summarize the communication requirements of row-parallel and column-parallel $\mathbf{A}_{11} \times \mathbf{p}_1(\mathbf{A}_{11})$ SpMxVs, respectively. Hence, it is meaningful to partition a compressed matrix along the dimension of compression. That is, the rowwise compressed $\mathbf{A}_{11}^{\mathrm{sp}}$ matrix is partitioned rowwise to induce a rowwise partition on $\mathbf{A}_{11}$, whereas the columnwise compressed $\mathbf{A}_{11}^{\mathrm{ps}}$ matrix is partitioned columnwise to induce a columnwise partition on $\mathbf{A}_{11}$.

In SP compression, for each site S_r, we compress the rows of $\mathbf{A}_{11}$ corresponding to the pages in site S_r into a single row r of $\mathbf{A}_{11}^{\mathrm{sp}}$. Here, the sparsity pattern of row r is set equal to the union of the sparsities of all rows representing the pages in site S_r. A weighted union is performed so that the weight $w(a_{rj}^{\mathrm{sp}})$ associated with a nonzero of $\mathbf{A}_{11}^{\mathrm{sp}}$ shows the number of nonzeros of $\mathbf{A}_{11}$ combined into the nonzero a_{rj}^{sp}. This rowwise compression corresponds to coalescing the nonzeros representing outlinks of a page pointing to the pages in the same site into a single nonzero. The nonzeros in row r of $\mathbf{A}_{11}^{\mathrm{sp}}$ identify the pages that contain outlinks pointing to site S_r. PS compression is the dual of the SP compression and employs a columnwise compression on $\mathbf{A}_{11}$. This columnwise compression corresponds to coalescing the nonzeros representing the in-links of a page pointed by the pages in the same site into a single nonzero. The nonzeros in column r of $\mathbf{A}_{11}^{\mathrm{ps}}$ identify the pages that are pointed by the outlinks in the pages of site S_r. Figs. 9a and 8a, respectively, show the $\mathbf{A}_{11}^{\mathrm{ps}}$ and $\mathbf{A}_{11}^{\mathrm{ps}}$ matrices for the sample $\mathbf{A}_{11}$ matrix given in Fig. 7.

To enforce symmetric partitioning, zero diagonals of $\mathbf{A}_{11}$ can be replaced by virtual nonzeros with zero weights before compression. A more efficient scheme is to add fewer virtual nonzeros to the compressed matrices to obtain the same effect. In $\mathbf{A}_{11}^{\mathrm{sp}}$, in a row r, zeros in the columns

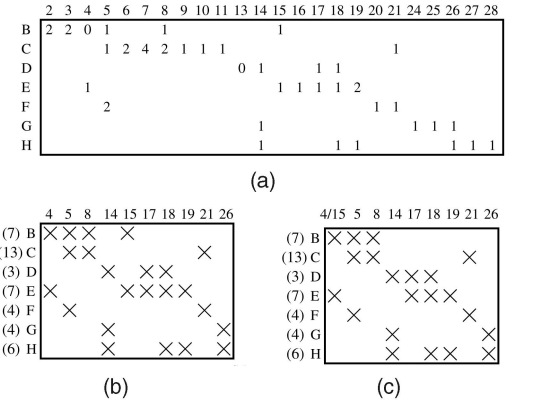


Fig. 8. Site-by-page compressed $\mathbf{A}_{11}^{\mathrm{sp}}$ matrix: (a) After SP compression of $(\mathbf{A}_{11})_{\mathrm{BL}}$ given in Fig. 7, (b) after elimination of the columns that have a single nonzero, (c) after coalescing columns with identical sparsity patterns.

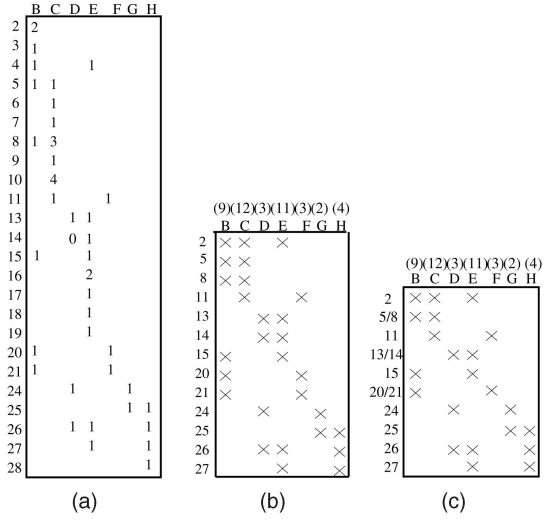


Fig. 9. Page-by-site compressed $\mathbf{A}_{11}^{\mathrm{ps}}$ matrix: (a) after PS compression of $(\mathbf{A}_{11})_{\mathrm{BL}}$ given in Fig. 7, (b) after elimination of rows that have a single nonzero, (c) after coalescing rows with identical sparsity patterns.

corresponding to the pages of S_r are replaced with virtual nonzeros. In $\mathbf{A}_{11}^{\mathrm{ps}}$, in a column r, zeros in the rows corresponding to the pages of S_r are replaced with virtual nonzeros. As seen in Figs. 8a and 9a, $a_{\mathrm{M},4}$ and $a_{\mathrm{Y},13}$ are virtual nonzeros inserted in $\mathbf{A}_{11}^{\mathrm{sp}}$, and $a_{14,\mathrm{Y}}$ is the virtual nonzero inserted in $\mathbf{A}_{11}^{\mathrm{ps}}$ to enforce symmetric partitioning.

Each row r of $\mathbf{A}_{11}^{\mathrm{sp}}$ is associated with a weight $w(a_{r*}^{\mathrm{sp}}) = \sum_{a_{rj}^{\mathrm{sp}} \neq 0} w(a_{rj}^{\mathrm{sp}})$, which is equal to the sum of the number of nonzeros in $\mathbf{A}_{11}$-rows that correspond to the pages of S_r. $w(a_{r*}^{\mathrm{sp}})$ represents the total number of in-links of the pages of site S_r. In a similar manner, each column r of $\mathbf{A}_{11}^{\mathrm{ps}}$ is associated with a weight $w(a_{*r}^{\mathrm{ps}}) = \sum_{a_{ir}^{\mathrm{ps}} \neq 0} w(a_{ir}^{\mathrm{ps}})$. which is equal to the sum of the number of nonzeros in $\mathbf{A}_{11}$-columns that correspond to the pages of S_r. $w(a_{*r}^{\mathrm{ps}})$ represents the total number of outlinks from the pages (with inlinks) of S_r. These weights associated with the rows and columns of $\mathbf{A}_{11}^{\mathrm{sp}}$ and $\mathbf{A}_{11}^{\mathrm{ps}}$ are shown in parantheses next to the row and column identifiers in Figs. 8 and 9.

We propose *single-nonzero* and *sparsity-pattern coalescing* optimizations to sparsen and further compress the compressed $\mathbf{A}_{11}^{\text{sp}}$ and $\mathbf{A}_{11}^{\text{ps}}$ matrices.

The single-nonzero optimization is based on the removal of $\mathbf{A}_{11}^{\text{sp}}$-columns and $\mathbf{A}_{11}^{\text{ps}}$-rows that have a single nonzero, because such $\mathbf{A}_{11}^{\text{sp}}$-columns and $\mathbf{A}_{11}^{\text{ps}}$-rows cannot incur communication in a row-parallel and column-parallel $\mathbf{A}_{11}\times\mathbf{p}_1(\mathbf{A}_{11})$ SpMxV. Such a column of $\mathbf{A}_{11}^{\text{sp}}$ corresponds to a page whose all outlinks are to its own site and such a row of $\mathbf{A}_{11}^{\text{ps}}$ corresponds to a page whose all in-links are from its own site. The weight of each discarded nonzero is still accounted in the weight of the respective row or column of $\mathbf{A}_{11}^{\text{sp}}$ or $\mathbf{A}_{11}^{\text{ps}}$. Figs. 8b and 9b show the $\mathbf{A}_{11}^{\text{sp}}$ and $\mathbf{A}_{11}^{\text{ps}}$ matrices obtained after discarding the columns and rows that have a single nonzero, respectively. As seen in the figures, 15 columns in $\mathbf{A}_{11}^{\text{sp}}$ and 12 rows in $\mathbf{A}_{11}^{\text{ps}}$ are discarded out of 24 rows/columns.

The sparsity-pattern optimization is based on the observation that the columns and rows that have the same sparsity patterns incur the same communication requirement in row-parallel and column-parallel $\mathbf{A}_{11}\times\mathbf{p}_1(\mathbf{A}_{11})$ SpMxVs, respectively. Hence, the columns that have the same sparsity pattern are coalesced into a single column in $\mathbf{A}_{11}^{\text{sp}}$, and the rows that have the same sparsity pattern are coalesced into a single row in $\mathbf{A}_{11}^{\text{ps}}$. The weights of the coalesced nonzeros are still accounted in the weights of the respective rows or columns of $\mathbf{A}_{11}^{\text{sp}}$ or $\mathbf{A}_{11}^{\text{ps}}$ matrices. Furthermore, each representative row or column is associated with an identical-row or identical-column count, which show the number of rows or columns represented by that row or column. Rows and columns that have the same sparsity patterns are efficiently identified by adapting the algorithms given in [32]. Figs. 8c and 9c, respectively, display the further compressed versions of the $\mathbf{A}_{11}^{\text{sp}}$ and $\mathbf{A}_{11}^{\text{ps}}$ matrices obtained by identifying and coalescing the rows and columns that have the same sparsity patterns.

Only the HP models are considered for partitioning $\mathbf{A}_{11}^{\text{sp}}$ and $\mathbf{A}_{11}^{\text{ps}}$ since the GP models are not suitable for partitioning rectangular matrices. For RW, the column-net hypergraph representation of $\mathbf{A}_{11}^{\text{sp}}$ is constructed. The identical-column count associated with a column of $\mathbf{A}_{11}^{\text{sp}}$ is assigned as the cost of the respective net of the hypergraph. The weight $w(v_r)$ of a vertex v_r corresponding to site S_r is computed as:

$$w(v_r) = 2 \times w(a_{r*}^{\text{sp}}) + 10|S_r|, \tag{8}$$

where $|S_r|$ is equal to the number of nondangling pages with in-links in site S_r. For CW, the row-net hypergraph representation of $\mathbf{A}_{11}^{\text{ps}}$ is constructed. The identical-row count associated with a row of $\mathbf{A}_{11}^{\text{ps}}$ is assigned as the cost of the respective net of the hypergraph. The two weights of a vertex v_r are computed as:

$$w_1(v_r) = 2 \times w(a_{*r}^{\text{ps}}), \quad w_2(v_r) = 10|S_r|. \tag{9}$$

4.2.2 2D Site-by-Site Compression

We propose an SS compression of $\mathbf{A}_{11}$ to construct the $\mathbf{A}_{11}^{\text{ss}}$ matrix. $\mathbf{A}_{11}^{\text{ss}}$, which is as an unsymmetric square matrix, is expected to be more compact than the site-by-page and page-by-site compressed matrices. $\mathbf{A}_{11}^{\text{ss}}$ can easily be obtained by performing an SP compression on $\mathbf{A}_{11}$ to obtain $\mathbf{A}_{11}^{\text{sp}}$ and then performing a PS compression on $\mathbf{A}_{11}^{\text{sp}}$ to obtain $\mathbf{A}_{11}^{\text{ss}}$, or vice versa. In $\mathbf{A}_{11}^{\text{ss}}$, a nonzero a_{rs}^{ss} means that there exists $w(a_{rs}^{\text{ss}})$ outlinks from the pages in site S_r to the pages in site S_s. Since compression is performed along both dimensions, rowwise and columnwise partitioning of $\mathbf{A}_{11}^{\text{ss}}$ are both meaningful for inducing a rowwise and a columnwise partition on $\mathbf{A}_{11}$, respectively. Fig. 10 shows the $\mathbf{A}_{11}^{\text{ss}}$ matrix for the sample $\mathbf{A}_{11}$ matrix given in Fig. 7.

	(9) B	(12) C	(3) D	(11) E	(3) F	(2) G	(4) H
(7) B	5	1		1			
(13) C	1	11			1		
(3) D			1	2			
(7) E	1			6			
(4) F	2				2		
(4) G			1			2	1
(6) H			1	2			3

Fig. 10. Site-by-site compressed $\mathbf{A}_{11}^{\text{ss}}$ matrix.

Since $\mathbf{A}_{11}^{\text{ss}}$ is a square matrix, both HP and GP models can be used for partitioning. Even though the weights of the nonzeros of $\mathbf{A}_{11}^{\text{ss}}$ correctly summarize the computational requirements of both row-parallel and column-parallel $\mathbf{A}_{11}\times\mathbf{p}_1(\mathbf{A}_{11})$ SpMxVs, the sparsity pattern of $\mathbf{A}_{11}^{\text{ss}}$ does not correctly summarize the communication requirements. Hence, as the superiority of the HP models depends on the correct modeling of the communication volume, it is not meaningful to use the HP models for partitioning $\mathbf{A}_{11}^{\text{ss}}$. However, by assigning proper edge costs as described below, the GP model can be successfully adopted within the same approximation factor it achieves in the page-based GP model.

For RW and CW schemes, only vertex weights differ in the graph representations of $\mathbf{A}_{11}^{\text{ss}}$, whereas the topology and the edge costs remain the same. For RW, the weight of a vertex v_r is computed according to (9). For CW, the two weights of v_r are computed according to (10). The edge cost $cost(v_r, v_s)$ associated with nonzero(s) a_{rs}^{ss} and/or a_{sr}^{ss} is computed as:

$$cost(v_r, v_s) = w\big(a_{rs}^{\text{ss}}\big) + w\big(a_{sr}^{\text{ss}}\big). \tag{10}$$

That is, $cost(v_r, v_s)$ is equal to the sum of the number of nonzeros in the off-diagonal blocks $(\mathbf{A}_{11})_{rs}$ and $(\mathbf{A}_{11})_{sr}$ of $(\mathbf{A}_{11})_{\text{BL}}$.

5 Repartitioning Models for Parallel PageRank

In a real-world setup, distribution of URLs based on a hash value among the processors could be assumed as a starting point for parallel PageRank computations. In such a setup, the data have to be redistributed among the processors for the sake of efficient parallel PageRank computations. So, the partitioning models should encapsulate the initial data redistribution overhead as well as the communication overhead that will be incurred during the parallel PageRank computations. For this scenario, we propose repartitioning models based on HP and GP with fixed vertices. The URL-based hashing naturally induces either a columnwise or a rowwise partition on $\mathbf{A}$ and hence $\mathbf{A}_{11}$ matrices. Since it is easier to construct the outlink information of pages, we assume a columnwise partitioning. That is, each processor initially stores distinct column slices of $\mathbf{A}_{11}$.

5.1 Page-Based Repartitioning Models

The row-net HP model used for columnwise partitioning of $\mathbf{A}_{11}$ is augmented by K fixed vertices and n_{11} 2-pin nets to construct a row-net repartitioning hypergraph. Fixed vertices represent processors, whereas all other original vertices (herein, called free vertices) represent columns of $\mathbf{A}_{11}$. Newly added two-pin nets encode the initial column-to-processor distribution. That is, each free vertex v_i is connected by a two-pin net to the fixed vertex representing the processor to which column i is initially assigned. The cost of that net is set equal to the number of nonzeros in column i. Each fixed vertex is fixed to a distinct vertex part during the partitioning process.

The column-net HP model used for rowwise partitioning of $\mathbf{A}_{11}$ is augmented by K fixed vertices and l2-pin nets, where l varies between n_{11} and $K \times n_{11}$, to construct a column-net repartitioning hypergraph. Fixed vertices represent processors, whereas free vertices represent rows of $\mathbf{A}_{11}$. Newly added two-pin nets encode the initial column-based nonzero-to-processor distribution. That is, each free vertex v_i is connected by a distinct two-pin net to each of the fixed vertices representing the processors among which the nonzeros of row i are initially distributed. The cost of a net connecting v_i to a fixed vertex is set equal to the number of nonzeros of row i initially residing in the processor corresponding to that fixed vertex.

In both row-net and column-net repartitioning HP models, the vertex partition obtained as a result of the HP process is decoded such that the free vertices in a part denote the columns/rows that will be assigned to the processor corresponding to the unique fixed vertex residing in that vertex part. In a partition of the row-net/column-net repartitioning hypergraph, the cutsize defined over the original nets still shows the communication volume to be incurred during the column-/row-parallel SpMxV of a single PageRank iteration, respectively. The cutsize defined over the newly added two-pin nets shows the communication volume to be incurred during the redistribution of the nonzeros of $\mathbf{A}_{11}$. Proper scaling between the costs of newly added two-pin nets and the unit costs of the original nets should be considered depending on both the expected number of iterations for convergence, and the number of times different PageRank vectors will be computed.

5.2 Site-Based Repartitioning Models

The page-based repartitioning model proposed for columnwise partitioning of $\mathbf{A}_{11}$ extends to site-based columnwise repartitioning of $\mathbf{A}_{11}^{\mathrm{ps}}$ under the assumption that site-based URL hashing is used during the initial distribution. That is, each free vertex v_r representing site S_r is connected by a single two-pin net to the fixed vertex representing the processor to which S_r is assigned initially. The cost of this two-pin net is set equal to the sum of the nonzeros of the columns corresponding to the pages of S_r. However, if site-based hashing is not used during the initial distribution, the free vertex v_r should be connected by several two-pin nets to different fixed vertices with appropriate costs according to the initial distribution of the pages of S_r among the processors. The proposed repartitioning model can be easily extends to the site-based GP model for columnwise partitioning by just replacing each two-pin net of $\mathbf{A}_{11}^{\mathrm{ss}}$ with a graph edge having the same cost.

Fig. 11a displays the row-net repartitioning hypergraph representation of the sample $\mathbf{A}_{11}^{\mathrm{ps}}$ matrix shown in Fig. 9c for a $K = 3$ processor system. Initial site-based columnwise

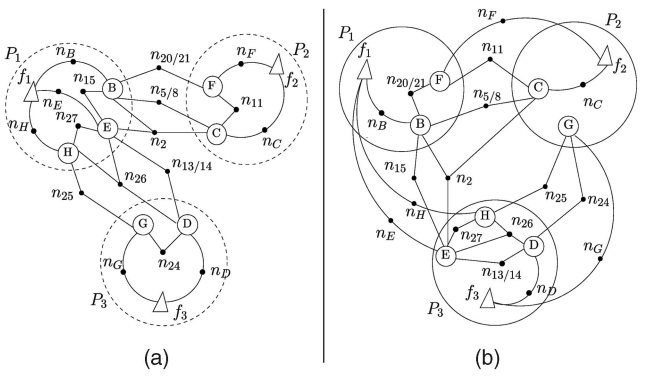


Fig. 11. (a) Row-net repartitioning hypergraph representation of $\mathbf{A}_{11}^{\mathrm{ps}}$ given in Fig. 9c for an initial site-based columnwise distribution on a three-processor system, (b) a sample repartition of the hypergraph given in (a).

distribution is assumed to be according to a round-robin distribution of the sites in Fig. 9c. That is, initially processors P_1, P_2, and P_3, respectively, store the columns corresponding to the pages of site sets {B, E, H}, {C, F}, and {D, G}. In the figure, seven small circles represent free vertices corresponding to sites, and three triangles represent fixed vertices corresponding to processors. Ten dots on straight lines represent original row-nets and seven dots on curved lines represent newly added two-pin nets. For example, $n_{20/21}$, which connects vertices B and F, represents the two original row nets n_{20} and n_{21}, and the two-pin net n_H, which connects free vertex H with fixed vertex f_1, is a newly added net to represent initial assignment of site H to processor P_1. The three large dashed circles in Fig. 11a represent the initial site distribution, and the three large solid circles in Fig. 11b represent the final distribution after the repartitioning. If no repartitioning is applied, the initial distribution of $\mathbf{A}_{11}$-matrix nonzeros among processors P_1, P_2, and P_3 are, respectively, $9 + 11 + 4 = 24$, $12 + 3 = 15$, and $3 + 2 = 5$ nonzeros, with an approximate computational imbalance of 64 percent, whereas the initial communication volume is 9 partial $\mathbf{y}^k$-vector results due to the cut nets, for the SpMxV of each PageRank iteration. The cutsize of the initial partitioning is 9, since it contains six cut-nets each with a part connectivity of 2 and 3 of these cut-nets ($n_{5/8}$, $n_{13/14}$, $n_{20/21}$) have a cost of 2.

As seen in Fig. 11b, the repartitioning process improves the partition by moving site E and H from P_1 to P_3, site F from P_2 to P_1, site G from P_3 to P_2 with a redistribution communication volume of $11 + 3 + 4 + 2 = 20$ nonzeros. The newly added two-pin nets n_{E}, n_{H}, n_{F}, and n_{G}, which connect migrated sites to their original processor vertices, remain on the cut of the repartition of Fig. 11b, thus correctly showing the redistribution costs. The redistribution improves the nonzero distribution as $9 + 3 = 12, 12 + 2 = 14$, and $3 + 11 + 4 = 18$ nonzeros for processors P_1, P_2, and P_3, respectively, thus reducing the computational load imbalance from 64 percent to 23 percent for the SpMxV of each PageRank iteration. The redistribution decreases the communication volume from 9 to $8\mathbf{y}^k$-vector entries for the SpMxV of each PageRank iteration.

The page-based repartitioning model proposed for rowwise partitioning of $\mathbf{A}_{11}$ extends to site-based rowwise repartitioning of $\mathbf{A}_{11}^{\mathrm{sp}}$. The free vertex v_r should be connected by distinct two-pin nets to each of the fixed vertices representing the processors among which the

TABLE 1
Properties of the Data Sets Used in the Experiments

	total number of					*number of links per page*					
							in-links		*out-links*		*intra-site*
dataset	*pages*	*dangling pages*	*pages w/o in-links*	*sites*	*links*	*avg*	*max*	*std*	*max*	*std*	*link ratio*
`google` [27]	913,569	233,370	71,430	15,819	4,480,218	4.9	5,989	33	618	6	87.4%
`balkan`*	816,427	44,529	0	37,722	32,355,385	39.6	27,915	320	4,953	42	79.2%
`de-fr` [24], [26]	8,212,781	4,217,128	69,128	39,307	39,330,881	4.8	11,594	52	1,692	14	92.9%
`webbase` [59]	2,662,002	574,863	269,420	2,199	44,843,770	16.9	75,128	22	4,133	40	95.5%
`indochina` [7], [9]	7,407,256	1,309,090	0	18,984	145,877,411	19.7	256,425	15	5,289	41	92.8%
`arabic-2005` [7], [9]	22,744,080	3,301,085	0	99,924	639,999,458	28.1	575,618	17	9,905	44	89.6%
`uk-2005` [7], [9]	39,459,925	4,650,964	0	587,205	936,364,282	23.7	1,776,852	1,655	5,213	37	80.0%
`uk-union` [7], [9]	133,633,040	12,129,754	0	256,099	5,507,679,822	41.2	6,366,524	1,743	22,429	116	93.8%

* We obtained the `balkan` dataset by crawling pages from the Balkan countries using the Larbin [47] crawler.

nonzeros of the rows corresponding to the pages of site S_r are initially distributed. That is, a free vertex v_r is connected by multiple distinct two-pin nets to fixed vertices representing each of the processors which crawled the pages linking to S_r. The costs of these two-pin nets are set according to the initial distribution of the pages linking site S_r among different processors. The proposed repartitioning model easily extends to the site-based GP model for rowwise partitioning of A_{11}^{ss} by just replacing each two-pin net with a graph edge having the same cost. Note that we are not presenting an example for the site-based column-net repartitioning model due to lack of space.

6 Experimental Results

6.1 Implementation Details

The compression schemes discussed in Section 4.2 are implemented in C. In the experiments, the two-constraint formulations ((7) and (9)) proposed for separately balancing the SpMxV and linear-vector operations are implemented as single-constraint formulations due to the following two reasons: First, the relative performances of the GP and HP tools differ in multiconstraint partitioning. Second, varying speedup values are obtained depending on the relative values of the maximum allowable imbalance ratios for the two constraints. We wanted to avoid such variations since the main purpose of this paper is to compare the compression schemes discussed in Section 4. In the adopted single-constraint implementations, the two vertex weights, representing the matrix-vector multiplication and linear vector operations, are added up to form a single vertex weight that represents an aggregate computational weight. We conducted limited experiments to compare the relative performance of two- and single-constraint formulations and observed that the former gives slightly better speedups than the latter. For example, in the columnwise partitioning of the $\mathbf{A}_{11}^{ss}$ matrices, the two-constraint formulation leads to an average speedup improvement of 2.16 percent on 40 processors, compared to the single-constraint formulation.

The parallel PageRank algorithm in Fig. 5 is implemented using a library for parallel SpMxVs [57]. This library implements both row and column-parallel SpMxV algorithms using MPI. Double-precision (64-bit) arithmetic is used in the implementations.

In our parallel PageRank algorithm, because of the adopted site-based partitioning schemes, site-based ordering is naturally exploited during the local SpMxVs. This is known to reduce the sequential SpMxV times in PageRank computations [40], [52]. For the sake of fairness in speedup computations, we also exploit the site-based row/column ordering for the SpMxVs in the sequential PageRank computations.

6.2 Data Set Properties

Table 1 summarizes the properties of the data sets used in the experiments. In Table 1, the data sets are listed in increasing order of the number of links, and they are divided into two groups by a horizontal line. The first and second group of data sets are referred to as the medium and large data sets. Experiments on medium data sets are conducted on a small-memory cluster in order to show the validity of the proposed site-based compression and partitioning schemes. Experiments on large data sets are conducted on a large-memory cluster in order to show the validity of the proposed repartitioning models.

As seen in Table 1, the number of dangling pages varies between 5.5 percent (`balkan`) and 51.3 percent (`de-fr`) of the total number of pages. The domain specific `google` and `webbase` data sets contain a significant number of pages without in-links (7.8 percent and 10.1 percent, respectively), whereas the other data sets (except `de-fr`) contain no pages without in-links. In Table 1, the *avg* and *max* columns denote average and maximum number of links per page and the *std* column denotes standard deviation in pages' link distribution. Existence of high *std* values complies with the fact that the web data behaves according to the power law [5]. The last column displays the number of intrasite links as a percent of the total number of links. The number of intra-site links varies between 79.2 percent and 95.5 percent of the total number of links. These figures conform with the previous observations [40], indicating that sites constitute natural page clusters.

6.3 Experimental Framework

The validity of proposed site-based (re)partitioning models are tested in terms of both preprocessing time and (re)partitioning quality. The (re)partitioning quality is assessed in terms of computational load imbalance, communication overhead, and speedup.

The communication overhead mainly depends on the following metrics: total message volume, maximum message volume handled by a processor, total message count, and maximum message count handled by a processor [32], [33], [58]. Since the test matrices are sufficiently large and speedups are obtained on small-to-medium number of processors, message volume metrics mainly determine the communication overhead. In accordance, we also report here

TABLE 2
Properties of the Matrices

		number of			*nnz per row*			*nnz per column*		
dataset	*matrix*	*rows*	*columns*	*nonzeros*	*max*	*avg*	*std*	*max*	*avg*	*std*
google	$\mathbf{A}$	913,569	913,569	4,480,218	5,989	4.90	32.67	618	4.90	6.43
	$\mathbf{A}_{11}$	608,769	608,769	3,839,113	5,398	6.30	35.27	618	6.30	6.42
	$\mathbf{A}_{11}^{\mathrm{sp}}$	15,819	51,863	194,334	935	12.28	42.66	134	3.74	3.18
	$\mathbf{A}_{11}^{\mathrm{ps}}$	35,099	15,819	145,760	319	4.15	8.02	546	9.21	27.46
	$\mathbf{A}_{11}^{\mathrm{ss}}$	15,819	15,819	78,901	275	4.99	2.71	327	4.99	3.04
balkan	$\mathbf{A}$	816,427	816,427	32,355,385	27,915	39.63	320.50	4,953	39.63	42.36
	$\mathbf{A}_{11}$	771,895	771,895	31,794,117	27,915	41.18	325.73	4,952	41.18	42.22
	$\mathbf{A}_{11}^{\mathrm{sp}}$	37,722	63,372	703,685	7,770	18.65	138.47	791	11.10	18.72
	$\mathbf{A}_{11}^{\mathrm{ps}}$	45,214	37,722	395,120	4,982	8.74	70.99	5,286	10.47	42.29
	$\mathbf{A}_{11}^{\mathrm{ss}}$	37,722	37,722	324,153	4,807	8.59	8.82	5,039	8.59	14.54
de-fr	$\mathbf{A}$	8,212,781	8,212,781	39,330,881	11,594	4.79	52.45	1,692	4.79	13.64
	$\mathbf{A}_{11}$	3,949,196	3,949,196	28,215,831	11,590	7.14	64.89	1,597	7.14	11.35
	$\mathbf{A}_{11}^{\mathrm{sp}}$	39,307	105,464	435,674	4,529	11.08	70.97	184	4.13	4.99
	$\mathbf{A}_{11}^{\mathrm{ps}}$	82,710	39,307	371,173	1,136	4.49	7.37	5,366	9.44	59.22
	$\mathbf{A}_{11}^{\mathrm{ss}}$	39,307	39,307	207,937	4,246	5.29	35.14	1,507	5.29	28.40
webbase	$\mathbf{A}$	2,662,002	2,662,002	44,843,770	75,128	16.85	217.66	4,133	16.85	39.82
	$\mathbf{A}_{11}$	1,817,719	1,817,719	37,486,201	75,128	20.22	219.58	3,979	20.22	42.54
	$\mathbf{A}_{11}^{\mathrm{sp}}$	2,199	11,915	42,002	1,555	19.10	65.06	49	3.52	2.25
	$\mathbf{A}_{11}^{\mathrm{ps}}$	6,209	2,199	27,204	184	4.38	7.41	401	12.37	29.76
	$\mathbf{A}_{11}^{\mathrm{ss}}$	2,199	2,199	13,148	103	5.98	0.47	240	5.98	0.59

that maximum message counts are observed to be very close to the upper bound of $K-1$ during both parallel matrix-vector multiplications and initial data redistribution.

The speedup values are measured on two PC clusters. The first (small-memory) cluster has 40 nodes interconnected by a nonblocking Fast Ethernet switch, and each node contains a single-core Intel P4 3GHz processor with 2 Gbytes of RAM. The second (large-memory) cluster [53] has 32 nodes. Each node has 32 Gbytes of RAM and contains eight AMD Opteron Dual Core 2.4 GHz processors. This cluster is interconnected by a nonblocking InfiniBand switch (20 Gbytes/s).

The sequential power method implementation given in Fig. 4 is used for speedup computations. The sequential running times are measured as 11.5, 35.1, 96.1, and 68.5 s for google, balkan, de-fr, and webbase, respectively. We have also implemented the Gauss-Seidel algorithm [1] so as to determine the best sequential algorithm to take as a benchmark. As the conventional Gauss-Seidel algorithm is not amenable to parallelization, we were able to test its convergence performance through its sequential implementation only on the medium data sets. With $\alpha = 0.90$ and $\varepsilon = 10^{-6}$, the power method and Gauss-Seidel algorithms converge in 94, 87, 85, 90, and 46, 52, 45, 46 iterations for the medium data sets google, balkan, de-fr, and webbase, respectively. These results conform with the results of [1] in that Gauss-Seidel usually converges in approximately half the number of iterations of the power method in PageRank computations. The parallel power method algorithm given in Fig. 5 converges in 87, 90, 90, and 88 iterations for the large data sets indochina, arabic-2005, uk-2005, and uk-union, respectively.

For partitioning experiments, the medium data sets are assumed to be available in a single processor of the small-memory cluster initially. For repartitioning experiments, the large data sets are assumed to be distributed among the processors of the large-memory cluster initially.

The sequential GP tool MeTiS [51] and the sequential HP tool PaToH [3], [17] are used to partition the matrices of the medium data sets. As the hypergraph representations of the matrices belonging to the large data sets do not fit into memory, the parallel HP tool Zoltan [18], which supports fixed vertices, is used to partition the repartitioning hypergraphs proposed in Section 5 for the large data sets. The maximum allowable imbalance ratio is selected as 10 percent for all partitioning tools.

The performance results are given for $K = 4$, 8, 16, 24, 32, 40, and 64-way partitioning of the test matrices given in Table 2. For each K value, the partitioning of each test matrix under a given model constitutes a partitioning instance. Since MeTiS, PaToH, and Zoltan use randomized algorithms, the experiments were run 10 times starting with randomly selected seeds for every partitioning instance. Each value (including speedup values) displayed in the following figures shows the average of 10 values.

6.4 Site-Based Partitioning Experiments on Medium Data Sets

For the site-based partitioning models, preprocessing times involve both matrix compression and partitioning times, whereas for the page-based partitioning models, preprocessing times involve only the partitioning time. For the partitioning experiments, the relative performance of the analyzed models follow a similar pattern in terms of maximum and total message volume metrics. Hence, only total message volumes are reported in this section.

6.4.1 Site-Based Compression Results

Table 2 displays the properties of the matrices obtained from the medium data sets. Note that the $\mathbf{A}_{11}$ matrix is used throughout the iterations of the proposed sequential and

TABLE 3
Percent Size Reduction in Compressed Matrices

	$\mathbf{A}_{11}^{sp}$			$\mathbf{A}_{11}^{ps}$		
	columns			*rows*		
	single nnz	*identical sparsity*	*total nnz*	*single nnz*	*identical sparsity*	*total nnz*
google	67.36	24.12	79.49	89.78	4.46	80.54
balkan	21.61	70.18	87.48	80.68	13.46	69.04
de-fr	89.75	7.58	90.81	96.13	1.78	91.42
webbase	79.74	19.60	98.27	98.67	0.98	98.54

parallel PageRank algorithms, as shown in Figs. 4 and 5. Comparison of the sizes of $\mathbf{A}$ and $\mathbf{A}_{11}$ matrices shows that handling dangling pages and pages with no in-links leads to a drastic reduction in row and column sizes as well as nonzero counts. As expected, $\mathbf{A}_{11}$ matrices are denser than $\mathbf{A}$ matrices for all medium data sets. In both $\mathbf{A}$ and $\mathbf{A}_{11}$, the variation (*std* values) of the nonzero counts over the rows is considerably higher than that of the columns. This implies that there is a greater variation over the in-link counts of the pages compared to the outlink counts. Note that the variations of the nonzero counts over both rows and columns of $\mathbf{A}_{11}^{ss}$ matrices are significantly less than those over the rows and columns of $\mathbf{A}_{11}^{sp}$ and $\mathbf{A}_{11}^{ps}$ matrices.

Table 3 shows the effect of eliminating rows/columns that have a single nonzero and coalescing rows/columns that have identical sparsity patterns. In the table, the reductions in the number of rows/columns and nonzeros are given as percentages of the number of rows/columns and nonzeros of the initial compressed $\mathbf{A}_{11}^{ps}$ and $\mathbf{A}_{11}^{sp}$ matrices. As seen in Table 3, a drastic reduction is obtained in the number of rows/columns due to the elimination of rows/columns with a single nonzero.

6.4.2 Performance Comparison of Page and Site-Based Schemes

We present the comparison of the page-based and site-based partitioning schemes only for the smallest data set google. Figs. 12 and 13, respectively, display the preprocessing time and partitioning quality of the page and site-based partitioning schemes with increasing number of processors. In these two figures, the results for HP-based partitioning models are presented.

As seen in Fig. 12, the proposed site-based partitioning schemes achieve drastic reductions in preprocessing times compared to the page-based schemes. For example, the site-based schemes perform the preprocessing approximately 8 and 14 times faster than the page-based schemes in RW and CW partitioning of the google data set, on average. The

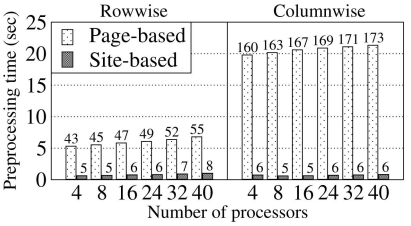


Fig. 12. Comparison of preprocessing times of page-based and site-based partitioning schemes for the google data set (on the small-memory cluster).

considerable difference between the RW and CW partitioning times of the page-based scheme can be explained as follows: In some parts of the HP partitioning tool PaToH, the running time increases with the square of the net sizes. As seen in Table 2, the $\mathbf{A}_{11}$ matrix for the google data set contains dense rows, which results in nets with high size in the row-net HP model used for CW partitioning. In Fig. 12, the number annotated with each bar shows the ratio of the preprocessing time to the sequential per-iteration time.

As seen in Fig. 13, as expected, the site-based schemes perform considerably worse than the page-based schemes (especially in RW partitioning) in terms of load balancing. On the other hand, the site-based schemes obtain slightly better partitions than the page-based schemes in terms of total message volumes. Despite higher load imbalance, the site-based schemes achieve slightly better speedups. These findings justify the validity of site-based schemes, which enable the use of the state-of-the-art sparse matrix partitioning models and tools at an affordable preprocessing cost.

6.4.3 Performance Comparison of Site-Based Compression Schemes

In this section, RW-SS and CW-SS, respectively, refer to rowwise and columnwise partitioning of the $\mathbf{A}_{11}^{ss}$ matrices, and RW-SP and CW-PS, respectively, refer to 1D rowwise and columnwise partitioning of the $\mathbf{A}_{11}^{sp}$ and $\mathbf{A}_{11}^{ps}$ matrices.

Fig. 14 displays a comparison of the preprocessing times of the site-based partitioning schemes for medium data sets with increasing number of processors. Similar to Fig. 12, the number annotated with each bar shows the ratio of the preprocessing time to the sequential per-iteration time. As seen in Fig. 14, preprocessing times vary between 0.5 to 10.3 iterations of sequential runs of the PageRank algorithm. It is nice to observe that the preprocessing time increases very slowly with increasing number of processors for all compression schemes. This is due to the fact that only the partitioning component depends on the number of processors. As seen in the figure, the SS schemes incur less preprocessing time than

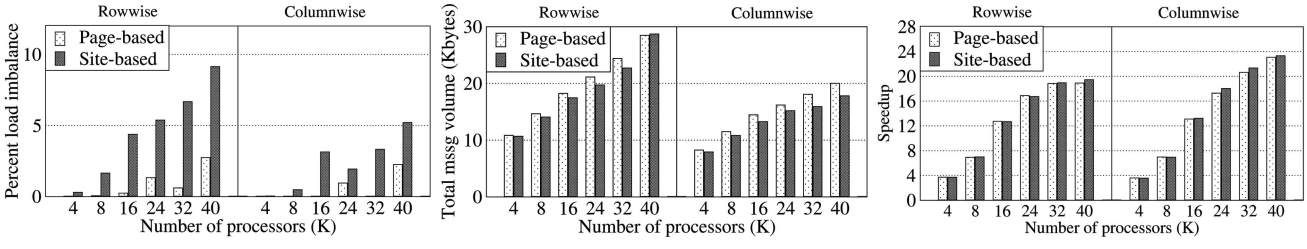


Fig. 13. Comparison of the partitioning quality of page-based and site-based schemes for the google data set.

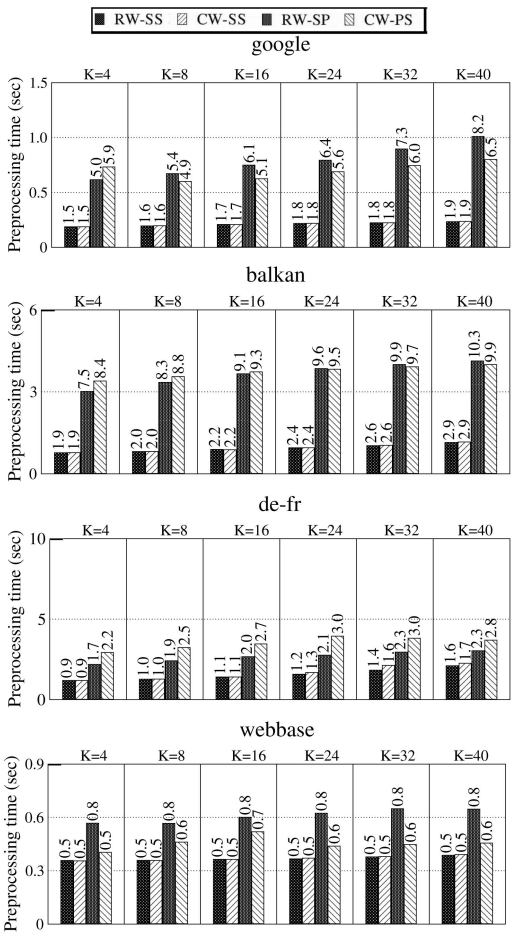


Fig. 14. Preprocessing time comparison of site-based schemes for medium data sets (on the small-memory cluster).

the SP and PS schemes. This is because $\mathbf{A}_{11}^{\text{ss}}$ matrices are considerably smaller than the $\mathbf{A}_{11}^{\text{sp}}$ and $\mathbf{A}_{11}^{\text{ps}}$ matrices, and HP takes relatively more time than GP.

Fig. 15 displays the relative performance of the compression and partitioning schemes for the medium data sets, with increasing number of processors. The factors that affect the relative computational load balancing performance of different partitioning schemes are the partitioning tools and the data set properties. As seen in the first column of Fig. 15, load imbalance values are smaller than 10 percent in all but two partitioning instances (40-way RW-SP and CW-PS partitionings of `balkan`). The GP tool MeTiS shows better load balancing performance than the HP tool PaToH in partitioning the respective compressed matrices. The load imbalance values obtained for `balkan` are considerably higher than those for the other data sets. This is due to the considerably higher vertex weight variation of `balkan`.

In Fig. 15, relative performance comparison of RW and CW partitioning models under the same compression scheme corresponds to comparing the first and second bars under the SS compression scheme, and the third and fourth bars under the SP and PS compression schemes, for a fixed (K, data set) pair. In terms of the message volume metric (the second column of Fig. 15), the CW schemes CW-SS and CW-PS, respectively, perform considerably better than the RW schemes RW-SS and RW-SP in all partitioning instances. This finding can be attributed to the expectation that the similarity among outlink patterns of pages are higher than the similarity among in-link patterns of pages. Thus, $\mathbf{A}_{11}^{\text{ps}}$ matrices are more amenable to produce better partitions. In terms of the speedup metric, (the third column of Fig. 15), CW schemes lead to better speedups than RW schemes in all partitioning instances except in four and eight-way partitionings of `google` and 40-way partitioning of `de-fr` under SP/PS compression. In general, the speedup gap between RW and CW schemes conforms with the gap observed in message volumes. Especially, in `balkan` and `webbase`, large differences between RW and CW schemes in message volumes cause high-speedup differences. In both data sets, speedups obtained by RW schemes begin to saturate around $K = 24$, whereas speedups obtained by CW schemes continue to scale almost linearly.

In Fig. 15, the relative performance comparison of the 1D compression schemes (SP and PS) and the 2D compression schemes (SS) under the same partitioning models corresponds to comparing the first and third bars under the RW partitioning model, and the second and fourth bars under the CW partitioning model. In terms of the message volume metric, RW-SP and CW-PS, respectively, perform better than RW-SS and CW-SS in all partitioning instances. However, the 2D compression leads to better load balancing than the 1D compression as discussed earlier. Thus, 1D and 2D compression schemes in general lead to comparable speedup performances.

6.5 Repartitioning Experiments on Large Data Sets

Repartitioning experiments are carried out according to an inital columnwise distribution of the A matrix (which is obtained by a URL hash-based distribution of the sites as mentioned in Section 5) to simulate a real-world setup. Since CW-PS performs better than other schemes for partitioning instances of medium data sets in general, experiments for large data sets are conducted only for the CW-PS scheme.

Fig. 16 displays the variation of the computational-load rebalancing performance of the repartitioning model with increasing number of processors. As seen in Fig. 16, repartitioning reduces the load imbalance significantly. Comparison of Figs. 15 and 16 shows that considerably higher load imbalance values are observed for repartitioning experiments conducted on large data sets (except for `uk-2005`) compared to partitioning experiments conducted on medium data sets. This observation may be attributed to high initial imbalances observed on the repartitioning experiments. A highly imbalanced initial distribution may restrict the solution space of the repartitioning tool in finding solutions that have both low imbalance and small redistribution overhead.

Fig. 17 displays the variation of the maximum message volume handled by a processor and the total message volume during the data redistribution and parallel PageRank computation phases for 16 and 32 processors. As seen, in both phases, total communication volume increases with increasing number of processors, whereas maximum message volume per processor decreases (except for `uk-union`) in the redistribution phase. Hence, a scalable performance increase can be expected in the redistribution and PageRank computation phases for nonblocking switches.

Fig. 18 displays the variation of data redistribution and parallel PageRank computation times with increasing number of processors. Both redistribution time and parallel PageRank computation time decreases considerably with

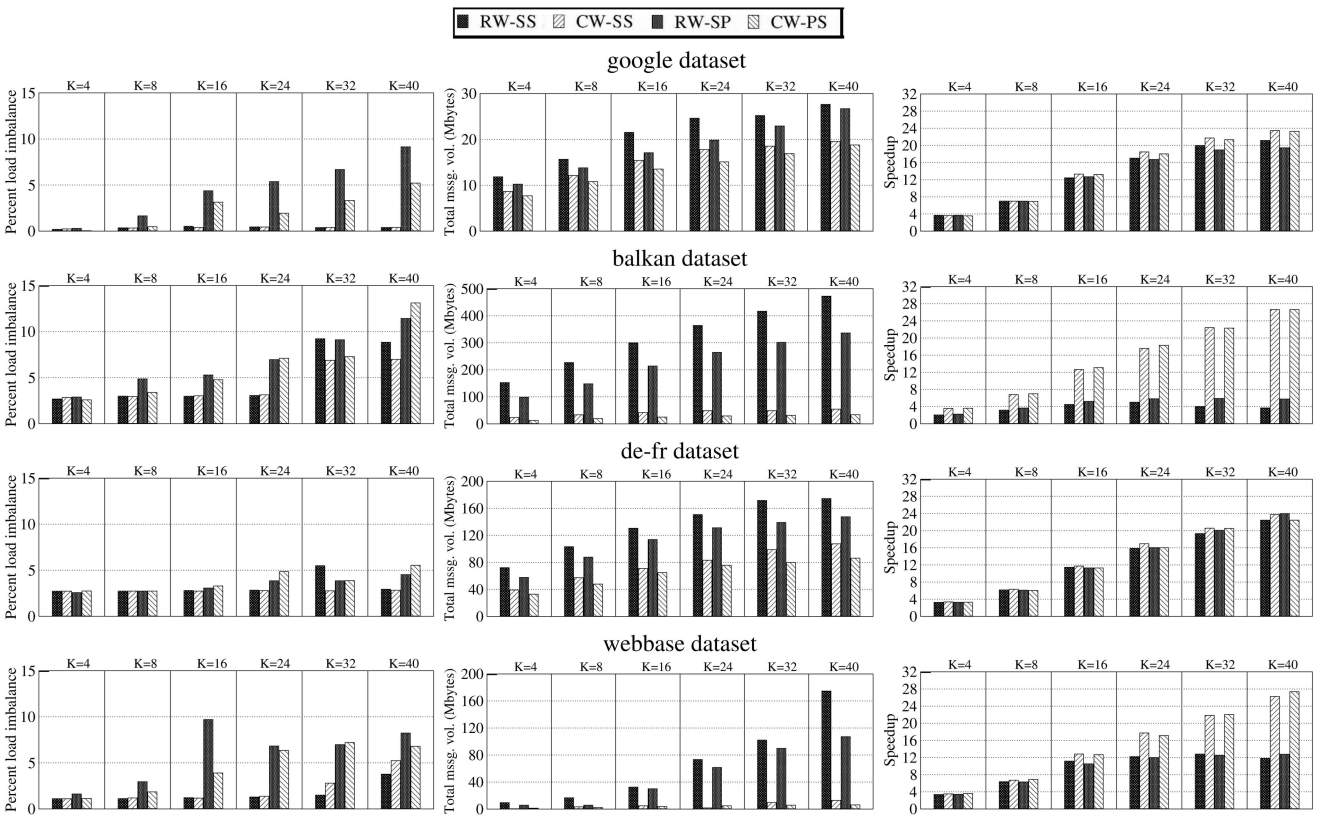


Fig. 15. Comparison of the partitioning qualities of the site-based schemes for medium data sets. (Speedup values are taken on the small-memory cluster.)

increasing number of processors, where this decrease is much more pronounced in the parallel PageRank computation times. The preprocessing overhead due to data redistribution seems to amortize even for a single PageRank vector computation up to 64 processors. We should note here that the communication times both during the data redistribution and PageRank computations can be much higher if the repartitioning, data redistribution, and parallel PageRank computations are to be performed on a slow communication medium such as a WAN, thus increasing the parallelization overhead and decreasing the speedup values. However, since the communication volumes during the data redistribution and parallel PageRank computation phases are comparable as seen in Fig. 17, the preprocessing overhead is still expected to amortize.

Fig. 19 presents the speedup curves for the large data sets. The parallel PageRank computations for the `indochina`, `arabic-2005`, and `uk-2005` data sets can be executed on $K \geq 4$ processors, whereas for `uk-union` it can be executed on $K \geq 16$ processors. Hence, the speedup values for the former and latter data set groups are computed with respect to the parallel computation times on $K = 4$ and $K = 16$ processors, respectively. The PageRank computation times are measured as 61.3, 264.3, and 434.8 seconds on four processors for the `indochina`, `arabic-2005`, and `uk-2005` data sets, respectively, and computation time is measured as 642.4 seconds on 16 processors for the `uk-union` data set. As seen in the figure, almost linear speedups are obtained in the given range of processors.

6.6 Alternative Approaches

PageRank computations in real-life problems require large amounts of memory, unless out-of-core algorithms or web-data compression techniques are used. For example, storing

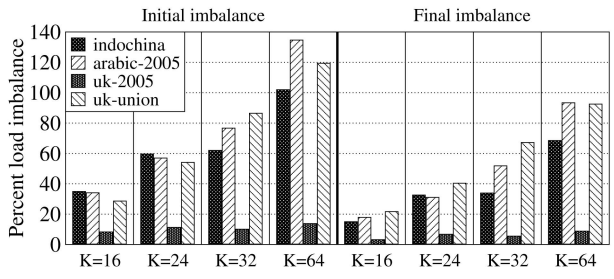


Fig. 16. Percent load imbalance values for large data sets.

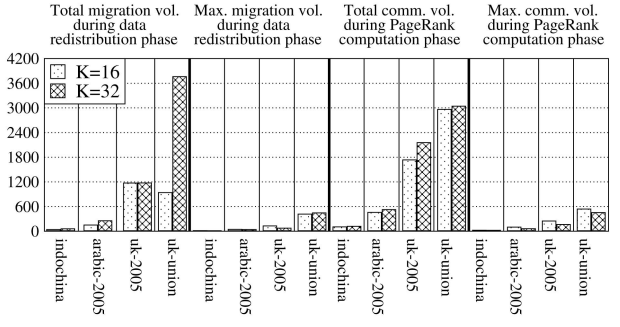


Fig. 17. Communication volumes (Mbytes) for large data sets.

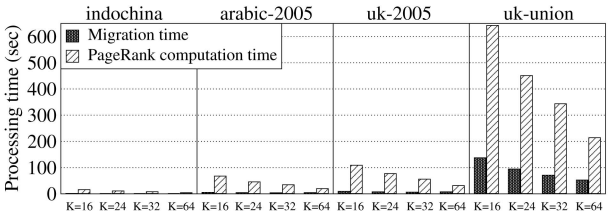


Fig. 18. Comparison of redistribution and parallel PageRank computation times for large data sets (on the large-memory cluster).

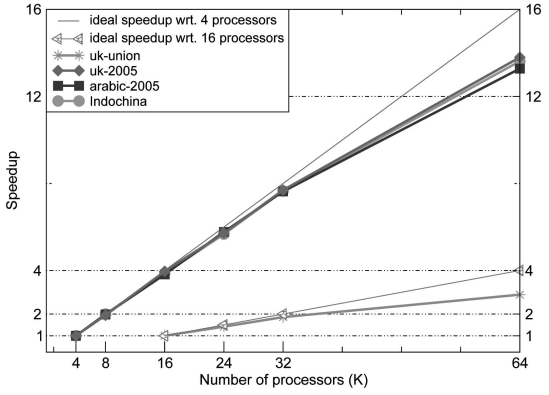


Fig. 19. Speedup curves for large data sets (on the large-memory cluster).

the web matrix for uk-union in CSR format with double precision values requires approximately 62 Gbytes of memory, whereas the PageRank vector for these data requires approximately 1 Gbyte of memory. Considering billions of pages and trillions of links, terabytes of memory are required to store the whole web matrix.

One approach to tackle this memory problem for small-scale parallel systems is disk buffering during PageRank computations, as proposed in [50]. However, this approach drastically increases the PageRank computation time. Our aim in this paper is to reduce PageRank computation time on large-scale systems via efficient parallelization. Hence, we avoid demonstrating results with disk buffering, for better presentation of proposed techniques and fair comparisons of speedup values.

Another approach for reducing the memory requirement is to store the web matrix in a compressed form. In [7], an effective compression technique is explained. Implementation of PageRank using compressed matrices is publicly available as LAW codes [8]. Using this implementation, it requires only several Gbytes to calculate PageRank for the uk-union data set. However, with LAW codes, it takes more than two hours to complete only one iteration of PageRank computation on a single processor, whereas in our implementation, it takes less than 6 min to complete the whole PageRank computation on 32 processors (3.8 s/iteration).

Another alternative to manage large web matrices is to employ distributed PageRank algorithms, for data stored on geographically distributed data centers. Wang and DeWitt [60] discuss the scalability problem for central crawling and propose a PageRank algorithm for a distributed search engine. The algorithm produces an approximation to the global PageRank vector. For such a distributed PageRank algorithm, our proposed techniques can be applied to calculate PageRank vectors that are local to data centers.

7 Conclusion and Future Work

Sparse matrix partitioning models are shown to be quite successful in load balancing and minimizing the total volume of communication during the repeated parallel SpMxV in PageRank computations. However, the vast sizes of the web matrices and the high-preprocessing overhead incurred by these models render this solution infeasible in practice. As a remedy, this paper investigated several site-based compression schemes and sparse matrix partitioning algorithms, targeting to obtain acceptable preprocessing overheads compared to the previously used page-based schemes. The results indicate that with similar load balance values and total communication volumes achieved, preprocessing overheads can be drastically reduced in site-based schemes. Among the several schemes tested, column-wise partitioning together with page-to-site compression is found to achieve much better performance. This can be attributed to the higher similarity among outlink patterns of pages than the inlink patterns of pages. We should note here that this finding slightly contradicts with the findings in a recent work [10], and this discrepancy requires further study. Furthermore, the proposed matrix repartitioning algorithms, which handle the data migration costs arising due to the already distributed nature of web matrices are also tested on very large scale real-world data and found to yield good speedups with scalable overheads.

As future work, we are planning to investigate the following issues: As the LAW codes minimize storage requirement through compression, the application and adaptation of the parallelization models and methods in this paper can be applied for efficient parallelization of LAW codes for the solution of large problems on parallel systems with relatively small memory. As the recursive lumping-based reordered PageRank algorithm of [45] reduces the size of the iteration matrix significantly, it should be investigated for efficient parallelization as well. The models and methods proposed in this work apply to the efficient parallelization of the first subsystem of [45]. However, the parallelization of the numerous SpMxV operations to be performed for finding the remaining subvectors needs research effort because of the highly sequential nature of the forward substitution scheme.

Acknowledgments

The authors would like to thank the anonymous referees, whose comments substantially improved the quality of this paper. This work is partially supported by The Scientific and Technological Research Council of Turkey under Grant EEEAG-109E019 and COST Action IC080 ComplexHPC.

References

[1] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin, "PageRank Computation and the Structure of the Web: Experiments and Algorithms," *Proc. 11th Int'l World Wide Web (WWW) Conf.*, Poster, 2002.

[2] C. Aykanat, F. Ozguner, and D. Scott, "Vectorization and Parallelization of the Conjugate Gradient Algorithm on Hypercube-Connected Vector Processors," *J. Microprocessing and Microprogramming*, vol. 29, pp. 67-82, 1990.

[3] C. Aykanat, B.B. Cambazoglu, and B. Ucar, "Multilevel Hypergraph Partitioning with Multiple Constraints and Fixed Vertices," *J. Parallel and Distributed Computing,* vol. 68, pp. 609-625, 2008.

[4] C. Aykanat, B.B. Cambazoglu, F. Findik, and T. Kurc, "Adaptive Decomposition and Remapping Algorithms for Object-Space-Parallel Direct Volume Rendering of Unstructured Grids," *J. Parallel and Distributed Computing,* vol. 67, pp. 77-99, 2006.

[5] A.-L. Barabasi and R. Albert, "Emergence of Scaling in Random Networks," *Science,* vol. 286, pp. 509-512, 1999.

[6] P. Berkhin, "A survey on PageRank Computing," *Internet Math.,* vol. 2, no. 1 pp. 73-120, 2005.

[7] P. Boldi and S. Vigna, "The WebGraph Framework I: Compression Techniques," *Proc. 13th Int'l World Wide Web (WWW) Conf.,* pp. 595-602, 2004.

[8] P. Boldi and S. Vigna, "Codes for the World Wide Web," *Internet Math.,* vol. 2, pp. 405-427, 2004.

[9] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "UbiCrawler: A Scalable Fully Distributed Web Crawler," *Software: Practice and Experience,* vol. 43, pp. 711-726, 2004.

[10] P. Boldi, M. Santini, and S. Vigna, "Permuting Web Graphs," *Proc. Sixth Int'l Workshop Algorithms and Models for the Web-Graph,* pp. 116-126, 2009.

[11] J.T. Bradley, D.V. De Jager, W.J. Knottenbelt, and A. Trifunovic, "Hypergraph Partitioning for Faster Parallel PageRank Computation," *Lecture Notes in Computer Science,* pp. 155-171, Springer, 2005.

[12] C. Brezinski, M. Redivo-Zaglia, and S. Serra-Capizzano, "Extrapolation Methods for PageRank Computations," *Comptes Rendus de l'Académie des Sciences de Paris,* vol. 340, pp. 393-397, 2005.

[13] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," *Computer Networks and ISDN Systems,* vol. 33, no. 3, pp. 107-117, 1998.

[14] B.B. Cambazoglu and C. Aykanat, "Hypergraph-Partitioning-Based Remapping Models for Image-Space-Parallel Direct Volume Rendering of Unstructured Grids," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 1, pp. 3-16, Jan. 2007.

[15] U.V. Çatalyürek and C. Aykanat, "Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 7, pp. 673-693, July 1999.

[16] U.V. Çatalyürek, C. Aykanat, and B. Ucar, "On Two-Dimensional Sparse Matrix Partitioning: Models, Methods and a Recipe," *SIAM J. Scientific Computing,* vol. 32, no. 2, pp. 656-683, 2010.

[17] U.V. Çatalyürek and C. Aykanat, "A Multilevel Hypergraph Partitioning Tool, V. 3.0," technical report, Dept. of Computer Eng., Bilkent Univ., 1999.

[18] U.V. Çatalyürek, E.G. Boman, K.D. Devine, D. Bozdag, R.T. Heaphy, and L.A. Riesen, "Dynamic Load Balancing for Adaptive Scientific Computations via Hypergraph Partitioning," *J. Parallel and Distributed Computing,* vol. 69, no. 8, pp. 711-724, 2009.

[19] A. Cevahir, C. Aykanat, A. Turk, and B. Barla Cambazoglu, "Site-Based Partitioning Tool for Parallel PageRank Computation," http://matsu-www.is.titech.ac.jp/~ali/pagerank.html, 2010.

[20] A. Cevahir, C. Aykanat, A. Turk, and B. Barla Cambazoglu, "Web-Site-Based Partitioning Techniques for Reducing the Preprocessing Overhead before the Parallel PageRank Computations," *Applied Parallel Computing. State of the Art in Scientific Computing,* Springer, June 2006.

[21] J. Cho and H. Garcia-Molina, "The Evolution of the Web and Implications for an Incremental Crawler," *Proc. World Wide Web Conf.,* pp. 200-209, May 1999.

[22] G.M. Del Corso, A. Gullí, and F. Romani, "Comparison of Krylov Subspace Methods on the PageRank Problem," *J. Computational and Applied Math.,* vol. 210, pp. 159-166, 2007.

[23] G.M. Del Corso, A. Gullí, and F. Romani, "Fast PageRank Computation via a Sparse Linear System," *Internet Math.,* vol. 2, no. 3, pp. 251-273, 2005.

[24] Web Graph Benchmark: http://hipercom.inria.fr/~viennot/webgraph, 2010.

[25] D. Gleich, L. Zhukov, and P. Berkhin, "Fast Parallel PageRank: A Linear System Approach," Technical Report YRL-2004-038, Yahoo!, 2004.

[26] J.-L. Guillaume, M. Latapy, and L. Viennot, "Efficient and Simple Encodings for the Web Graph," *Proc. 11th Int'l World Wide Web (WWW) Conf.,* 2002.

[27] Google Programming Contest: http://www.google.com/programming-contest/, 2004.

[28] http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html, 2010.

[29] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Combating Web Spam with TrustRank," *Proc. 30th Very Large Data Bases (VLDB) Conf.,* vol. 1, pp. 257-263, 2004.

[30] G.H. Golub and J.F.V. Loan, *Matrix Computation,* third ed. John Hopkins Univ. Press, 1996.

[31] T. Haveliwala, "Topic Sensitive PageRank," *Proc. 11th Int'l World Wide Web Conf.,* pp. 517-526, 2002.

[32] B. Hendrickson and E. Rothberg, "Improving the Run Time and Quality of Nested Dissection Ordering," *SIAM J. Scientific Computing,* vol. 20, no. 2, pp. 468-489, 1998.

[33] B. Hendrickson and T.G. Kolda, "Graph Partitioning Models for Parallel Computing," *Parallel Computing,* vol. 26, pp. 1519-1534, 2000.

[34] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms.* Computer Science Press, 1978.

[35] I.C.F. Ipsen and S. Kirkland, "Convergence Analysis of a PageRank Updating Algorithm by Langville and Meyer," *SIAM J. Matrix Analysis and Applications,* vol. 27, pp. 952-967, 2006.

[36] I.C.F. Ipsen and T.M. Selee, "PageRank Computation, with Special Attention to Dangling Nodes," *SIAM J. Matrix Analysis and Applications,* vol. 29, pp. 1281-1296, 2007.

[37] I.C.F. Ipsen and R.S. Wills, "Mathematical Properties and Analysis of Google's PageRank," *Bol. Soc. Exp. May. Apl.,* vol. 34, pp. 191-196, 2006.

[38] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub, "Extrapolation Methods for Accelerating PageRank Computations," *Proc. 12th Int'l World Wide Web Conf.,* pp. 261-270, 2003.

[39] S. Kamvar, T. Haveliwala, and G. Golub, "Adaptive Methods for Computation of PageRank," *Proc. Int'l Conf. Numerical Solution of Markov Chains,* 2003.

[40] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub, "Exploiting the Block Structure of the Web for Computing PageRank," technical report, Stanford Univ., 2003.

[41] C. Kohlschtter, R. Chirita, and W. Nejdl, "Efficient Parallel Computation of PageRank," *Proc. 28th European Conf. IR Research (ECIR),* pp. 241-252, 2006.

[42] G. Kollias and E. Gallopoulos, "Asynchronous PageRank Computation in an Interactive Multithreading Environment," *Proc. Seminar Web Information Retrieval and Linear Algebra Algorithms,* 2007.

[43] G. Kollias, E. Gallopoulos, and D.B. Szyld, "Asynchronous Iterative Computations with Web Information Retrieval Structures: The PageRank Case," http://arxiv.org/abs/cs/0606047, 2006.

[44] A. Langville and C. Meyer, "Deeper Inside PageRank," *Internet Math.,* vol. 1, no. 3, pp. 335-380, 2005.

[45] A. Langville and C. Meyer, "A Reordering for the PageRank Problem," *SIAM J. Scientific Computing,* vol. 27, no. 6, pp. 2112-2120, 2006.

[46] A. Langville and C. Meyer, "Updating Markov Chains with an Eye on Google's PageRank," *SIAM J. Matrix Analysis and Applications,* vol. 27, no. 4, pp. 968-987, 2006.

[47] Larbin Home Page, http://larbin.sourceforge.net/index-eng.html/, 2010.

[48] C. Lee, G. Golub, and S. Zenios, "A Fast Two-Stage Algorithm for Computing PageRank," technical report, Stanford Univ., 2003.

[49] K. Avrachenkov and N. Litvak, "Decomposition of the Google PageRank and Optimal Linking Strategy," technical report, INRIA, 2004.

[50] B. Manaskasemsak and A. Rungsawang, "An Efficient Partition-Based Parallel PageRank Algorithm," *Proc. 11th Int'l Conf. Parallel and Distributed Systems,* vol. 1, pp. 257-263, 2005.

[51] G. Karypis and V. Kumar, "MeTiS: Unstrctured Graph Partitioning and Sparse Matrix Ordering System," technical report, Dept. of Computer Science, Univ. of Minnesota, 1995.

[52] F. McSherry, "A Uniform Approach to Accelerated PageRank Computation," *Proc. 14th Int'l World Wide Web (WWW) Conf.,* pp. 575-582, 2005.

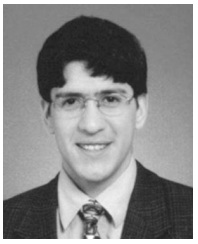
[53] S. Matsuoka, *Petascale Computing Algorithms and Applications—Chapter 14: The Road to TSUBAME and Beyond.* Chapman & Hall/CRC, pp. 289-310, 2008.

[54] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," technical report, Stanford Univ., 1999.

[55] J.X. Parreira, C. Castillo, D. Donato, S. Michel, and G. Weikum, "The Juxtaposed Approximate PageRank Method for Robust PageRank Approximation in a Peer-to-Peer Web Search Network," *The Int'l J. Very Large Data Bases,* vol. 17, pp. 291-313, 2008.
[56] A. Trifunovic and W.J. Knottenbelt, "Parkway 2.0: A Parallel Multilevel Hypergraph Partitioning Tool," *Lecture Notes in Computer Science,* pp. 789-800, Springer, 2004.
[57] B. Ucar and C. Aykanat, "A Library for Parallel Sparse Matrix-Vector Multiplies," technical Report BU-CE-0506, Dept. of Computer Eng., Bilkent Univ., 2005.
[58] B. Ucar and C. Aykanat, "Encapsulating Multiple Communication-Cost Metrics in Partitioning Sparse Rectangular Matrices for Matrix-Vector Multiplies," *SIAM J. Scientific Computing,* vol. 25, pp. 1837-1859, 2004.
[59] The Stanford WebBase Project Home Page, http://dbpubs.stanford.edu:8091/~testbed/doc2/WebBase/, 2010.
[60] Y. Wang and D.J. DeWitt, "Computing Pagerank in a Distributed Internet Search System," *Proc. 13th Int'l Conf. Very Large Data Baes (VLDB),* vol. 30, pp. 420-431, 2004.
[61] R.S. Wills and I.C.F. Ipsen, "Ordinal Ranking for Google's PageRank," *SIAM J. Matrix Analysis and Applications,* vol. 30, pp. 1677-1696, 2008.

Ali Cevahir received the BSc and MSc degrees from the Computer Engineering Department, Bilkent University, in 2004 and 2006, respectively. He is currently working toward the PhD degree at the Tokyo Institute of Technology. His research interests include parallel scientific computing and GPU computing.

Cevdet Aykanat received the BS and MS degrees from the Middle East Technical University, Ankara, Turkey, both in electrical engineering, and the PhD degree in electrical and computer engineering from Ohio State University, Columbus. He was a Fulbright scholar during the PhD studies. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Ankara, Turkey, where he is currently a professor. His research interests mainly include parallel computing, parallel scientific computing and its combinatorial aspects, parallel computer graphics applications, parallel data mining, graph and hypergraph theoretic models for load balancing, high-performance information retrieval systems, parallel and distributed databases, and grid computing. He has (co)authored about 60 technical papers published in academic journals indexed in ISI and his publications received about 400 citations in ISI indexes. He is the recipient of the 1995 Young Investigator Award of The Scientific and Technological Research Council of Turkey and the 2007 Parlar Science Award. He was appointed as a member of the IFIP Working Group 10.3 (Concurrent System Technology) in April 2004, as a member of the EU-INTAS Council of Scientists in June 2005, and as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* in December 2008.



Ata Turk received the BSc and MSc degrees from the Computer Engineering Department, Bilkent University, in 2002 and 2004, respectively. He is currently working toward the PhD degree at the Bilkent University. His research interests include parallel information retrieval and algorithms.

B. Barla Cambazoglu received the BS, MS, and PhD degrees from the Computer Engineering Department, Bilkent University, in 1997, 2000, and 2006, respectively, all in computer engineering. He then has worked as a postdoctoral researcher in the Biomedical Informatics Department, Ohio State University. He is currently a researcher in the Barcelona Lab of Yahoo! Research. He has worked in several research projects, funded by the Scientific and Technological Research Council of Turkey, the European Union Sixth and Seventh Framework Programs, and the National Cancer Institute. In 2007, he received Embodying the Vision Award as a developer of the caBIG project. His research interests mainly include information retrieval, machine learning, distributed computing, and algorithms. He is the author or coauthor of various papers published in prestigious journals including the *IEEE TPDS*, *JPDC*, and *Information Systems* as well as top IR conferences, such as the SIGIR, the WWW, and the WSDM.