WEB-SITE-BASED PARTITIONING TECHNIQUES FOR EFFICIENT PARALLELIZATION OF THE PAGERANK COMPUTATION

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Ali Cevahir

September, 2006

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Cevdet Aykanat (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Tuğrul Dayar

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Mustafa Akgül

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

WEB-SITE-BASED PARTITIONING TECHNIQUES FOR EFFICIENT PARALLELIZATION OF THE PAGERANK COMPUTATION

Ali Cevahir
M.S. in Computer Engineering
Supervisor: Prof. Dr. Cevdet Aykanat
September, 2006

Web search engines use ranking techniques to order Web pages in query results. PageRank is an important technique, which orders Web pages according to the linkage structure of the Web. The efficiency of the PageRank computation is important since the constantly evolving nature of the Web requires this computation to be repeated many times. PageRank computation includes repeated iterative sparse matrix-vector multiplications. Due to the enormous size of the Web matrix to be multiplied, PageRank computations are usually carried out on parallel systems. However, efficiently parallelizing PageRank is not an easy task, because of the irregular sparsity pattern of the Web matrix. Graph and hypergraph-partitioning-based techniques are widely used for efficiently parallelizing matrix-vector multiplications. Recently, a hypergraph-partitioning-based decomposition technique for fast parallel computation of PageRank is proposed. This technique aims to minimize the communication overhead of the parallel matrix-vector multiplication. However, the proposed technique has a high prepropocessing time, which makes the technique impractical. In this work, we propose 1D (rowwise and columnwise) and 2D (fine-grain and checkerboard) decomposition models using web-site-based graph and hypergraph-partitioning techniques. Proposed models minimize the communication overhead of the parallel PageRank computations with a reasonable preprocessing time. The models encapsulate not only the matrix-vector multiplication, but the overall iterative algorithm. Conducted experiments show that the proposed models achieve fast PageRank computation with low preprocessing time, compared with those in the literature.

Keywords: PageRank, Parallel Sparse-Matrix Vector Multiplication, Graph and Hypergraph Partitioning.

ÖZET

SAYFADEĞERİ HESAPLAMASININ ETKİN OLARAK PARALELLEŞTİRİLMESİ İÇİN AĞ SİTESİ TABANLI BÖLÜMLEME YÖNTEMLERİ

Ali Cevahir
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Cevdet Aykanat
Eylül, 2006

Ağ arama motorları, sorgu sonucunda gelen sayfaları sıralamak için birtakım sıralama yöntemleri uygular. SayfaDeğeri, ağ sayfalarını Ağ'ın bağ yapısına göre sıraya koyan önemli bir yöntemdir. SayfaDeğeri hesaplamasının etkin olması önemlidir, çünkü Ağ'ın sürekli değişen doğası bu hesaplamanın sıklıkla tekrarlanmasını gerektirir. SayfaDeğeri hesaplaması tekrarlayan seyrek matris-vektör çarpımları içerir. Matris-vektör çarpımı, SayfaDeğeri hesaplamasının anahtar işlemidir. Çarpılan matrisin çok büyük olmasından dolayı SayfaDeğeri genellikle paralel sistemlerde hesaplanır. Fakat bu çok büyük matrisin düzensiz yapısından dolayı SayfaDeğeri hesaplamasının verimli bir şekilde paralelleştirilmesi kolay bir iş değildir. Çizge ve hiperçizge bölümleme yöntemleri matris-vektör çarpımlarını etkin olarak paralelleştirilmesinde sıkça kullanılan yöntemlerdir. Yakın zamanda matris-vektör çarpımından kaynaklanan haberleşme yükünü azaltarak hızlı paralel SayfaDeğeri hesaplamak için hiperçizge bölümleme tabanlı bir yöntem öne sürülmüştür. Fakat sunulan yöntem yüksek ön işleme zamanı gerektirir. Bu da yöntemi sürekli değişen Ağ için pratikte elverişsiz kılar. Bu çalışmada, makul bir ön işlemeyle paralel SayfaDeğeri hesaplamasının haberleşme yükünü azaltacak Ağ sitesi tabanlı çizge ve hiperçizge bölümleme modelleri sunuyoruz. Sunduğumuz modeller tek boyutlu (satır sıralı ve sütun sıralı) ve iki boyutlu (ince taneli ve dama tahtası) bölümleme modelleridir. Modeller sadece matris-vektör çarpımını kapsamakla kalmayıp, bütün dolaylı algoritmayı kapsar. Yürütülen deneyler, sunulan modellerin, şu ana kadar yapılan çalışmalarla kıyaslandığında, düşük bir ön işleme zamanıyla beraber hızlı SayfaDeğeri hesaplamasını başardığını göz önüne koyar.

Anahtar sözcükler: SayfaDeğeri, Paralel Seyrek Matris-Vektör Çarpımı, Çizge ve Hiperçizge Bölümleme.

Acknowledgement

I thank to my advisor Prof. Dr. Cevdet Aykanat for his invaluable helps, guidance and motivation throughout my MS study. It has always been a pleasure to study with him. Algorithmic and theoretic contributions of this thesis is originated from his precious comments and ideas.

I would like to thank Assoc. Dr. Tuğrul Dayar and Assoc. Dr. Mustafa Akgül for reading and commenting on my thesis.

I am thankful to Ata Türk and Berkant Barla Cambazoğlu for their contributions in development of this thesis. I also thank to Bora Uçar for his support, especially in ParMxVLib library and Evren Karaca for his technical support for the parallel system used in experiments.

I thank to Ahmet, Ata, Gökhan, Hakkı, Hasan, Kamer, Muhammet, Murat, Mustafa, Osman, Serkan, Tahir and my other friends whose names I have not mentioned, for their moral support.

I would like to express my deepest gratitute to my family for their persistent support, understanding, love and for always being by my side. They make me feel stronger. I am incomplete without them.

I acknowledge the Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting my MSc studies under MSc Fellowship Program.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Wide Web (WWW) is one of the most important achievements in computer technology. It is a dynamic global library which includes vast amount of information. There are billions of Web pages (pages) in this library, which are linked by *hyperlinks*. In order to reach necessary information in this library, the existence of index searching systems is inevitable. Such searching systems are called *search engines*. Because of the huge size of the WWW, search queries may result with thousands of pages, most of which may be irrelevant to the query or less important. It is unwise to force Internet users extract relevant information within thousands of pages. Hence it is important to bring more relevant pages on top of the query results. State-of-the-art search engines, such as Google™, use ranking techniques to list query results in the order of relevance to the query. A ranking among the query results may be achieved through a combination of an analysis of the page contents and the hyperlink structure existing within the WWW. This structure is called the *Web graph*.

PageRank computation is one of the most effective and widely used query-independent way of ranking pages by utilizing the Web graph information. PageRank is first introduced by Google's founders Page and Brin at Stanford University [14]. Google claims that the heart of their software is PageRank [3]. In PageRank, every incoming link to a page contributes to its rank value. PageRank also takes the rank value of referring pages into account.

PageRank is an iterative computation, based on a random surfer model [14]. Many researchers proposed different acceleration techniques after the proposal of the basic model. Algorithmic/numeric optimizations that try to reduce the number of iterations [30, 31, 38] and I/O efficient out-of-core algorithms for reducing the disk swapping time for single processor [22, 26] are some of the proposed techniques for improving the PageRank computation performance. We will provide the background information about the basics of PageRank algorithm and some important improvement techniques in Section 2.

PageRank should be calculated repeatedly with the change of the Web graph. Unfortunately, computing PageRank is not an easy task for billions and even millions of pages. It is expensive in both time and space. Hence, it is inevitable to use efficient parallel algorithms for PageRank calculation. Various approaches are proposed on parallel and distributed PageRank computations [21, 43, 46, 54]. Some issues related to parallelization of PageRank and some of the techniques in the current literature are presented in Section 3.

PageRank algorithms iteratively solve the eigenvector of a linear system. The core operation of PageRank calculation is sparse matrix-vector multiplication. The matrix in this multiplication is called the *transition matrix* which is a matrix representation of the Web-graph. Since each page in the Web links only a small subset of the Web, Web graph is sparse, hence the transition matrix. In order to efficiently parallelize PageRank computation, efficient parallelization of matrix-vector multiplication is necessary. Various hypergraph-partitioning-based (HP-based) models [17, 51] are effective for workload partitioning of parallel sparse matrix-vector multiplications, correctly encapsulating total inter-processor communication volume. Graph-partitioning-based (GP-based) models which can also be used for workload partitioning of parallel sparse matrix-vector multiplications, try to minimize wrong metric for total communication volume. In Section 4, we give the definition of the GP and HP problems and some background information about GP and HP. Then, we review hypergraph-partitioning-based models for parallel matrix-vector multiplications in this section.

Recently there is a study which utilize the HP-based models directly for parallel PageRank computation [13]. Unfortunately, because of the huge size of the Web graph, HP-based models are not scalable, when applied directly over the Web matrix. Even though the computations reported in [13] are fairly fast; the preprocessing time for partitioning takes even longer than the sequential PageRank computation. To avoid this problem, we suggest site-based graph and hypergraph partitioning models which reduces the sizes of the graphs and hypergraphs used in partitioning, considerably. In addition to reduced preprocessing time, we consider the overall iterative algorithm including the linear vector operations and norm operations as well as matrix-vector multiplications for load balancing in the partitioning model, whereas [13] only consider matrix-vector multiplies. Furthermore, our models correctly handles pages without in-links to reduce communication overhead and obtain better imbalance. One more contribution in this work is we propose 2D-Checkerboard partitioning model and 1D columnwise model in addition to 1D rowwise and 2D fine-grain partitioning models which were analyzed in [13] as well.

As anticipated, experimental results on different datasets indicate that our site-based approach reduces the preprocessing time drastically. In some cases, site-based approach also reduces the parallel computation time by reducing the communication volume. Also, the newly applied 1D columnwise model reduce the communication volume better than the 1D rowwise model and the 2D checkerboard model minimizes the number of communications which may be significant for large number of processors. In Section 6, dataset properties are explained, experimental results regarding the proposed models are presented and trade-offs between the models are discussed.

Our methods for computing PageRank provide faster computation time with little preprocessing time when compared to available methods in current literature. However, there is still room for improvement. In Section 7, we make a conclusion and make a discussion on some future work for further improvements.

Chapter 2

Background

Following the first proposal of the PageRank in 1998 [14], numerous works concerning methods for optimization of the basic model were published. We explain the basics of the PageRank computation in Section 2.1 and survey the literature on improvements of PageRank algorithm in Section 2.2.

2.1 Basics of the PageRank

PageRank basically assigns authority scores for each page, independent of the page content. The ranking is discovered from the topological structure of the Web. The idea of PageRank was taken from academic citation literature. A hyperlink from page i to page j is counted as a vote from page i to page j. Every vote does not have the same score. The importance of a vote is proportional to the score of referring page i and inversely proportional to the total number of votes (i.e., hyperlinks) originated from page i.

PageRank can be explained with another probabilistic model, called *random surfer model* [14]. Consider an Internet user randomly visiting pages by following hyperlinks within pages or typing a random URL to the browser. Let the surfer visit page i at any step. On the next click, the probability of the surfer to visit the

page j, which is referred by page i, is proportional to the probability of visiting i and inversely proportional to total number of hyperlinks in page i. Let there are n pages in total. When the surfer gets bored with following hyperlinks, every page can be visited with a probability of $\frac{1}{n}$ assuming uniform probability distribution for visiting a random page. The probability of following hyperlinks is d, which is called *damping factor*, and visiting a random page is 1-d. Damping factor is a constant that is usually chosen between 0.85 and 0.99.

We can now give the basic PageRank formulation after explanations of above intuitive models. PageRank values can be iteratively calculated as follows. At iteration k, rank R_i^k of page i is

$$R_i^k = \frac{(1-d)}{n} + d \sum_{j \in \mathcal{N}_i} \frac{R_j^{k-1}}{deg(j)}, \tag{2.1}$$

where d is the damping factor, $\mathcal{N}_i$ is the set of pages, that contain outgoing links to page i, and $deg(j)$ is the number of outgoing links of page j.

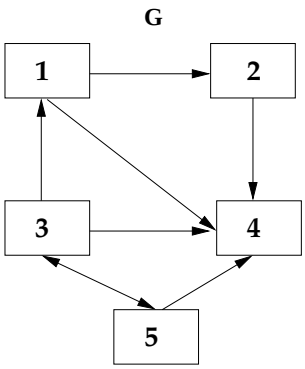
Actually, PageRank algorithm solves eigenvector of the adjacency matrix representation of the Web-graph. Equation 2.1 can be rewritten in terms of matrix-vector multiplication as

$$\mathbf{p}^k = \frac{(1-d)\mathbf{e}}{n} + d\mathbf{A} \cdot \mathbf{p}^{k-1}, \tag{2.2}$$

where $\mathbf{p}^k$ is the vector of PageRank scores of size n. Matrix $\mathbf{A}$ is called *transition matrix* and $\mathbf{A} = \mathbf{P^T}$, where $\mathbf{P}$ is the adjacency matrix representation of the Web-graph. d is the damping factor as explained above and $\mathbf{e}$ is a vector of all 1's with size n. The adjacency matrix $\mathbf{P}$ contains entries:

$$\mathbf{P}_{ij} = \begin{cases} \frac{1}{deg(i)} & \text{if there is a link from page i to page j} \\ 0 & \text{otherwise} \end{cases}$$

Figure 2.1 shows the adjacency matrix representation of a sample Web-graph.



P

0	1/2	0	1/2	0
0	0	0	1	0
1/3	0	0	1/3	1/3
0	0	0	0	0
0	0	1/2	0	1/2

Figure 2.1: Sample Web-graph and its adjacency matrix representation

The PageRank vector $\mathbf{p}$ converges to a stationary solution after repeated iterations, provided that $d < 1$. In practice, $\mathbf{p}^0$ is chosen to be a vector of size n and all entries are equal to $\frac{1}{n}$. Actually, the basic model explained by Google [14] and defined in Equation 2.2 is the Jacobi algorithm for solving linear systems [12].

The pages without outgoing links are called *dangling pages*. In order to find a probability distribution, dangling pages need special attention in PageRank computation, since they cause existence of zero-columns in transition matrix [11]. Some efforts to handle dangling pages include simply getting rid of them [14], getting rid of them but considering them in final iterations [32] and adding an ideal sink to the Web-graph which all pages points to it [12]. The most popular approach is to uniformly connect dangling pages to all pages in the graph [31, 39, 45]. The latter approach requires a great change in the Web-graph. As a result, adjacency matrix changes and becomes much denser. Fortunately, the original sparse transition matrix can be used instead of the denser matrix for computing PageRank. A power method solution for PageRank computation is presented in Figure 2.1 [31]. $\mathbf{v}$ is the *personalization* vector which contains the probabilities of visiting pages without following hyperlinks. Since $\mathbf{v}$ contains entries of a probability distribution, the sum of the entries in $\mathbf{v}$ is equal to 1.

PageRank($\mathbf{A}$, $\mathbf{v}$)
  $\mathbf{p} \leftarrow \mathbf{v}$
  repeat
    $\mathbf{q} \leftarrow \mathrm{d}\mathbf{A}\mathbf{p}$
    $\gamma \leftarrow ||\mathbf{p}||_1 - ||\mathbf{q}||_1$
    $\mathbf{q} \leftarrow \mathbf{q} + \gamma\mathbf{v}$
    $\delta \leftarrow ||\mathbf{q} - \mathbf{p}||_1$
    $\mathbf{p} \leftarrow \mathbf{q}$
  until $\delta < \varepsilon$
  return p

Figure 2.2: Power method solution for PageRank

2.2 PageRank Algorithms for Faster Computation

Iterative algorithms are used for computing PageRank. Hence, there are two considerations in faster computing of PageRank: reducing per-iteration time and reducing number of iterations.

One of the most widely used methods to compute PageRank is the power method. There are several reasons for popularity of the power method, despite its slow convergence. First of all, it is a simple method which makes computation on a sparse matrix, rather than a dense matrix. The original sparse transition matrix can be used in power method while handling dangling pages (see Figure 2.1). This ensures faster per-iteration time. Power method requires less memory with a few number of vectors and a sparse transition matrix.

The BlockRank algorithm proposed by Kamvar et al. [32] use pre-computed values for $\mathbf{p}^0$ instead of $\mathbf{v}$. BlockRank algorithm utilizes the block structure of the Web for faster convergence of PageRank vector. In the WWW, most of the links within pages are to the pages within the same host (site). Authors observe that more than 80% of the hyperlinks in the Web are intra-site links. This means that the rank of a page is mostly determined by the links from the same site. Considering this fact, BlockRank algorithm first computes a ranking

between Web-sites, called HostRank and use this ranking as initial approximation for PageRank. Computing HostRank is much faster than computing PageRank, since the host graph is much smaller than Web-page graph.

Other iterative methods for computing PageRank such as GMRES, BiCG and BiCGSTAB are discussed in [23]. We will discuss this work in Chapter 3. Iterative aggregation is applied to PageRank in [37]. Directed acyclic graph structure of the Web is utilized in [8] for faster convergence of PageRank computation. Adaptive methods which utilize quick convergence of most of the Web are introduced in [30].

The huge size of the Web-graph emerges I/O efficient techniques for sequential calculations of PageRank. These techniques mostly aim to reduce per-iteration time rather than the number of iterations. Efficient encoding techniques are presented in [27]. Block-based disk access approach is considered in [26]. Based on the work in [26], Chen et al. introduce I/O efficient techniques in [22].

In literature, some less popular alternative methods are introduced for ranking Web-pages, such as HITS (Hyperlink-Induced Topic Search) [36] and SALSA (Stochastic Approach for Link Structure Analysis) [40, 41]. These two methods iteratively compute two scores for each page, namely *hubs* and *authorities*, and runs at query time on the query sub-graph of the Web.

Chapter 3

Parallel PageRank

PageRank citation ranking is rather a simple problem which can be solved using old methods. What makes it extremely difficult is the size of the problem. PageRank computation is claimed to be "the World's largest matrix computation" by Cleve Moler [44]. Most of the time, the matrix to be computed, as a whole, is too large to be stored in the main memory of a single machine. Only the PageRank vector itself with a billion entries requires 4 gigabytes of main memory. This fact enforces researchers to discover efficient parallel and distributed algorithms. However, it is not easy to come up with an efficient parallel algorithm because of the issues related to parallel PageRank computation, which will be discussed in Section 3.1. Some attempts on parallelizing PageRank in current literature are explained in Section 3.2.

3.1 Issues in Parallel PageRank Computation

As explained in Section 2.2, PageRank can be computed by various algorithms, which have many common features. In order to efficiently parallelize any PageRank algorithm, there are several issues to be considered. Some of the important issues can be listed as follows:

- Irregularly sparse and huge Web adjacency matrices
- Load balancing
- High communication volume
- Fine-grain computations
- Handling zero-rows in transition matrix **A** (rank computations of pages without incoming links)

Sparse matrix-vector multiplication is the core operation in PageRank computations. Several state-of-the-art techniques are recently proposed for efficient parallelization of sparse matrix-vector multiplications [17, 19, 51]. However, it is not very practical to directly apply this techniques to enormous size Web-matrices due to the space limitations and high preprocessing time required. On the other hand, because of the irregularity in the sparse transition matrix, it is not easy to find intelligent distribution of the matrix and the PageRank vector to processors without applying state-of-the-art techniques.

Load balancing in PageRank computation corresponds to balancing computational load for each processor. This computational load is mostly originated from matrix-vector multiplication. In order to balance computation in matrix-vector multiplication, non-zeros of the matrix should be evenly distributed among the processors. Furthermore, linear vector operations and norm operations should also be considered for load balancing of PageRank computation. Without considering communication overhead, load balancing is relative easy to be provided. However, considering only load balancing in partitions may lead to high communication volume. High communication volume slows down the parallel execution time seriously, since there is a fine-grain computation between two consecutive communication phases.

Another consideration of parallel PageRank computation is handling zero-rows (rows without non-zeros) in transition matrix. This problem is specific to the PageRank problem. In most of the matrix-vector multiplication applications,

there exists no such rows. This problem originates from the pages without incoming links in the Web-graph. Those pages are different from dangling pages (see Section 2.2). Note that, we do not need to have information about other pages' ranking scores for computing PageRank values of those pages, since they have no incoming link. For this reason, in parallel implementations of PageRank algorithms, those pages may be utilized for reducing communication volume and obtaining better load balancing. To the best of our knowledge, in current literature, there has been no effort for handling zero-rows in parallelization of the PageRank. We will explain how we handle zero-rows in Chapter 5.

3.2 Survey on Parallel and Distributed Algorithms

There are mainly two types of approaches in parallelization of PageRank: distributed approximation algorithms and massively parallel implementations of original algorithms. Approximate methods target to reduce the number of iterations by some numerical techniques, allowing tolerable errors in ranking. Latter approach tries to implement sequential algorithms in parallel without allowing any numerical differences in PageRank vector.

Wang and DeWitt [53] propose a distributed algorithm for an architecture where every Web-server answers queries on its own data. Algorithm, basically, first computes PageRank values for each server and by exchanging links across servers adjusts local PageRank values. Local PageRank is computed by the power method. ServerRank concept is proposed in this work, which corresponds to a ranking among servers. ServerRank is also computed by the power method. Then, using local PageRank and ServerRank information together with inter-server link information, local PageRank values are approximated. This approximate, architecture dependent method does not have any strategy to reduce the communication volume while exchanging inter-server links.

Another distributed PageRank algorithm applies iterative aggregation-disaggregation methods for reducing the number of iterations [54]. This work is similar to [53] in the point that they both first compute local PageRank vectors, then approximate the result using inter-server link information. Iterative aggregation-disaggregation method reduces the number of iterations, but it increases per-iteration time, since each iteration requires several matrix-vector multiplications. At the end of each iteration, local PageRank values are sent to global coordinator to check for convergence.

In 2002, Gürdağ [25] parallelized PageRank and experimented the performance on a distributed memory PC cluster with different network interfaces. In this work, although the author explains the graph partitioning based approach, because of the high preprocessing overhead and high imbalance, Gürdağ refrains implementing graph partitioning based parallel PageRank. Unfortunately, there is no special attempt to reduce the communication volume; instead, pages are simply assigned to the processors using uniform block partitioning. That is first $\frac{n}{p}$ rows are assigned to processor 1, second $\frac{n}{p}$ rows are assigned to processor 2 and so on, where n is the total number of rows and p is the number of processors. Another deficiency of this work is, although there are lots of experiments and performance comparisons with different settings, the experiments are carried out with randomly generated Web graphs, instead of real datasets.

In a similar work, Manaskasemsak and Rungsawang propose a massively parallel PageRank algorithm to be run on gigabit PC clusters [43]. The authors parallelize the basic PageRank algorithm proposed by Brin and Page [14]. The algorithm suffers from the high communication volume, since transition matrix is distributed to processors using uniform block partitioning. The algorithm requires communication of all inter-processor PageRank scores. To overcome this problem, authors suggest to do communication for every x iterations. However, this approach results with error in computation of rank values.

Another parallelization effort of PageRank is explained in [23]. As in [43], authors do not try to minimize the communication volume. They refrain applying graph partitioning based models for PageRank computation, claiming that

graph partitioning does not perform well for huge, power-law Web data. They parallelize PageRank using different iterative methods such as Jacobi, GMRES, BiCG and BiCGSTAB. This work is important in the sense that it shows the relative performances of different iterative methods for computing PageRank, such as number of iterations and per-iteration times.

Chapter 4

Graph Theoretical Models for Sparse Matrix Partitioning

Graph and hypergraph partitioning based models are widely used to obtain better matrix decompositions for parallel matrix-vector multiplications. Both models are used as a preprocessing to minimize inter-processor communication volume while obtaining load balance. Graph partitioning (GP) based models minimize wrong metric for communication [17, 29], whereas hypergraph partitioning (HP) based models correctly encapsulate the minimization of communication volume, hence find better quality solutions (see Section 4.4).

This chapter aims to give some background information on GP and HP. We review applications of HP-based models to PageRank problem at the end of this chapter. We explain GP and HP problems in Sections 4.1 and 4.2, respectively. Widely used multilevel paradigm to find a solution to GP and HP is explained in Section 4.3. HP-based models for parallelization of matrix-vector multiplication are reviewed in Section 4.4. A recently proposed work on HP-based models for PageRank computation is reviewed in Section 4.5.

4.1 Graph Partitioning Problem

An undirected graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ is defined as a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. Each edge $e_{ij}\in\mathcal{E}$ connects two distinct vertices v_i and v_j in $\mathcal{V}$. Weight w_i or multiple weights $w_i^1, w_i^2, \ldots, w_i^M$ may be associated with a vertex $v_i \in \mathcal{V}$. c_{ij} is called as the cost of the edge $e_{ij}\in\mathcal{E}$.

$\Pi=\{\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_K\}$ is said to be a K-way partition of $\mathcal{G}$ where each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, parts are pairwise disjoint, and the union of the K parts is equal to $\mathcal{V}$. For each part $\mathcal{V}_k \in \mathcal{E}$, a *balanced* partition Π satisfies the balance criteria

$$W_k^m \leq (1+\epsilon)W_{\text{avg}}^m, \qquad \text{for } k=1,2,\ldots,K \text{ and } m=1,2,\ldots,M. \tag{4.1}$$

In Equation 4.1, each weight W_k^m of a part $\mathcal{V}_k$ is defined as the sum of the weights w_i^m of the vertices in that part. W_{avg}^m is the average weight of all parts. ϵ is the maximum imbalance ratio allowed.

In a partition Π of $\mathcal{G}$, an edge is *cut* if its vertices are in different parts, *uncut* otherwise. The cutsize which represents the cost $\chi(\Pi)$ of a partition Π is

$$\chi(\Pi) = \sum_{e_{ij}\in\mathcal{E}} c_{ij} \tag{4.2}$$

A K-way graph partitioning problem is dividing a graph into K parts such that cutsize is minimized (Equation 4.2) while obtaining the balance on weights (Equation 4.1). This problem is known to be NP-hard. If multiple weights are associated with vertices (i.e. $M > 1$ in Equation 4.1), the problem is called *multiconstraint graph partitioning.*

4.2 Hypergraph Partitioning Problem

A hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$. Each net $n_j \in \mathcal{N}$ is a subset of vertices in $\mathcal{V}$. The vertices of a net n_j are called as the pins of net n_j. Weight w_i or multiple weights $w_i^1, w_i^2, \ldots, w_i^M$ may be associated with a vertex $v_i \in \mathcal{V}$. c_j is called as the cost of the net $n_j \in \mathcal{N}$.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is said to be a K-way partition of $\mathcal{H}$ where each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, parts are pairwise disjoint, and the union of the K parts is equal to $\mathcal{V}$. For each part $\mathcal{V}_k \in \mathcal{N}$, a *balanced* partition Π satisfies the balance criteria

$$W_k^m \leq (1+\epsilon)W_{\text{avg}}^m, \qquad \text{for } k=1,2,\ldots,K \text{ and } m=1,2,\ldots,M. \tag{4.3}$$

In Equation 4.3, each weight W_k^m of a part $\mathcal{V}_k$ is defined as the sum of the weights w_i^m of the vertices in that part. W_{avg}^m is the average weight of all parts. ϵ is the maximum imbalance ratio allowed.

In a partition Π of $\mathcal{H}$, a net *connects* a part, if it has at least one pin in that part. *Connectivity set* Λ_j of a net n_j is the set of parts that the net n_j connects. Connectivity $\lambda_j = |\Lambda_j|$ of a net n_j is the number of parts connected by n_j. A net is *cut* if it connects more than one part ($\lambda_j > 1$), *uncut* otherwise. Cut-nets are called external nets and uncut-nets are called internal nets of the parts that they connect.

A K-way hypergraph partitioning problem is dividing a hypergraph into K parts such that a partitioning objective defined over the nets is minimized while obtaining the balance on weights (Equation 4.3). This problem is known to be NP-hard [42]. If multiple weights are associated with vertices (i.e. $M > 1$ in Equation 4.3), the problem is called *multiconstraint hypergraph partitioning*.

The objective function to be minimized defined over the nets are called *cutsize*. Most widely used partitioning objective is called *connectivity*-1 cutsize metric:

$$\chi(\Pi) = \sum_{n_i \in \mathcal{N}} c_i(\lambda_i - 1), \tag{4.4}$$

in which each net contributes $c_i(\lambda_i - 1)$ to the cost $\chi(\Pi)$ of a partition Π. This cutsize metric is widely used in VLSI [42] and sparse matrix community [17, 19, 51]. In this work, we will concentrate on minimizing this definition of cutsize.

Another definition of cutsize is called cut-net metric and defined as

$$\chi(\Pi) = \sum_{n_i \in \mathcal{N}} c_i, \tag{4.5}$$

in which the cutsize is equal to the sum of costs of the cut-nets. This objective is less used than the connectivity-1 metric, however there are cases in which this metric is useful [10].

4.3 Multilevel Paradigm for Graph and Hypergraph Partitioning

Heuristic methods are used for partitioning graphs and hypergraphs, since GP and HP problems are NP-complete. The most widely used technique is multilevel partitioning paradigm [15]. Multilevel approach consists of three phases: coarsening, initial partitioning, uncoarsening with refinement.

In the coarsening phase, vertices are visited in a random order and matched or clustered by a given vertex matching criteria. Vertices might be matched according to the number of nets connecting them, various distance definitions between them or randomly. Matched vertices form a super vertex of the next level. The weight of a super vertex is the sum of the weights of vertices that form the super vertex. This matching continues until the number of vertices falls below a predetermined threshold.

The coarsest graph or hypergraph is easy to be partitioned. In the initial partitioning phase, the coarsest graph or hypergraph is partitioned by one of the various heuristics.

After the initial partitioning phase, the quality of partitioning by means of cutsize and imbalance may be improved. The last phase of the multilevel partitioning is uncoarsening. In this phase, the coarsest graph or hypergraph is uncoarsened in the reverse direction of the coarsening phase. During each uncoarsening level, cutsize is refined using one of the various heuristics. This refinement is realized by trying to change partitions of the vertices so that the cutsize across the partitions are reduced.

There are two main approaches to K-way partition a graph or hypergraph using multilevel paradigm: recursive bisection and direct K-way partitioning. Recursive bisection approach recursively partitions a graph or hypergraph into two until K-partitions are generated. Direct K-way partitioning applies multilevel phases once and directly generates K-partitions. Figure 4.1 depicts direct K-way approach to multilevel paradigm for graph and hypergraph partitioning. Multilevel paradigm enabled implementations of powerful graph and hypergraph partitioning tools [9, 16, 18, 33, 34, 35, 47].

4.4 Hypergraph Partitioning Models for Parallel Matrix-Vector Multiplication

For efficient parallelization of repetitive matrix vector multiplications ($\mathbf{y} = \mathbf{Ax}$), it is required to evenly distribute non-zeros in the sparse matrix to processors while achieving small amount of communication. This is a hard problem which requires a permutation of rows and columns of the matrix $\mathbf{A}$. To find a good permutation, several HP-based models are proposed in the literature [17, 19, 20].

Hypergraph models are used to assign non-zero entries of matrix $\mathbf{A}$ together with vectors $\mathbf{x}$ and $\mathbf{y}$ to processors. If vectors $\mathbf{x}$ and $\mathbf{y}$ undergo linear vector

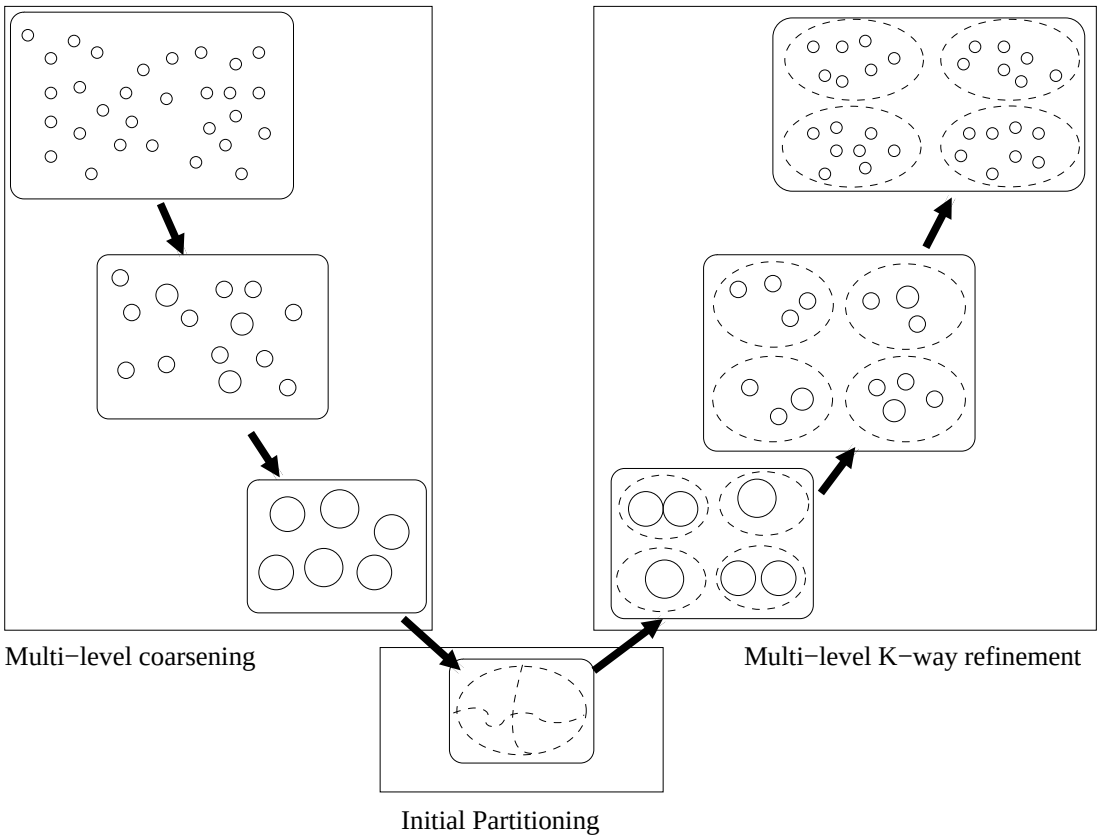
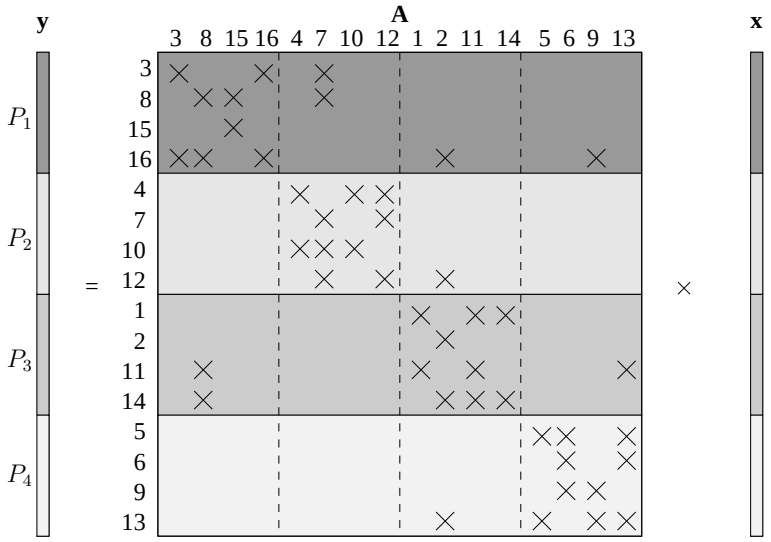


Figure 4.1: 4-way multilevel graph or hypergraph partitioning [16]

operations in an iterative method (which is the case in PageRank computation), than each processor should obtain same portions of input vector $\mathbf{x}$ and output vector $\mathbf{y}$. This type of partitioning is called *symmetric* partitioning.

In this section, we are going to explain four HP-based models for different parallelization techniques of matrix vector multiplications. Rowwise and columnwise decomposition techniques are referred as 1D partitioning, in which matrix $\mathbf{A}$ is assigned to processors according to its rows or columns, respectively. In 2D fine-grain partitioning, non-zeros are individually assigned to processors. 2D checkerboard partitioning, as the name implies, is an assignment of non-zero entries to the processors in a checkerboard pattern.



Shaded areas indicate assignment for the matrix and vectors to processors. For example, rows 3, 8, 15 and 16 are assigned to one processor together with corresponding $\mathbf{x}$ and $\mathbf{y}$ vector entries.

Figure 4.2: Rowwise partitioning of $\mathbf{y}=\mathbf{A}\mathbf{x}$.

4.4.1 1D Rowwise Partitioning

In row-parallel multiplication $\mathbf{y}=\mathbf{A}\mathbf{x}$, each processor is responsible for multiplication of different rows. Rowwise partitioning assigns rows of the matrix and vector entries to processors (see Figure 4.2). In this scheme, if row j is assigned to processor P_i, then P_i is responsible for calculating $\mathbf{y}_j$. To calculate $\mathbf{y}_j$, P_i should have the $\mathbf{x}$ vector entries that correspond to the non-zeros in row j. Otherwise, the $\mathbf{x}$ vector entry should be requested from the processor that holds this entry.

Figure 4.2 depicts a symmetric partitioning on input and output vectors. For computing $\mathbf{y}_{16}$, P_1 should have $\mathbf{x}_3$, $\mathbf{x}_8$, $\mathbf{x}_{16}$, $\mathbf{x}_2$ and $\mathbf{x}_9$. $\mathbf{x}$ vector entries 3, 8 and 16 is already assigned to P_1. However, for calculating $\mathbf{y}_{16}$, $\mathbf{x}_2$ is requested from P_3 and $\mathbf{x}_9$ is requested from P_4. Hence, in order to reduce total communication volume, non-zeros in the matrix should be gathered to the diagonal blocks. Off-diagonal blocks incur communication. Note that, there is no dependency between processors on $\mathbf{y}$ vector entries in row-parallel multiplication, hence they are never communicated.

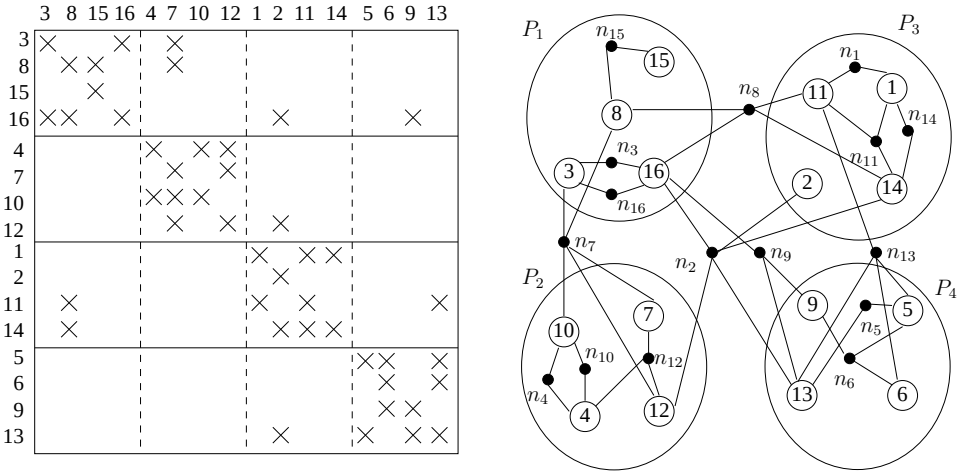


Figure 4.3: 4 way decomposition of column-net hypergraph of matrix **A**.

The *column-net* hypergraph partitioning model is proposed by Çatalyürek and Aykanat [17] for reducing total communication in row-parallel matrix-vector multiplications. In column-net model, for each row, there exists a vertex and for each column, there exists a net (see Figure 4.3). Vertices are weighted with the number of non-zeros in corresponding row, to denote the workload of that row. Each net has a unit cost. n_j contains the pin v_i, if matrix **A** contains non-zero a_{ij}. In other words, each net n_j connects non-zeros associated with $\mathbf{x}_j$ (i.e. non-zeros in column j). During multiplication, if non-zeros that are multiplied with the same **x** vector entry are not in the same processor, then the vector entry should be communicated. Hence, partitioning the column-net hypergraph while minimizing the cutsize using connectivity-1 metric, minimizes the communication volume of row-parallel matrix-vector multiplication.

4.4.2 1D Columnwise Partitioning

Column-parallel matrix-vector multiplication $\mathbf{y}=\mathbf{Ax}$ is similar to row-parallel multiplication, but in this case columns are distributed among processors (see Figure 4.4). Hence, each processor is now responsible for doing multiplications incurred from local **x** vector entries. At the end of the local multiplications, each processor contains partial **y** vector results. These results are communicated to

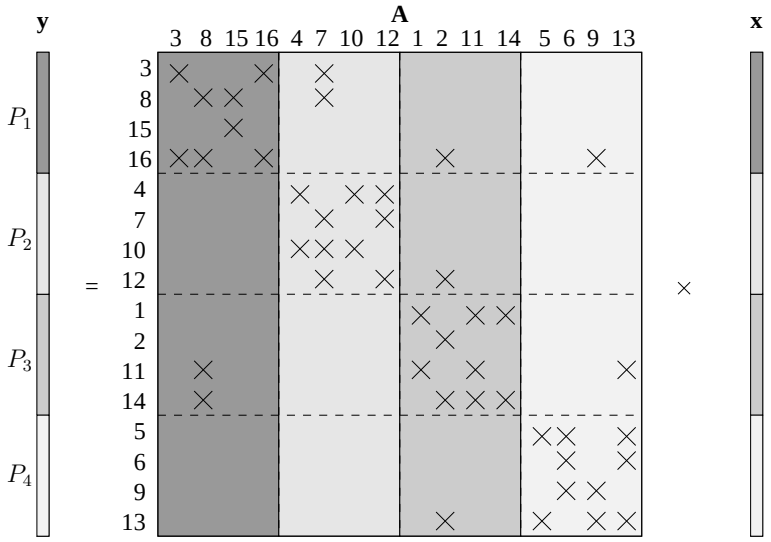


Figure 4.4: Columnwise partitioning of $\mathbf{y}=\mathbf{A}\mathbf{x}$.

compute actual $\mathbf{y}$ vector. For example, in Figure 4.4, for $\mathbf{y}_{16}$, multiplication incurred by $\mathbf{x}_3$, $\mathbf{x}_8$ and $\mathbf{x}_{16}$ done by P_1, $\mathbf{x}_2$ is done by P_3 and $\mathbf{x}_9$ is done by P_4. Partial results computed by P_3 and P_4 are sent to P_1, which is responsible for computing $\mathbf{y}_{16}$. In column-parallel multiplication, dependency between processors are on $\mathbf{y}$ vector entries, instead of $\mathbf{x}$.

The *row-net* hypergraph partitioning model is proposed by Çatalyürek and Aykanat [17] for reducing total communication in column-parallel matrix-vector multiplications. Row-net hypergraph is dual of column-net hypergraph. In a row-net hypergraph, each column of the matrix corresponds to a vertex with weight is equal to the number of non-zeros in the corresponding column and each unit-cost net corresponds to a row which contains the vertices that correspond to the non-zero columns in that row. Hence, each vertex denotes workload of a column and each net denotes dependencies on a $\mathbf{y}$ vector entry. For partitioning, connectivity-1 metric is used, since an external net n_i refers a communication of partial $\mathbf{y}$ vector results to the owner of $\mathbf{y}_i$ from the other processors which contributes the result of $\mathbf{y}_i$.

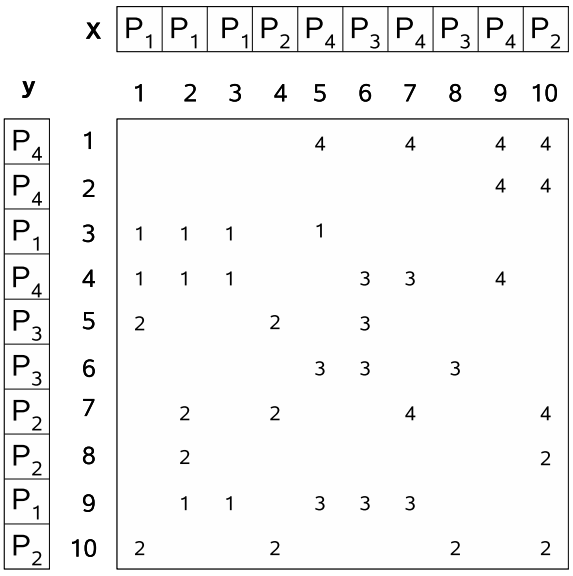


Figure 4.5: Fine-grain partitioning of $\mathbf{y}=\mathbf{A}\mathbf{x}$ [48]

4.4.3 2D Partitioning Models

2D partitioning models are applicable to row-column-parallel matrix-vector multiplications $\mathbf{y}=\mathbf{A}\mathbf{x}$. In a 2D partition of the matrix, non-zeros in a row or a column may be assigned to different processors. Hence, processors require communication for both $\mathbf{x}$ and $\mathbf{y}$ vector entries. In row-column-parallel multiplication, processors first communicate on $\mathbf{x}$ vector entries. After receiving necessary $\mathbf{x}$ vector entries, each processor can do multiplications with local non-zeros. Finally, processors communicate partial $\mathbf{y}$ vector entries to compute actual $\mathbf{y}$ vector. Figure 4.5 depicts a sample 2D partitioning. For the partitioning represented in the figure, in order to compute $\mathbf{y}_5$, P_2 receives $\mathbf{x}_1$ from P_1 and partial result $a_{51}\mathbf{x}_1 + a_{54}\mathbf{x}_4$ computed by P_2 is sent to P_3.

Minimization of total communication volume for 2D fine-grain partitioning and 2D checkerboard partitioning can be modeled by HP-based approach. Fine-grain model tries to reduce the total communication volume by reducing dependencies on $\mathbf{x}$ and $\mathbf{y}$ vector entries [19]. Checkerboard partitioning model proposed by Çatalyürek and Aykanat [20] correctly encapsulates minimization of total communication volume with smaller number of inter-processor communications.

4.4.3.1 2D Fine-Grain Partitioning

1D partitioning schemes try to assign rows or columns to processors, as a whole. According to the partitioned dimension, communication occurs before or after the local multiplication. Another approach to parallel matrix-vector multiplication may be assignment of non-zeros to processors individually, instead of coarse grain assignment of rows or columns. This type of partitioning is called *fine-grain* partitioning. 2D partitioning represented in Figure 4.5 is a fine-grain partitioning.

For fine grain decomposition of the matrix $\mathbf{A}$ in $\mathbf{y}=\mathbf{Ax}$, Çatalyürek and Aykanat [19] propose a HP-based model. The hypergraph in the model contains a vertex for each non-zero in $\mathbf{A}$ and two nets connected to each vertex. Each vertex denotes one multiplication incurred by its corresponding non-zero, therefore has unit weight. For each row and for each column, there is a net which contains vertices corresponding to the non-zeros in the row or column, indicating the dependencies of non-zeros to the $\mathbf{x}$ and $\mathbf{y}$ vector entries.

4.4.3.2 2D Checkerboard Partitioning

Checkerboard partitioning is a 2D coarse grain partitioning scheme for matrix-vector multiplication $\mathbf{y}=\mathbf{Ax}$. There are several methods for checkerboard partitioning [28]; but we will concentrate on a more elegant method proposed by Çatalyürek and Aykanat [20].

In checkerboard partitioning scheme, matrix $\mathbf{A}$ is first partitioned into r row blocks, using column-net hypergraph model for rowwise decomposition. Then, each row block is partitioned into c column blocks. As a result, matrix is partitioned into $r \times c$ parts, which naturally maps to $r \times c$ mesh of processors. c-way columnwise partitioning is applied on a row-net hypergraph with multiweights. Each vertex of row-net hypergraph has r weights, where j^{th} multiweight of vertex i (vertex corresponding to i^{th} column), w_i^j, is equal to number of non-zeros of column i in row-block j. This assignment of multiweights ensures balanced partitioning of c loads in each of r row-blocks. Partitioning column-net hypergraph

reduces the communication volume on $\mathbf{x}$ and partitioning row-net hypergraph reduces the communication volume on $\mathbf{y}$.

4.5 Page-Based Models

Matrix-vector multiplication is the kernel operation of many iterative methods. HP-based models effectively address the problem of parallelizing matrix-vector multiplication and hence is very powerful for parallelization of many iterative methods [49]. PageRank algorithms are iterative methods based on repetitive matrix-vector multiplications. With this motivation, Bradley et al. [13] utilized HP-based models for PageRank computation. We will refer their work as *page-based approach* from now on, since they partition model hypergraphs for the page-to-page transition matrix.

In their work, Bradley et al., uses the power method formulation represented in Figure 2.1 for computing PageRank. 1D rowwise and 2D fine-grain HP-based models are applied to the matrix-vector multiplication in the power method for fast parallel computing of PageRank. Parallel hypergraph partitioning tool Par$k$way [47] is used for partitioning hypergraphs. As a result, hypergraph partitioning approach reduced the per-iteration time of the PageRank computation by a factor of two, compared with the most effective approach until this work.

Despite halving the per iteration time, page-based approach has an important deficiency. Hypergraph partitioning is a preprocessing for matrix-vector multiplications. For PageRank calculation, the transition matrix to be partitioned is very big in size, which requires more time and space for partitioning. Hence, even with a parallel partitioning tool, it may take quite long time to partition such matrices. For example, the data provided by Bradley et al. reveals that in some cases, partitioning takes 10,000 times longer than one iteration. On the other hand, computing PageRank takes 80-90 iterations for their method and convergence criteria. Hence, it can only be used for calculating values of many PageRank vectors which shares the same partitioning.

Chapter 5

Site-Based Models

As discussed in the previous chapter, hypergraph-partitioning-based models for the parallelization of PageRank computations are quite successful in minimizing the total communication volume during the parallel PageRank computations. On the other hand, the preprocessing overhead incurred due to the partitioning of the page-to-page (P2P) transition matrix $\mathbf{A}$ is quite significant.

In this chapter, we investigate the ways to reduce the preprocessing overhead before the parallelization of PageRank computation without degrading the parallel computation performance. For this purpose, we propose to partition the compressed version of the transition matrix. Using the observation that Web sites form natural clustering of pages [8, 32], we compress the page-to-page transition matrix to site-to-page (S2P), page-to-site (P2S) and site-to-site (S2S) matrices and partition them employing the hypergraph and graph partitioning based models. Partitioning a compressed matrix corresponds to partitioning a coarser hypergraph or graph. Besides reducing the preprocessing time, partitioning coarser site hypergraph or graph has an advantage of finding comparable cutsize to page-based partitioning, since natural clustering of pages provide high quality coarsening. We also utilize zero-rows in the transition matrix for better load balancing and reducing the communication overhead.

In this chapter, we explain our contributions to parallel PageRank computation problem. In Section 5.1, we propose a technique to decrease the number of global synchronization points for parallelization of the power method. In Section 5.2, site-based HP models for 1D and 2D decomposition of transition matrix are introduced. In Section 5.3, graph models for 1D decomposition are explained.

5.1 Coarse-Grain Power Method

In Chapter 2, we discussed a power method formulation for computing PageRank. Power method is popular for computing PageRank, because of the low memory requirements and fast per-iteration time. In this work, our main focus is to decrease the per iteration time of PageRank algorithms. Although our approach is applicable to other alternative iterative methods, we choose power method for parallelization, since it is simple yet efficient for computing PageRank. In [23], performance of several iterative methods are presented. The cited work reveals that despite the number of iterations are less for other iterative methods, the power method has very little per-iteration time. As a result, overall completion time of power method is smaller than alternative methods.

Figure 2.1 depicts the power method that is used for computing PageRank. When the power method is parallelized in this form, there are two global synchronization points. To assign global γ and δ values, processors exchange their local γ and δ values. It is possible to compute global values for scalars γ and δ with only one all-to-all communication.

To reduce the number of global synchronization points to one, we delay computation of δ for one iteration. By doing so, processors can exchange local γ and δ values in one all-to-all communication. The coarse-grain power method for parallel PageRank computation is represented in Figure 5.1.

Parallel-PageRank($\mathbf{A}$, $\mathbf{v}$)
  $\mathbf{q} \leftarrow \mathbf{v}$
  repeat
    $\mathbf{r} \leftarrow \mathrm{d}\mathbf{A}\mathbf{q}$
    $\gamma_{local} \leftarrow \|\mathbf{q}\|_1 - \|\mathbf{r}\|_1$
    $\delta_{local} \leftarrow \|\mathbf{q} - \mathbf{p}\|_1$
    $\langle \gamma, \delta \rangle \leftarrow$ Allreduce_sum($\langle \gamma_{local}, \delta_{local} \rangle$) *//Sync. point*
    $\mathbf{r} \leftarrow \mathbf{r} + \gamma\mathbf{v}$
    $\mathbf{p} \leftarrow \mathbf{q}$
    $\mathbf{q} \leftarrow \mathbf{r}$
  until $\delta < \varepsilon$
  return q

Figure 5.1: Coarse-grain power method for parallel PageRank computation

5.2 Site-Based HP Models

In this section, we introduce several site-based HP models for faster decomposition of the transition matrix. As the original models, site-based HP models correctly capture the total communication volume.

General framework of site-based hypergraph models is as follows. Rows and columns corresponding to the pages without in-links of the transition matrix $\mathbf{A}$ are moved to the end for handling zero-rows (see Figure 5.2 for reordering of matrix $\mathbf{A}$). After reordering, the new matrix $\mathbf{B}$, which does not contain any rows and columns corresponding to the pages without in-links, is compressed. This corresponds to one level coarsening of the page hypergraph, where each supervertex maps to a site. After compression, number of vertices decreases from number of pages to number of sites. However, there is no decrease in the number of nets. To decrease the number of nets, we first eliminate nets which have no vertices or connect only one vertex, since they do not affect the cutsize of any partition. This reduces the number of nets drastically, because most of the pages include hyperlinks only to pages in the same site and most of them have no links. Remaining hypergraph has many nets which connect to same set of vertices. The number of nets are further reduced by combining identical nets together, which

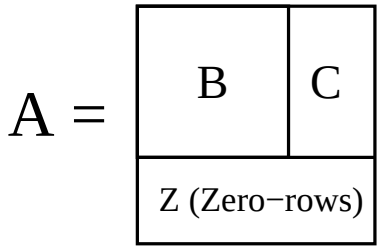


Figure 5.2: Ordering of matrix **A**

also reduces the number of nets significantly. As the last step of preprocessing, the resulting hypergraph is partitioned and part vectors for **A** are generated mapping the sites in **B** and pages without in-links to the partitions. Site-based models for each partitioning scheme are explained in detail in the following subsections.

5.2.1 Rowwise Partitioning

In this section, each step of rowwise decomposition of transition matrix **A**, that are mentioned above, is explained in detail. We use **x** and **y** to denote input and output vectors to be partitioned for matrix-vector multiplication, respectively.

5.2.1.1 Handling Pages without In-links

Zero-rows in **A** corresponds to pages that have no incoming links in Web graph $\mathcal{G}$. Therefore, their PageRank values are $\gamma\mathbf{v}$ for each iteration (see Figure 5.1). To compute their PageRank values, processors do not need to exchange their local **x** vector entries, since each processor can calculate γ and may store the value of **v**. Hypergraph partitioning naturally prevents communication on **x** for zero-rows. However, pages without in-links affect the PageRank values of the pages that they have hyperlinks to. Therefore, they require communication for computing PageRank values of the pages they link, if they are assigned to different processors. Nevertheless, this communication can be avoided, since their PageRank values, $\gamma\mathbf{v}$, can be calculated by any processor. Hence, we let HP-based model find partitions for only nonzero-rows and assign zero-rows to under-loaded processors.

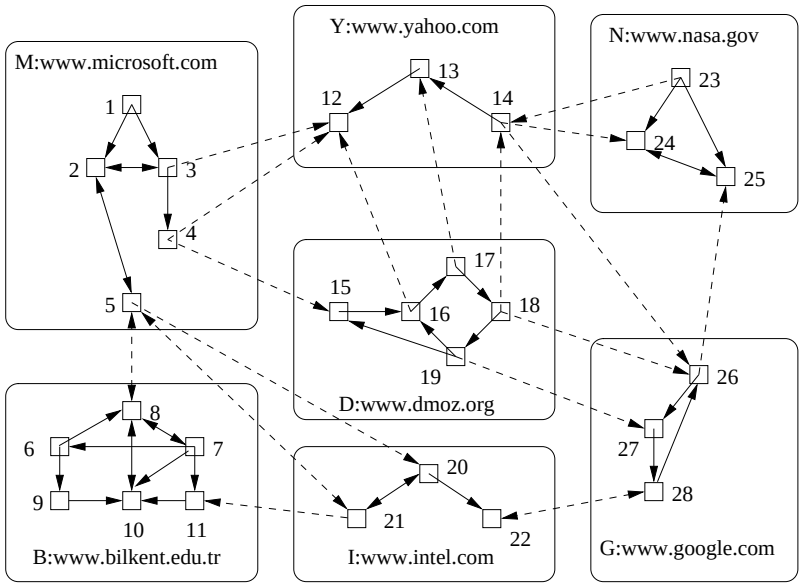


Figure 5.3: Sample Web graph

In order to exclude zero-rows from the column-net hypergraph, we reorder the transition matrix, so that rows and columns corresponding to the no-in-link pages are moved to the end. Figure 5.4 represents the ordering of the **A** matrix of the sample Web graph in Figure 5.3. In Figure 5.4, the upper left matrix bordered with thick lines, **B**, contains no zero-row. Shaded diagonal blocks inside **B** represents the intra-site links. Columns representing the links given by no-in-link pages forms the sub-matrix **C**.

5.2.1.2 Site-to-Page Compression and Identical Net Elimination

Even after zero-row elimination, the matrix **B** is still too big to be partitioned by HP-based models. To reduce the size of the hypergraph to be partitioned, **B** is compressed on its rows in linear time with the number of non-zeros in **B**. In the site-to-page (S2P) compressed matrix $\mathbf{B}^{comp}$, each row represents a site and each column represents a page. There is a non-zero in $\mathbf{B}^{comp}_{ij}$ if there is a link from site i to page j in $\mathcal{G}$. During compression, if page j belongs to site i and $\mathbf{B}^{comp}_{ij}$ does not exist, we add non-zero $\mathbf{B}^{comp}_{ij}$ to correctly encapsulate the communication volume for symmetric partitioning of **x** and **y** vectors [17]. Compression of P2P matrix to S2P matrix corresponds to coarsening the vertices of the column-net hypergraph of **B** by clustering pages into their sites. Figure 5.5 depicts compression of **B**.

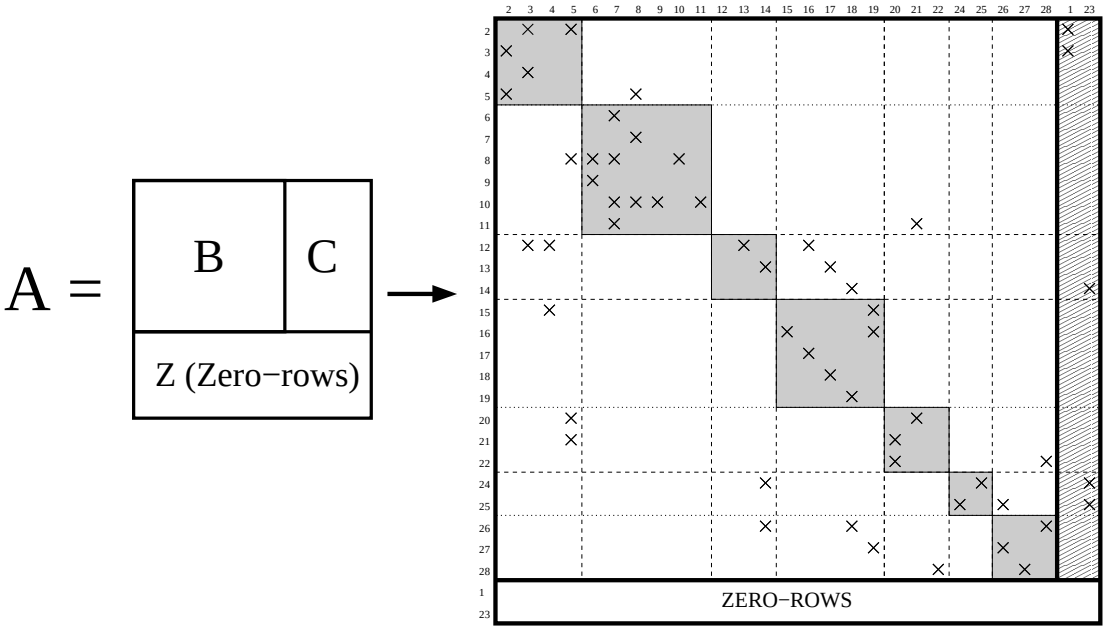


Figure 5.4: Ordered transition matrix **A** of the sample Web graph in fig. 5.3

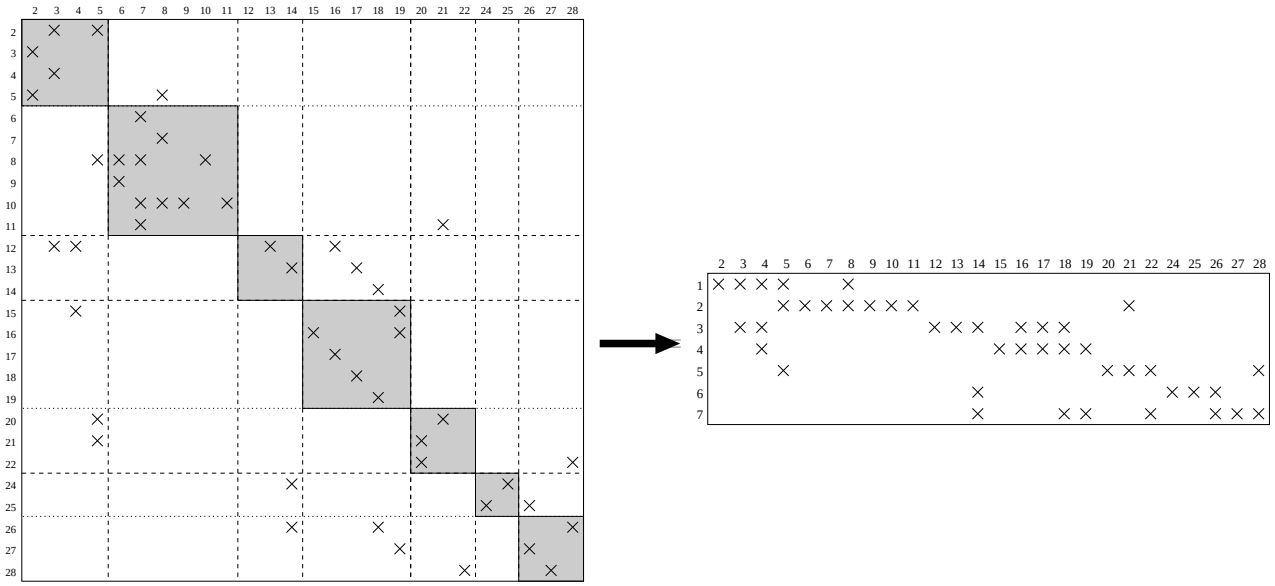


Figure 5.5: Site-to-page compression of **B**

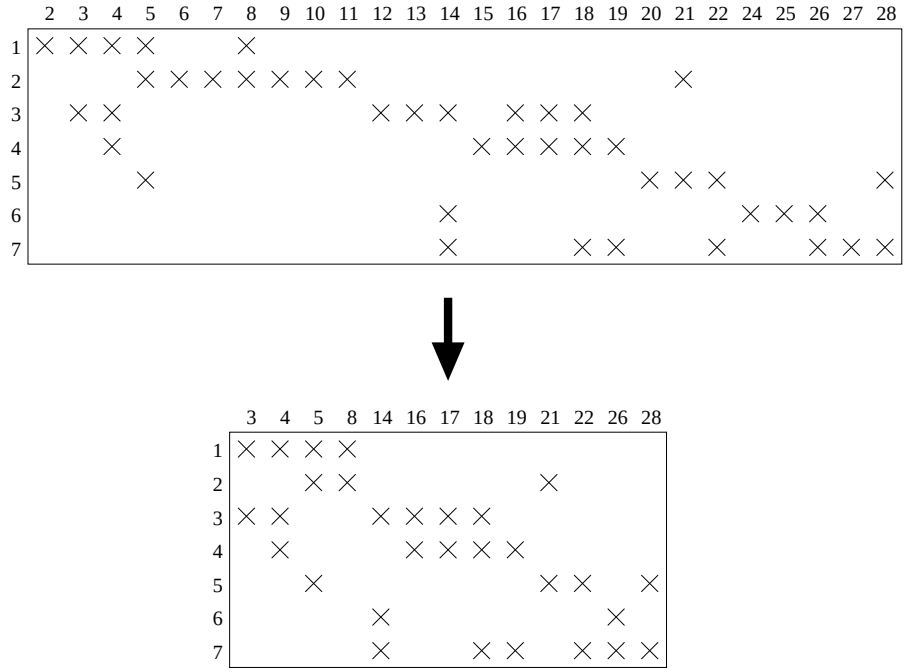


Figure 5.6: Removal of columns that contain single non-zero

After compression, many columns have only one non-zero. These columns represent the pages that contain only intra-site links. In the column net hypergraph, these columns are represented with nets that contain one vertex. Such nets are called *one-pin nets*. One-pin nets do not affect the cutsize, in other words they are always internal nets. Hence, we may remove these nets from the site hypergraph. Removal of columns that contain only one non-zero is depicted in Figure 5.6.

In the compressed matrix, many columns have the same non-zero pattern. This means that in the site hypergraph, many nets contain same set of vertices. In any partition, cutsize of the identical nets are also identical. Hence, we may gather identical nets into a single net with its cost set equal to the number of identical nets gathered together. Identical net elimination can be realized with a linear time algorithm as described in [9]. Figure 5.7 pictures the elimination of identical columns in S2P matrix, which corresponds to elimination of identical nets in column-net hypergraph of the S2P matrix with proper cost assignment.

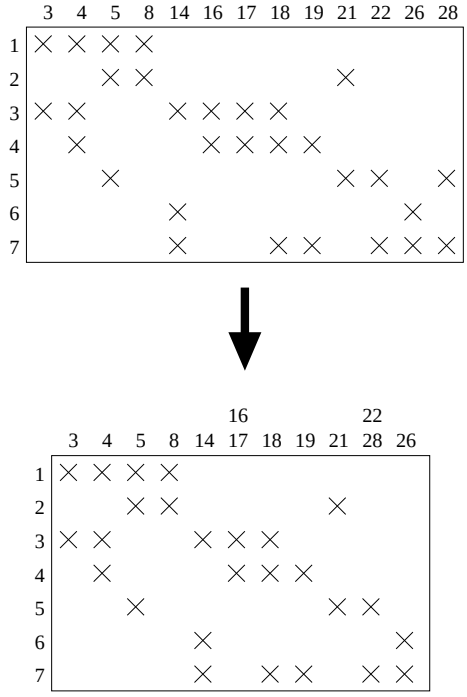


Figure 5.7: Gathering identical columns together

5.2.1.3 Vertex Weighting and Partitioning

For rowwise partitioning of matrices with column-net hypergraph model, vertices are generally weighted with the number of non-zeros in the corresponding row. This type of weighting considers only the workload of matrix-vector multiplication. However, in PageRank algorithm, we have linear vector and norm operations, which also incur computational load. While weighting vertices of site-based hypergraph, we consider the linear vector operations as well as the matrix-vector multiplication.

In calculation of computational load, we assume the time taken for scalar multiplication and addition operations are identical. Under this assumption, for each iteration, the cost of computing PageRank for a page i is $2x + 7$ flops, where x is the number of non-zeros in the i^{th} row of transition matrix $\mathbf{A}$. We calculate this cost as follows:

- Each non-zero in matrix-vector multiplication requires one multiplication and one addition ($y_i \leftarrow y_i + A_{ij}x_j$),
- Dampening factor requires one scalar multiplication ($y_i \leftarrow dy_i$),
- Calculation of γ and δ requires one subtraction and one addition, for each (see Figure 5.1),
- Addition of $\gamma\mathbf{v}$ with $\mathbf{r}$ requires one addition and one multiplication.

For the site-based model, vertices of the coarse hypergraph denote sites, instead of individual pages. Hence, each vertex in coarse hypergraph is weighted with the sum of weights of the pages in the corresponding site. For our sample Web matrix represented in Figure 5.4, the weight of vertex v_1 in column-net site hypergraph (which corresponds to the site M:`www.microsoft.com`) is $2 \times 8 + 4 \times 7 = 44$. There are totally 8 non-zeros in rows 2, 3, 4 and 5, which requires 16 flops for matrix-vector multiplication. These four pages require 28 flops for linear vector operations. After weighting vertices, the site hypergraph is partitioned. The matrix excluding zero-rows is partitioned according to vertex partition found by the hypergraph partitioning. $\mathbf{x}$ and $\mathbf{y}$ vectors are symmetrically partitioned according to the row distribution.

While weighting vertices, we do not count zero-rows, since they are not represented as vertices in site-hypergraph. Zero rows do not necessitate computation for matrix-vector multiplication, but they require computation for linear vector operations. PageRank value for a no-in-link page is $\gamma\mathbf{v}$. δ is calculated for these pages as usual. γ is equal to $\|\mathbf{q}\|_1$, since $\|\mathbf{r}\|_1$ is 0. Hence, computational load of zero-rows is 4 flops. We assign the responsibility of computing PageRank values for no-in-link pages to processors after hypergraph partitioning. Zero-rows can be thought as disconnected small vertices in site-based hypergraph. We assign zero-rows with a greedy bin-packing approach. For each zero-row, we assign it to the least loaded processor.

The $\mathbf{x}$ vector entries corresponding to the no-in-link pages are not assigned to processors, as they can be easily computed by any processor. This causes

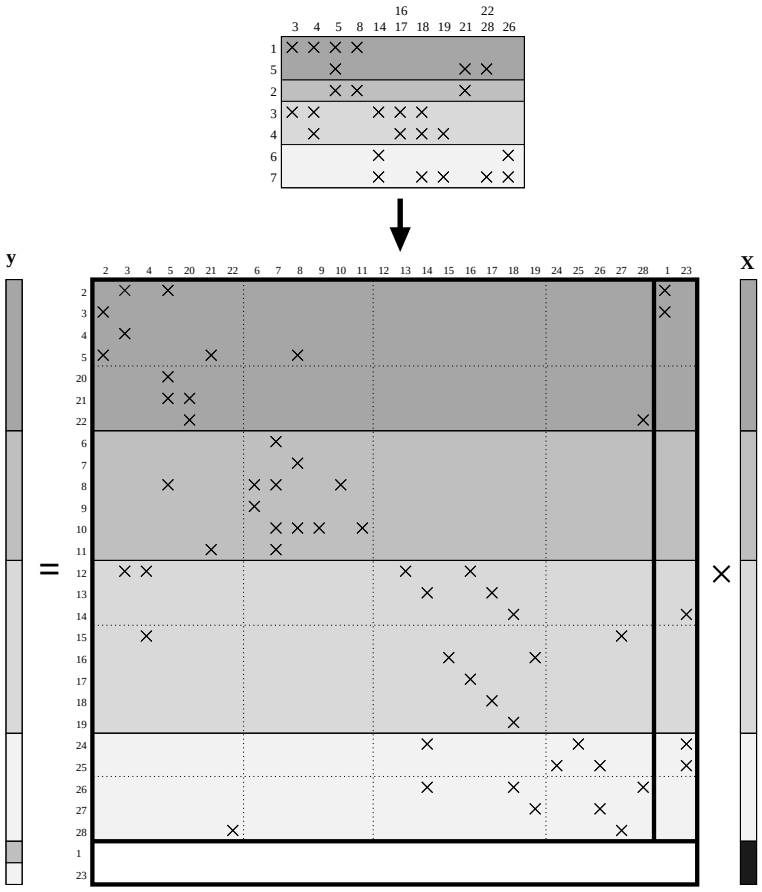


Figure 5.8: Rowwise partitioning of S2P matrix and its projection on **A**.

a small amount of computational redundancy on these vector entries to avoid expensive communication. The partitioning of S2P matrix and its projection to the transition matrix **A** and vectors **x** and **y** is presented in Figure 5.8.

5.2.2 Columnwise Partitioning

Site-based HP model for columnwise decomposition is similar to the rowwise model. In columnwise partitioning, the treatment for zero-rows and, as a result, vertex weighting differs. In columnwise model, the transition matrix is compressed on its columns, i.e., the row-net hypergraph of **A** is coarsened, while in rowwise partitioning, we compress the matrix **B** on its rows. In previous section,

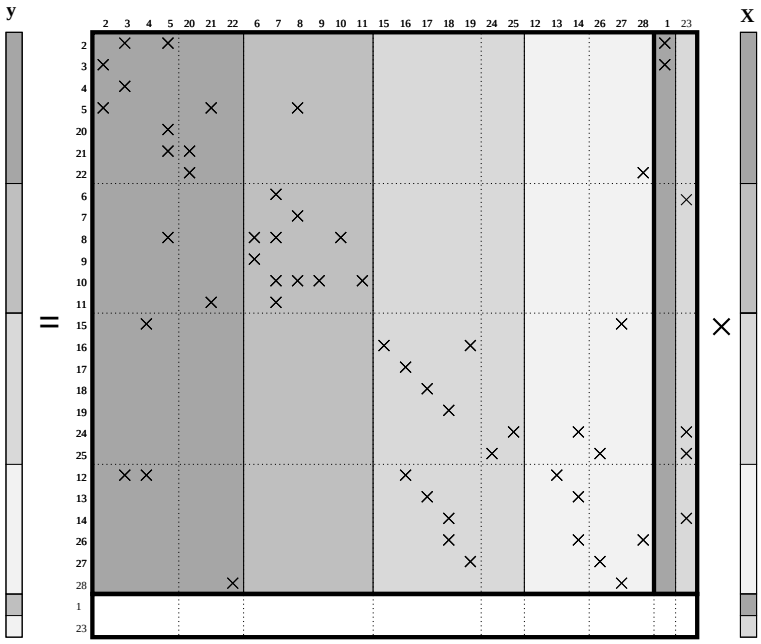


Figure 5.9: Columnwise partitioning of **A**.

we discussed that rowwise decomposition of sub-matrix **C** does not require communication on **x**, because PageRanks of no-in-link pages can be computed by any processor. However, columnwise partitioning of **C** may incur communication on output vector during parallel execution, since **C** is used to calculate PageRank values for pages with in-links. Hence, page-to-site matrix of **A**, including **C**, is partitioned on its columns by row-net site-hypergraph model.

Since PageRank values can be computed by any processor for no-in-link pages, symmetric partitioning is not required for input and output vectors of **A**. Under these circumstances, each no-in-link page contributes $2x$ units of weight and others contribute $2x+7$ units of weight in row-net hypergraph. Weights are calculated in the same manner with the rowwise model. We do not consider the linear vector operations for no-in-link pages in hypergraph model, using the flexibility of unsymmetric partitioning. Required linear vector operations on PageRank vector corresponding to the zero-rows may be assigned independent of the column distribution of **C**. PageRank calculation for no-in-link pages and vector operations cost 4 flops for each iteration. After columnwise decomposition of **B** and **C** by partitioning site-based coarse row-net hypergraph of **A**, PageRank computation

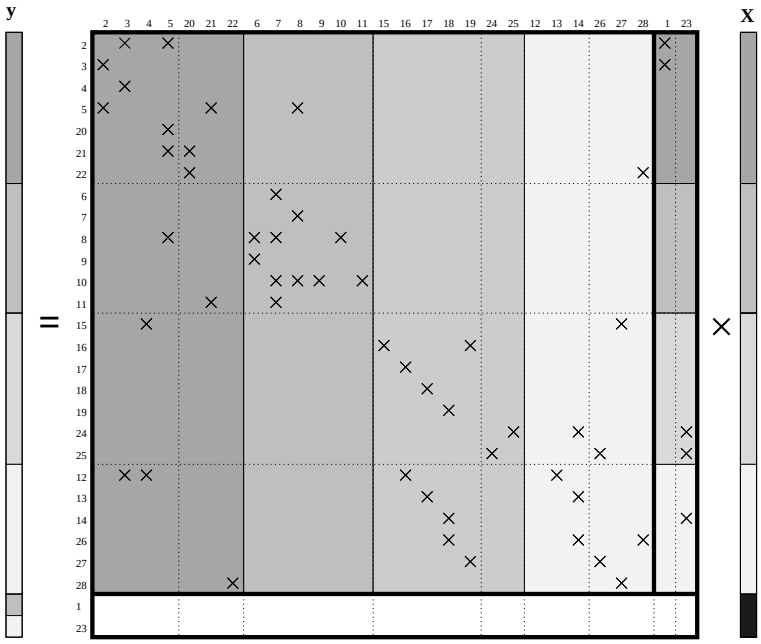


Figure 5.10: Columnwise partitioning of **A** while avoiding communication for sub-matrix **C**.

of no-in-link pages are assigned to processors by greedy bin packing. Columnwise decomposition of sample Web matrix in Figure 5.4 is represented in Figure 5.9.

This approach for columnwise partitioning has an advantage of finding better load imbalance. However, on the contrary to the rowwise model, columnwise model does not avoid the communication incurred by the sub-matrix **C**. To avoid communication for **C**, it should be partitioned rowwise, because of the reasons discussed above. Another approach to obtain lower communication volume may be partitioning **B** columnwise and partitioning **C** rowwise, accordingly. If column i of **B** is assigned to processor p, then row i of **C** should also be assigned to processor p. This can be achieved by partitioning row-net hypergraph of **B** considering the row weights of **C**. Hence, each vertex v_i of site-hypergraph to be partitioned is weighted $2(c_{Bi} + r_{Ci}) + 7n_i$, where c_{Bi} is the number of non-zeros in columns of **B** corresponding to site i, r_{Ci} is the number of non-zeros in rows of **C** corresponding to site i and n_i is the number of pages with in-links in site i. Columnwise partitioning of **B** in accordance with the rowwise partitioning of **C** is depicted in Figure 5.10.

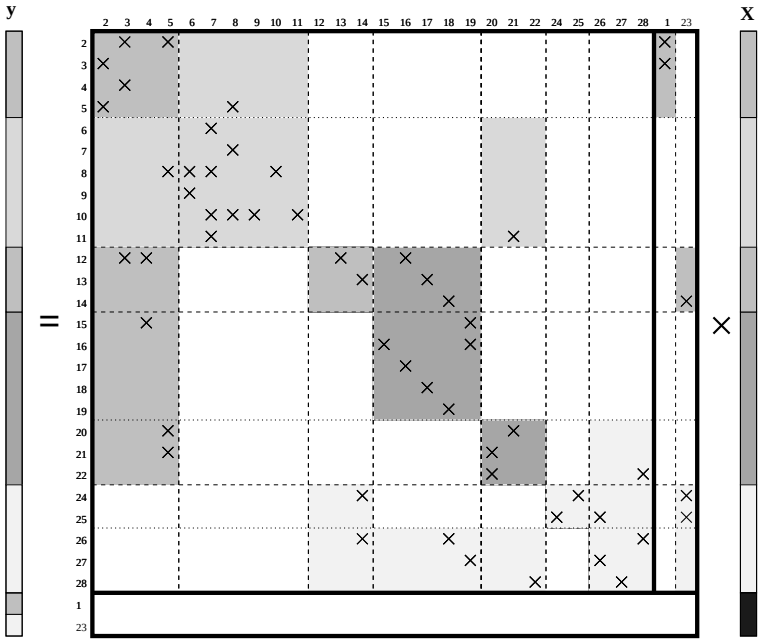


Figure 5.11: Site HP-based fine-grain partitioning of **A**.

5.2.3 Fine-Grain Partitioning

Fine-grain model is expected to find partitions with lower communication volume and better imbalance than other HP-based models. However, partitioning time for fine-grain hypergraph is longer.

In page-based fine-grain HP model, each non-zero in transition matrix **A** is considered as a vertex and each vertex is connected to two nets, one for its row and one for its column. For a Web-graph containing billions of links, it is very difficult to partition fine-grain hypergraph with current hypergraph partitioning tools, because of time and space complexity of the problem. Therefore, we compress the transition matrix site-to-site and each non-zero in S2S matrix is considered as a vertex in site-based fine-grain hypergraph.

To correctly encapsulate the communication volume, a vertex v_{ij}, which corresponds to the non-zero a_{ij} in the S2S matrix, is a pin of net n_r for all pages r in site i and net n_c for all page c in site j which has at least one in-link. Note that, the sub-matrix **C** require communication on **y**, but not require on **x**. Hence, we do not consider column-nets for **C**. One-pin nets and identical nets are eliminated in a similar fashion as discussed in Section 5.2.1.2.

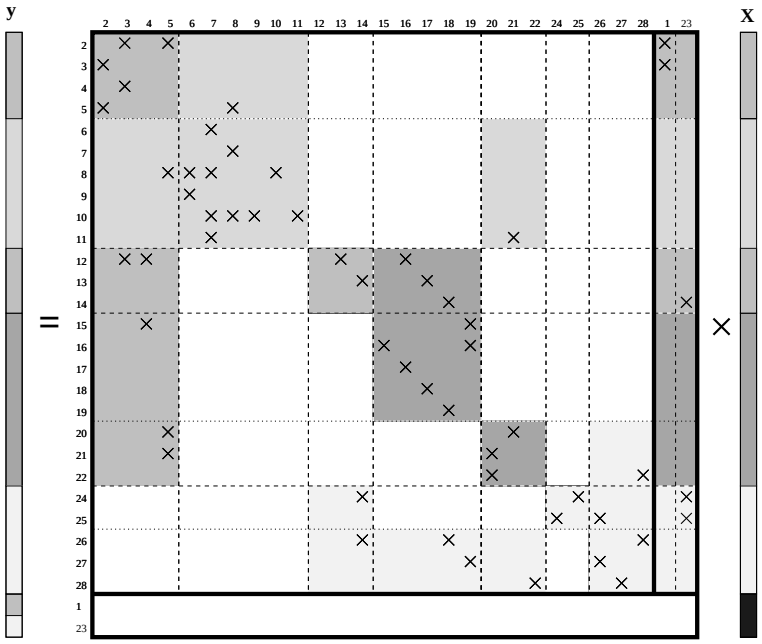


Figure 5.12: Fine-grain partitioning of **A** while avoiding communication for submatrix **C**.

A vertex v_{ij} has a weight of $2x$, where x is the number of links from site j to site i if $i \neq j$. Vertex v_{ii} is weighted as $2x+7y$, where x is the number of internal links in site i and y is the number of pages in site i which has at least one in-link. By doing so, we add the weights of vector operations to v_{ii}s. Therefore, processors which consists of vertices v_{ii} is responsible for computing PageRank values of pages in site i, excluding no-in-link pages. PageRank computation for no-in-link pages can be assigned to any processor without incurring any communication, with 4 units of weight for each. A sample fine-grain partitioning is represented in Figure 5.11 for matrix **A** in Figure 5.4.

To avoid communication for multiplication of **C** completely, it should be partitioned rowwise. Fine-grain partitioning of **B** in accordance with rowwise partitioning of **C** can be achieved by adding weights of rows on **C** to the vertices of **B** which corresponds to the diagonal entries in S2S compressed matrix of **B**. In this case, v_{ii} of site-based fine-grain hypergraph of **B** has an extra weight of $2z$, where z is the total number of non-zeros in rows of **C**, corresponding to the pages in site i. Fine-grain partitioning of **B** in accordance with rowwise partitioning of **C** is illustrated in figure 5.12.

5.2.4 Checkerboard Partitioning

Checkerboard partitioning is a 2D partitioning scheme, which has an advantage of reducing the number of inter-processor communications. Recall from the Section 4.4.3.2 that HP-based checkerboard partitioning is a two phase method, where two consecutive 1D partitionings generate a 2D partitioning.

Without loss of generality, assume a k-way partitioning, where $k = r \times c$. Matrix $\mathbf{A}$ is first r-way partitioned rowwise. This r-way partitioning is carried out with the same way as explained in Section 5.2.1. The only difference is, distribution of zero-rows to processors is delayed after partitioning on the other dimension. After partitioning rowwise, each r parts are c-way partitioned columnwise by multiconstraint hypergraph partitioning with r weights on each vertex. This is also done by a similar way as explained in Section 5.2.2. A vertex v_i, which corresponds to site i has r multi-weights, such that

$$w_i^j = \begin{cases} 2x + 7y & \text{if } v_i \text{ is in part } j \text{ in rowwise partitioning of } \mathbf{A} \\ 2x & \text{otherwise} \end{cases}$$

for all $1 \leq j \leq r$, where x is the number of non-zeros in j^{th} rowwise part of columns corresponding to site i and y is the number of pages in site i which has at least one in-link. Namely, vector operations for pages other than no-in-link pages are considered in their rowwise parts. Columnwise partitioning with multiconstraints finds c balanced partitions for each r rowwise partition. PageRank computation of no-in-link pages are assigned to processors as usual after checkerboard partitioning of $\mathbf{A}$, excluding zero-rows. A sample checkerboard partitioning is represented in Figure 5.13.

Note that, checkerboard partitioning described above does not require communication for input vector of $\mathbf{C}$, like other models. Rowwise partitioning of $\mathbf{C}$ in accordance with columnwise partitioning of $\mathbf{B}$ for avoiding communication on output vector of $\mathbf{C}$ can be achieved by modifying multiweights during columnwise partitioning. In this case, during columnwise multiconstraint partitioning of $\mathbf{B}$, if site i is assigned to processor j after rowwise partitioning, then w_i^j has an extra

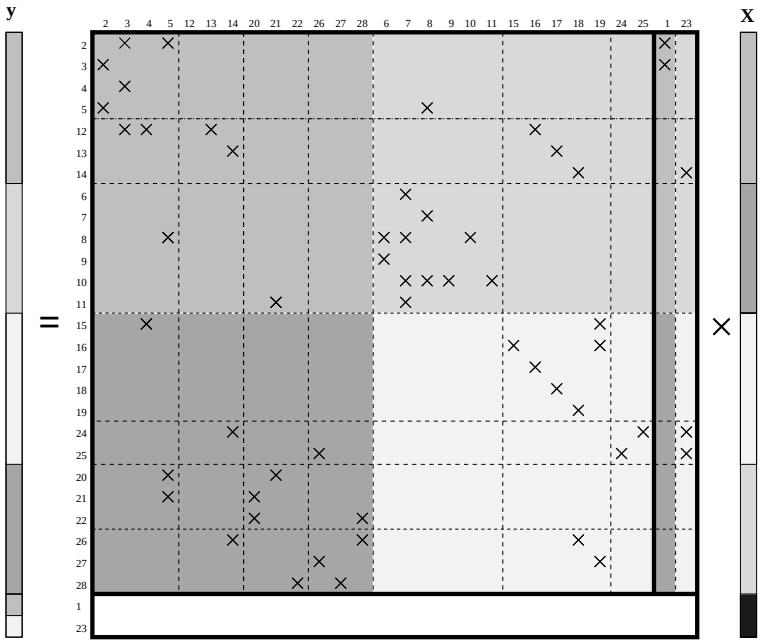


Figure 5.13: Site HP-based checkerboard partitioning of **A**.

weight of $2z$, where z is the number of non-zeros in rows of **C**, which corresponds to site i.

In this section, we have explained rowwise partitioning followed by a multiconstraint columnwise partitioning for checkerboard partitioning. First columnwise partitioning and then multiconstraint rowwise partitioning is possible with appropriate vertex weighting.

5.3 Site-Based GP Models

Site-based compression drastically reduces the partitioning overhead before the parallel PageRank computation. In HP models, hypergraphs are coarsened by site-based clustering. As a result of coarsening, one-pin nets and identical nets occur. Eliminating such nets drastically reduce the number of nets, too.

In 1D models, site-based coarsening of row-net or column-net hypergraph corresponds to S2P or P2S compression of the transition matrix. S2S compression

further reduces the preprocessing overhead. However, it is impossible to correctly encapsulate the total communication volume with row- or column-net hypergraph of S2S matrix.

S2S compression may be applicable to GP-based models, which try to minimize an approximation on total communication volume. The objective of GP-based models is to minimize the total number of non-zeros that require communication for multiplication, which does not mean to minimize the total communication volume. Standard GP-models that assign unit weight for each edge is for structurally symmetric matrices. Çatalyürek and Aykanat [17] propose GP-based models for structurally unsymmetric matrices. Our models are based on this work, since Web-matrices are structurally unsymmetric.

In this work, we propose site-based GP-based models for 1D rowwise and columnwise partitioning. Our approach is similar with 1D HP-based models. We compress the matrix $\mathbf{B}$ S2S to avoid communication for $\mathbf{C}$. Each site is represented as vertices and weighted with the same weights with HP-models (Sections 5.2.1 and 5.2.2). S2S matrix corresponds to the site-link graph of $\mathcal{G}$, where each vertex represents a site and each undirected edge represents link(s) between sites. The cost of each undirected edge e_{ij} is the total number of links from site i to site j and site j to site i. Partitioning of site-link graph assigns sites to processors which corresponds to partitioning of transition matrix rowwise or columnwise, according to weight assignment to sites.

GP-based models require shorter time for partitioning. On the other hand, HP-based models require less communication during parallel PageRank computation. Statistics and performance comparisons of GP- and HP-based models are presented in the next section.

Chapter 6

Experimental Results

In previous chapters, we have discussed the difficulties of parallelizing the PageRank and some shortcomings of parallelization efforts in the current literature. In the previous chapter, we have proposed six models for efficient parallelization of the PageRank computation. Site-based approach makes GP- and HP-based models applicable to huge-size Web data for parallel PageRank computation while providing comparable per-iteration time with the page-based approach.

In this chapter, we first provide the information on dataset properties and experimental environment, and then we will make performance comparisons of various models. Performance comparisons include preprocessing times, per-iteration times, communication volumes and load imbalances of page- and site-based GP and HP models.

6.1 Dataset Properties

We have used four Web-graph data with different sizes for our experiments. One of the datasets, which we will refer as the `Google` dataset, is provided by Google that includes .edu domain pages in the US [2]. `Google` dataset includes 913,569 pages and 15,819 sites with 4,480,218 links. Another dataset is a result of crawling

.de and .fr domains by Larbin [1], which we will refer as the `de-fr` dataset [5, 24]. `de-fr` dataset contains 8,212,782 pages, 69,129 sites and 39,330,881 unique links. Other two datasets, `in-2004` and `indochina`, are crawled by UbiCrawler [4] and uncompressed using WebGraph software [6]. Since `in-2004` and `indochina` are crawling results of restricted domains, they contain some sites that dominate datasets. For example, there is a site in `indochina` dataset which has nearly 7,500 pages and 50 million internal links. We have removed several such extraordinary sites from these two datasets. After removal of one site from `indochina` and four sites from `in-2004`, we have 7,407,256 pages, 18,984 sites and 145,877,411 links for `indochina` and 1,347,446 pages, 4,376 sites and 13,416,945 links for `in-2004`. The detailed information for datasets is provided in table 6.1. The properties for the transition matrix $\mathbf{A}$ can be extracted from the Web-graph information. Since we compress and partition the sub-matrix $\mathbf{B}$ instead of the transition matrix itself, our main focus is the matrix $\mathbf{B}$. Properties of $\mathbf{B}$ is presented in table 6.2. `In-2004` and `indochina` datasets are raw results for single crawl. Therefore, there are only a few zero-rows for those pages. For this reason, we do not reorder the transition matrix and handle zero-rows for them. For these two datasets, $\mathbf{B}$ is equal to the transition matrix $\mathbf{A}$.

Table 6.3 tabulates the site-based compression statistics for GP-based 1D partitionings. Tables 6.4, 6.5 and 6.6 presents the site-based column-net, row-net and fine-grain hypergraph properties, respectively. As can be seen from the tables, one-pin net and identical net eliminations drastically reduces the number nets. As a result, number of nets become comparable to number of vertices. For example, for column-net site-based hypergraph of `de-fr` dataset in table 6.4, 6.6 million of 8.1 million nets are one-pin nets, after compression. After removal of one-pin nets, there remains only 1.5 million nets. 1.3 million of 1.5 million nets are identical. Eliminating identical nets, we have only 0.2 million nets, which is roughly six times the number of vertices.

	number of				# of in-links		# of out-links		percentage of	
Web graph	pages	pages w/o in-links	sites	links	max	avg	max	avg	inter-site links	intra-site links
Google	913,569	132,167	15,819	4,480,218	5,989	4.90	618	4.90	87.42	12.58
in-2004	1,347,446	86	4,376	13,416,945	21,866	9.96	7,753	9.96	95.92	4.08
de-fr	8,212,782	69,129	3,8741	39,330,881	11,594	4.79	1,692	4.79	75.06	24.94
indochina	7,407,256	288	18,984	145,877,411	256,425	19.69	5,289	19.69	92.80	7.20

Table 6.1: Dataset Web graph information

	number of			maximum number of nonzeros per		average number of nonzeros per	
Dataset	rows	columns	nonzeros	row	column	row	column
Google	781,402	781,402	4,133,201	5,398	618	5,29	5,29
in-2004	1,347,446	1,347,446	13,416,945	21,866	7,753	9,96	9,96
de-fr	8,143,653	8,143,653	39,024,601	11,587	1,692	4,79	4,79
indochina	7,407,256	7,407,256	145,877,411	256,425	5,289	19.69	19.69

Table 6.2: Properties of the page-to-page matrix **B** after eliminating pages without in-links from the transition matrix **A**

	Before compression						After compression					
	number of		maximum		average		number of		maximum		average	
Dataset	vertices	edges	vertex degree	edge cost	vertex degree	edge cost	vertices	edges	vertex degree	edge cost	vertex degree	edge cost
Google	781,402	3,304,520	5,403	2	8.46	1.25	15,819	78,962	598	2,734	9.98	6.30
in-2004	1,347,446	10,772,385	21,869	2	15.99	1.21	4,376	26,048	1,071	100,275	11.90	21.00
de-fr	8,143,653	38,959,092	11,601	2	9.57	1.00	38,741	433,150	6,236	107,331	22.36	22.53
indochina	7,407256	133,238,699	256,457	2	35.98	1.09	18,984	246,689	9,861	140,402	25.99	42.56

Table 6.3: Site-graph properties for 1D decomposition

	After compression				After eliminating identical nets								
	number of				number of			maximum			average		
Dataset	one-pin nets	nets	pins	identical nets	vertices	nets	pins	net size	vertex degree	vertex weight	net size	vertex degree	vertex weight
Google	566,743	214,659	600,952	154,143	15,819	60,516	236,734	149	3,829	147,365	37.99	75.98	912.21
in-2004	1,142,340	205,106	555,195	193,818	4,376	11,288	49,938	475	20,003	781,515	126.87	253.75	8,287.48
de-fr	6,627,963	1,515,690	3,973,354	1,278,886	38,741	236,804	1,017,830	384	28,503	2,848,037	103.26	206.51	3,501.91
indochina	5,394,884	2,012,372	7,540,839	1,934,801	18,984	77,571	522,641	2,611	264,945	5,550,845	397.22	794.44	18,099.75

Table 6.4: Site-based column-net hypergraph models for rowwise decomposition

	After compression				After eliminating identical nets								
	number of				number of			maximum			average		
Dataset	one-pin nets	nets	pins	identical nets	vertices	nets	pins	net size	vertex degree	vertex weight	net size	vertex degree	vertex weight
Google	701,812	79,590	251,246	35,865	15,819	43,725	176,494	599	1,154	147,879	15.88	31.77	912.21
in-2004	1,323,809	23,637	77,691	15,296	4,376	8,341	43,421	923	1,245	816,577	17.75	35.51	8,287.48
de-fr	6,059,727	2,083,926	4,947,961	1,814,805	38,741	269,121	1,037,824	1,340	20,278	2,847,943	128.24	256.47	3,501.91
indochina	7,156,359	250,897	799,683	198,546	18,984	52,351	369,842	1,501	10,088	4,598,146	42.12	84.25	18,099.75

Table 6.5: Site-based row-net hypergraph models for columnwise decomposition

	After compression				After eliminating identical nets								
	number of				number of			maximum			average		
Dataset	one-pin nets	nets	pins	identical nets	vertices	nets	pins	net size	vertex degree	vertex weight	net size	vertex degree	vertex weight
Google	1,268,555	294,249	852,198	181,224	108,992	113,025	434,031	599	1,589	145,343	7.82	15.64	132.40
in-2004	2,466,149	228,743	632,886	207,998	32,434	20,745	96,257	923	9,969	781,509	19.51	39.03	1,118.15
de-fr	12,687,690	3,599,616	8,921,315	3,087,525	485,049	512,091	2,071,242	1336	19299	2,847,617	18.39	36.79	279.70
indochina	12,551,243	2,263,269	8,340,522	2,130,385	275,958	132,884	904,330	2,611	11,331	4,598,140	30.22	60.45	1,245.14

Table 6.6: Site-based hypergraph models for fine-grain decomposition

6.2 Experimental Setup

We conducted our experiments on a 32-node PC cluster interconnected with a 100 Mb/s Fast Ethernet switch. Each node contains an Intel Pentium IV 3.0 GHz processor with 1 GB of RAM. For partitioning Web matrices, we used an Intel Pentium IV 3.0 GHz machine with 2 GBs of RAM. The parallel PageRank algorithm is implemented using the MPI-based ParMxvLib C library [50, 52]. Direct k-way multilevel hypergraph partitioning tool kPaToH [9] is used for partitioning hypergraphs and graph partitioning tool METIS is used for multilevel k-way partitioning of graphs [34]. Imbalance tolerance is set to 3% for both graph and hypergraph partitioning.

The Web-matrices we work on, have unusual properties. As can be seen from the tables provided in previous section, datasets are highly irregular. Hence, state-of-the art techniques for partitioning the Web data may not provide the expected results. In a recent work, Abou-rjeili and Karypis [7] state that the partitioning quality for power-law graphs depends on the matching and clustering scheme used in the coarsening phase. During our experiments, we also observed that partitioning quality in HP-based models highly depends on the coarsening scheme in kPaToH. Hence, instead of default vertex based matching scheme for coarsening, we prefer absorption clustering scheme using nets. For multi-constraint partitioning, we use scaled heavy connectivity matching. In METIS, default parameters perform quite well, so we use default parameters for graph partitioning.

In our experiments, we always partition the sub-matrix $\mathbf{C}$ rowwise in accordance with partitioning of $\mathbf{B}$ in order to reduce the total communication volume avoiding the communication incurred by $\mathbf{C}$. Convergence parameter is set to 10^{-8}. With this settings, `Google` matrix converges in 91 iterations, `in-2004` converges in 90 iterations, `de-fr` converges in 83 iterations and `indochina` converges in 84 iterations.

6.3 Performance Evaluation

In this section, we compare preprocessing times, communication volumes, load imbalances and speedups of page-based and site-based approaches. We discuss trade-offs between various site GP- and HP-based models.

6.3.1 Preprocessing Times

Graph or hypergraph partitioning is the preprocessing for page-based models. For site-based GP, before partitioning the model graph, there is a compression. For site-based HP models, preprocessing refers to compression, identical net elimination and hypergraph partitioning. By employing site-based partitioning schemes, we become able to reduce the preprocessing overhead before the parallel PageRank computations. In this section, we provide preprocessing times for page- and site-based partitioning models to justify our arguments.

6.3.1.1 Page-Based Partitioning

As the size of the transition matrix increases, page-based partitioning becomes infeasible because of the time and space limitations. We have been able to partition two matrices in our dataset with page-based approach. The smallest dataset can be partitioned using 1D and 2D partitioning schemes by page-based HP and GP. However, with the hardware settings and sequential partitioning tools described in Section 6.2, it is only possible to 1D partition the second smallest matrix in our dataset using page-based approach. As in site-based models described in Chapter 5, we partition matrices by partitioning the sub-matrix $\mathbf{B}$, to handle zero-rows.

Figures 6.1 and 6.2 display page-based partitioning times for `Google` and `in-2004` Web-matrices. The numbers annotated with the bars represent the ratio of partitioning times to sequential per iteration time of PageRank computation. In the chart, the letter K stands for number of partitions. rw, cw, fine and

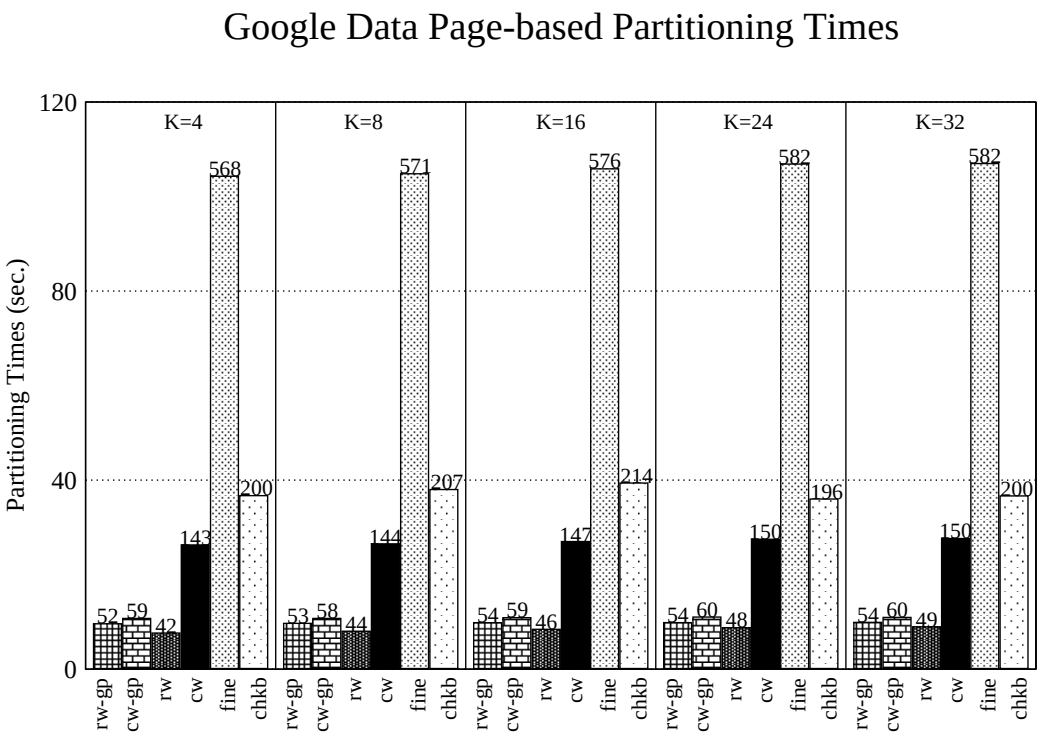


Figure 6.1: Page-based partitioning times for `Google` data

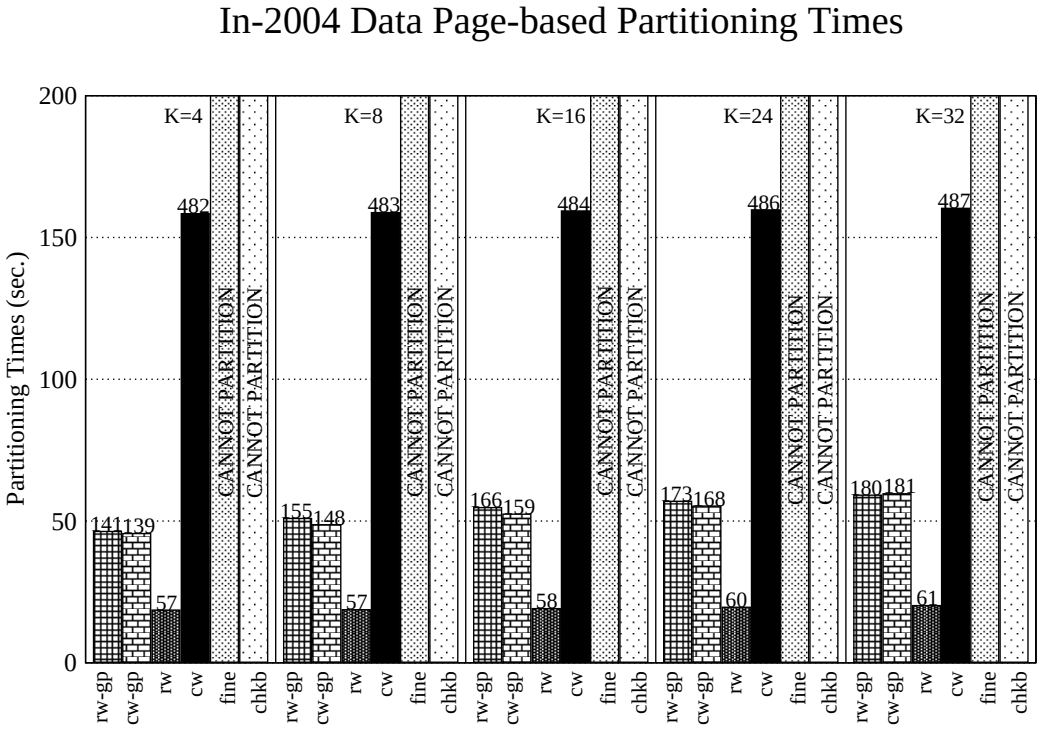


Figure 6.2: Page-based partitioning times for `in-2004` data

chkb are abbreviations for rowwise, columnwise, fine-grain and checkerboard partitionings, respectively. Partitionings are HP-based, unless otherwise stated. We append "-gp" to denote GP-based models. The statistics presented in the figures reveals that, at least 45 iterations are needed to amortize the preprocessing overhead of PageRank computation for `Google` data. For 2D partitioning schemes, hundreds of iterations can amortize preprocessing. It is not even possible to 2D partition `in-2004` matrix. On the other hand, PageRank vector for these two matrices stabilize in 90-91 iterations, by setting the convergence criteria to 10^{-8}.

6.3.1.2 Site-Based Partitioning

Site-based partitioning reduces the problem size, hence makes partitioning feasible for parallel PageRank computing. Figures 6.3 through 6.6 depicts the preprocessing times for test matrices we use in this work. It can be observed from the results that only several power iterations are required to amortize the preprocessing overhead for site-based partitioning schemes. For GP-based methods, preprocessing costs nearly one sequential power iteration. For `in-2004`, the slowest partitioning costs only 4.1 times per-iteration time.

HP-based models better minimize communication volume, as will be discussed in Section 6.3. However, GP-based models outperform HP-based models in preprocessing time. 2D models require relatively high preprocessing times, since fine-grain hypergraph has more vertices and checkerboard partitioning is two phase and require multiconstraint partitioning.

To construct site-based graph or hypergraph to be partitioned, compression of the transition matrix (i.e., coarsening the graph/hypergraph) is required. Identical net elimination from the coarser hypergraph makes the hypergraph to be partitioned smaller. Table 6.7 tabulates preprocessing times of site-based models for 16-way partitioning of Web-matrices. Figure 6.7 represents the percent dissections of preprocessing times of site-based models for 16-way partitioning. Compression time dominates the total preprocessing time for GP-based models. For HP-based models, dominating preprocessing step is the partitioning.

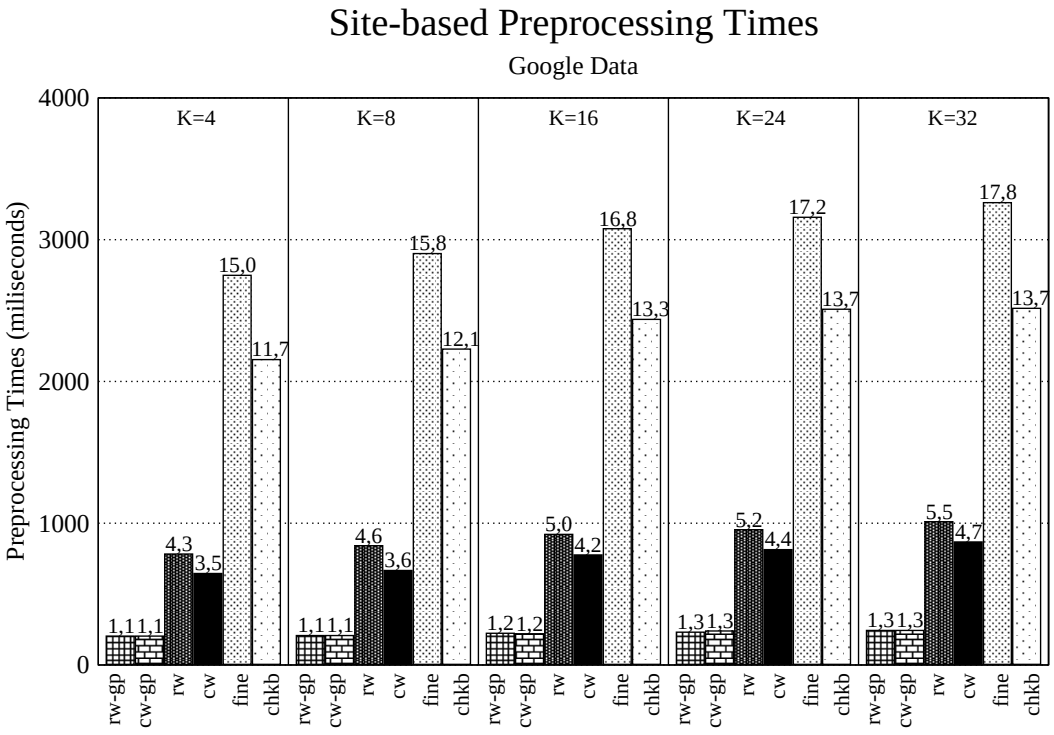


Figure 6.3: Site-based partitioning times for `Google` data

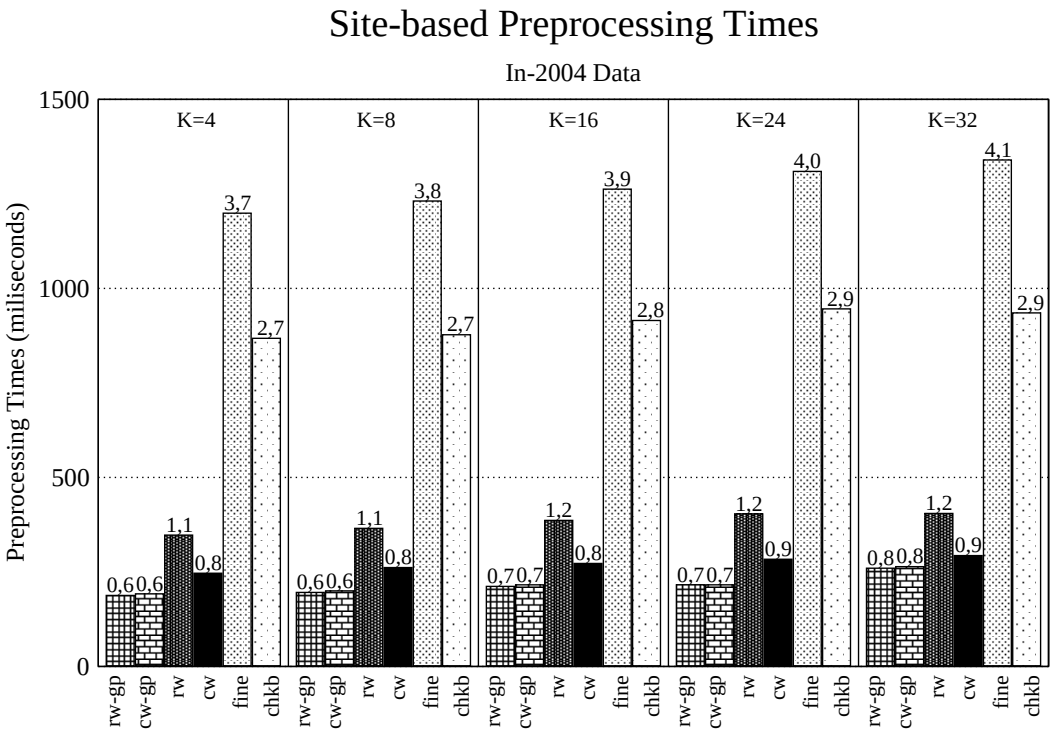


Figure 6.4: Site-based partitioning times for `in-2004` data

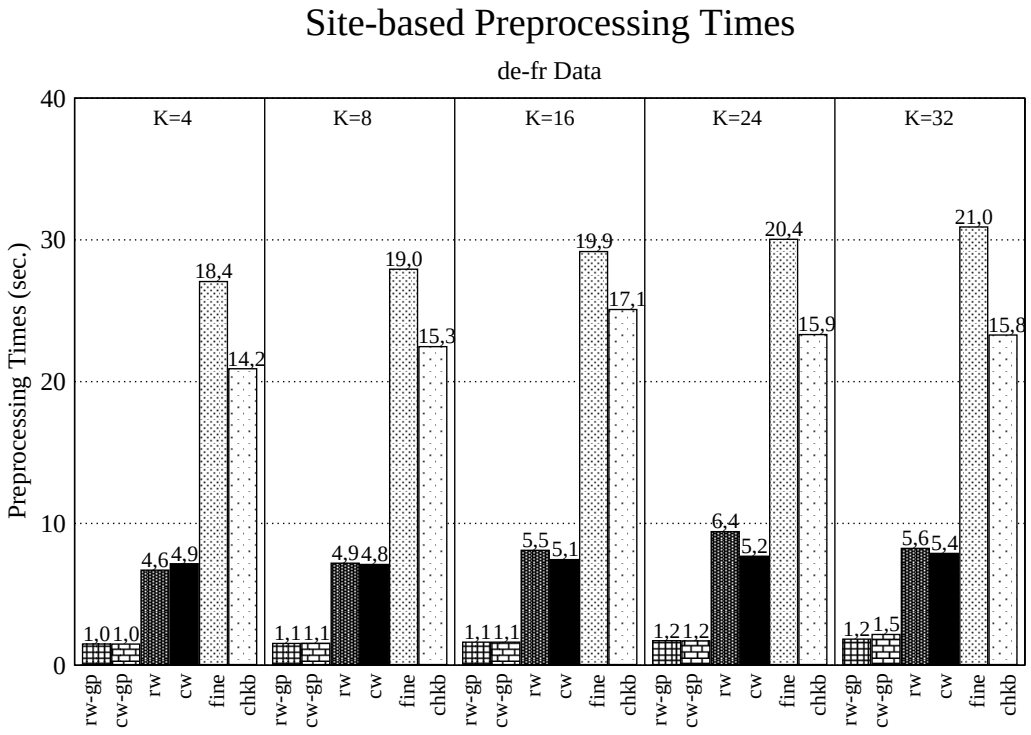


Figure 6.5: Site-based partitioning times for `de-fr` data

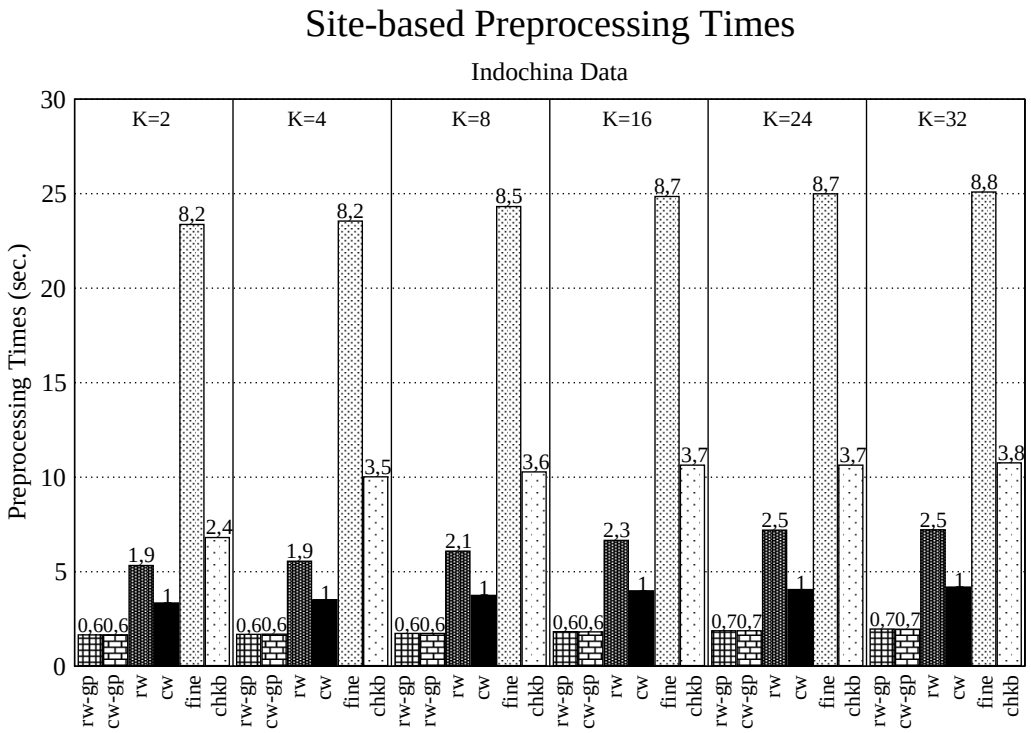


Figure 6.6: Site-based partitioning times for `indochina` data

Dataset	Partitioning Scheme	Times in milliseconds		
		Compression	Iden. Net Elim.	Partitioning
Google	Rowwise GP	136	-	88
	Columnwise GP	136	-	84
	Rowwise HP	148	88	685
	Columnwise HP	116	32	628
	Fine-grain HP	576	140	2,361
	Checkerboard HP	260	116	2,061
In-2004	Rowwise GP	168	-	44
	Columnwise GP	168	-	48
	Rowwise HP	200	36	150
	Columnwise HP	156	8	108
	Fine-grain HP	752	40	470
	Checkerboard HP	348	40	527
de-fr	Rowwise GP	1,072	-	564
	Columnwise GP	1,072	-	556
	Rowwise HP	1,028	484	6,605
	Columnwise HP	1,072	628	7,919
	Fine-grain HP	7,008	1,040	5,745
	Checkerboard HP	2,036	1,104	21,131
Indochina	Rowwise GP	1,516	-	306
	Columnwise GP	1,516	-	298
	Rowwise HP	1,944	464	4,249
	Columnwise HP	1,468	72	2,447
	Fine-grain HP	8,728	544	15,581
	Checkerboard HP	3,368	520	6,751

Table 6.7: Preprocessing times for 16-way partitioning with site-based partitioning schemes

6.3.2 Partition Statistics and Speedups

In this section, we compare parallel speedups of different partitioning schemes. Total communication volume, maximum communication volume per processor and load imbalance are some of important factors that affect the parallel computation time. These three factors should be smaller for better partitions. We provide partitioning results and speedups for page- and site-based models and justify that parallel computation performance do not degrade with the achievement of low preprocessing time.

Number of communications is another factor that affects the parallel performance of the iterative algorithm. We do not tabulate the number of communications in this section, since almost all processors communicate each other. Usually,

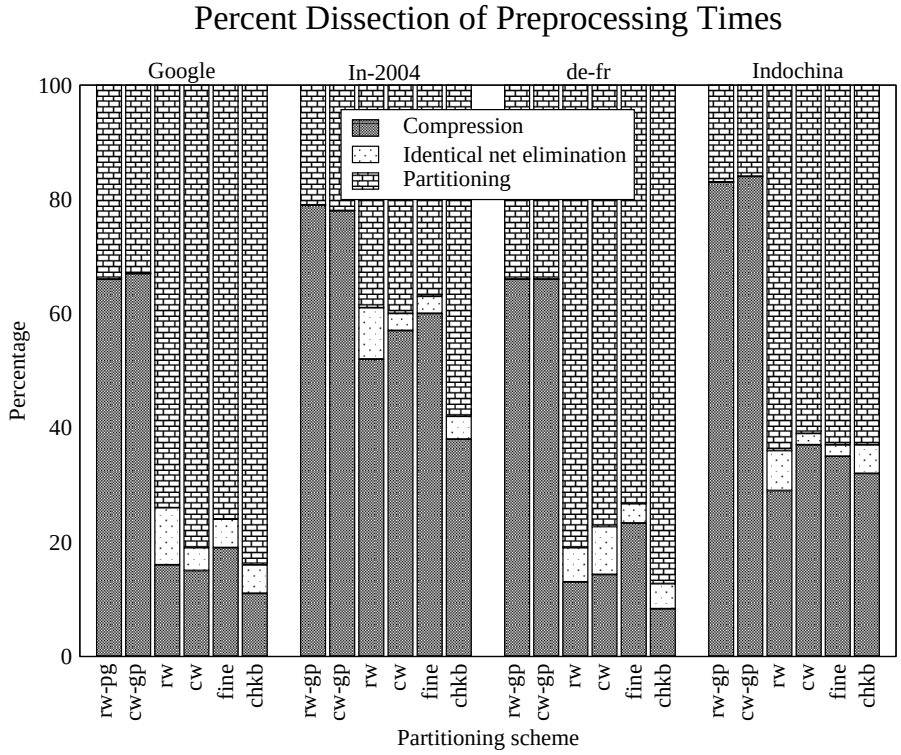


Figure 6.7: Percent dissections of preprocessing times of site-based models for 16-way partitioning.

maximum number of communications per processor is $K-1$ for 1D partitioning schemes, where K is the number of processors. For fine-grain partitioning, maximum number of communications per processor is $2(K-1)$ and for checkerboard partitioning it is $(r-1)+(c-1)$, where $K = r \times c$, r is the number of row blocks and c is the number of column blocks of checkerboard-partitioned iteration matrix. For parallel applications with high communication volume and high computation time, number of communications become insignificant.

Figures 6.8 to 6.15 depicts page-based partitioning statistics and Figures 6.16 to 6.31 depicts the site-based partitioning statistics, in bar charts. We provide page-based partitioning statistics for only two datasets, since others are too big to be partitioned using page-based approach. Total communication volumes and maximum sending communication volumes per-processor are given in number of double-words exchanged. Load imbalance of a partition is defined as the ratio

of difference between maximum load and average load to average load. Imbalances are given in percentages. Speedup is the ratio of sequential running time to parallel running time. Ideal speed up for k-way parallel execution is k. For `indochina` dataset, speed up is measured with comparison to 2-way parallel running time, since sequential implementation for this data requires disk swapping. As the figures provided in Section 6.3.1, K indicates the number of partitions and suffix "-gp" indicates the GP-based partitioning.

6.3.2.1 Page-based vs. Site-based Performance

As seen in the figures 6.8 to 6.23, site-based models achieves comparable performance to page-based models. In general, page-based partitioning obtains better load imbalance(Figures 6.10, 6.14, 6.18 and 6.22). For GP-based partitioning, site-based models achieve lower total communication volume (Figures 6.8, 6.12, 6.16 and 6.20), hence better speedup values (Figures 6.11, 6.15, 6.19 and 6.23). For HP-based partitioning, in some cases site-based models achieve better cut-size (lower total communication volume) and in some cases page-based models achieve better cut-size. For example, site-based partitioning for rowwise and checkerboard models of `Google` Data has lower communication volume than page-based partitioning, but for columnwise model, page-based partitioning performs slightly better.

The results obtained support the observations that sites are natural clusters for pages and site-based clustering provides a good coarsening for GP and HP.

6.3.2.2 Comparisons of Site-based Models

We observe that HP-based partitioning always provides lower total communication volume than GP-based partitioning. This is natural since GP tries to minimize a different metric, whereas HP tries to minimize correct metric for total communication volume. However, in some cases maximum communication volumes of GP-based partitions are close to HP-based partitions (columnwise

partitioning of `in-2004`, Figure 6.21). GP-based partitioning generally provides better load imbalance. As a result, GP-based partitioning rarely achieves better speedup values over HP-based partitioning (columnwise partitioning of `in-2004`, Figure 6.23).

In general, columnwise partitioning provides lower communication volume than rowwise partitioning, since columns are more similar than rows. Recall that columns of the transition matrix **A** corresponds to hyperlinks inside a page. For `indochina` dataset, there is a big difference between total volumes of columnwise and rowwise decomposition (Figure 6.28). Hence, columnwise model provides nearly two times better speedup (Figure 6.31). Parallel matrix-vector multiplication based on columnwise partitioning requires extra addition operations on **y** vector entries after communication (see Section 4.4.2). This may have a negative effect on speedup values of columnwise decomposition, especially for sparser matrices (see HP-based rowwise and columnwise speedups for `Google` data in Figure 6.19).

In general, the smallest communication volume and load imbalance is provided by 2D fine-grain partitioning model. Lower communication volume and better imbalance make fine-grain model to perform best for `indochina` data (Figure 6.31). On the other hand, fine-grain model requires communication before and after computation for matrix-vector multiplication. Number of communications for fine-grain model is more than the other models. For this reason, better reduction on total communication volume for fine-grain model, does not always reflect to speed-up values, especially for reduction of small communication volumes (Figure 6.19).

Checkerboard model mostly provides worse load imbalance and total communication volume is relatively high, compared to other models. However, checkerboard model reduces the number of communications. Especially, for large number of processors and lower total communication volume, this reduction on number of communications is significant. For example, checkerboard model performs best for 32-way site-based decomposition of `in-2004` data (Figure 6.23).

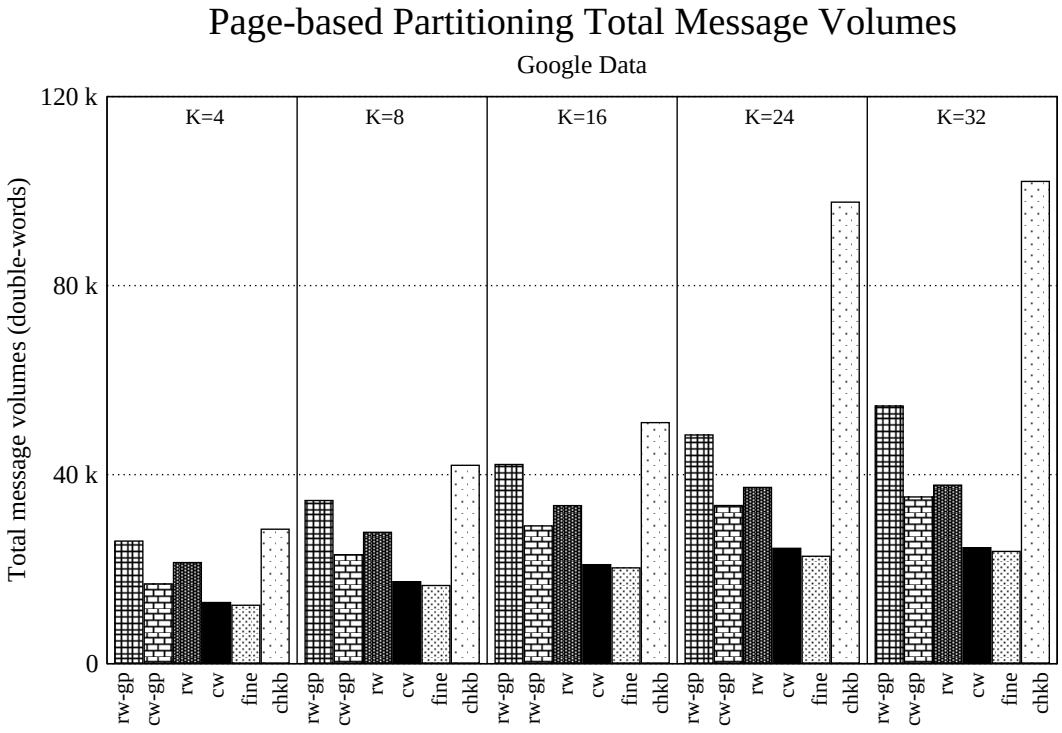


Figure 6.8: Average total communication volumes for page-based partitionings of `Google` data.

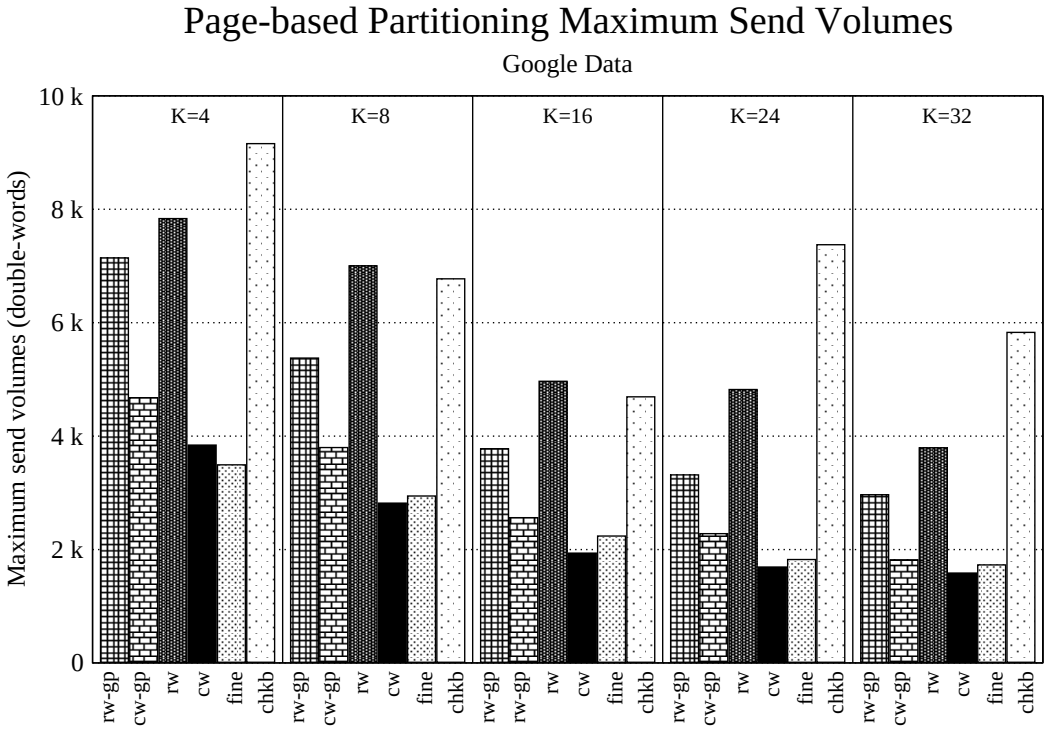


Figure 6.9: Average maximum per-processor communication volumes for page-based partitionings of `Google` data.

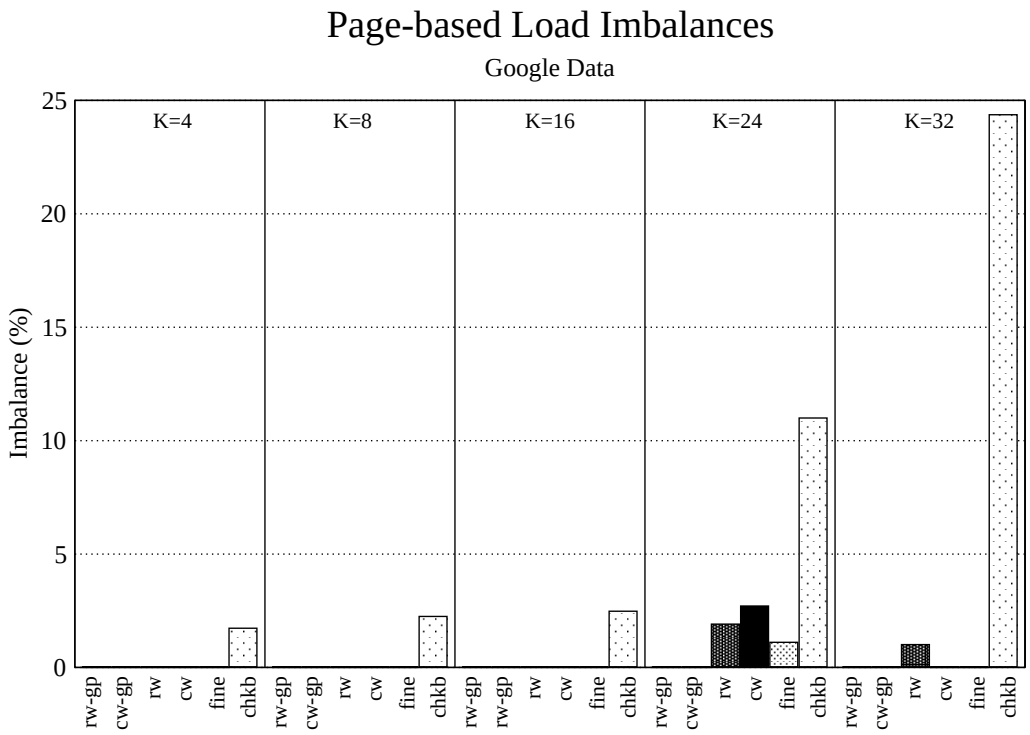


Figure 6.10: Average load imbalances for page-based partitionings of `Google` data.

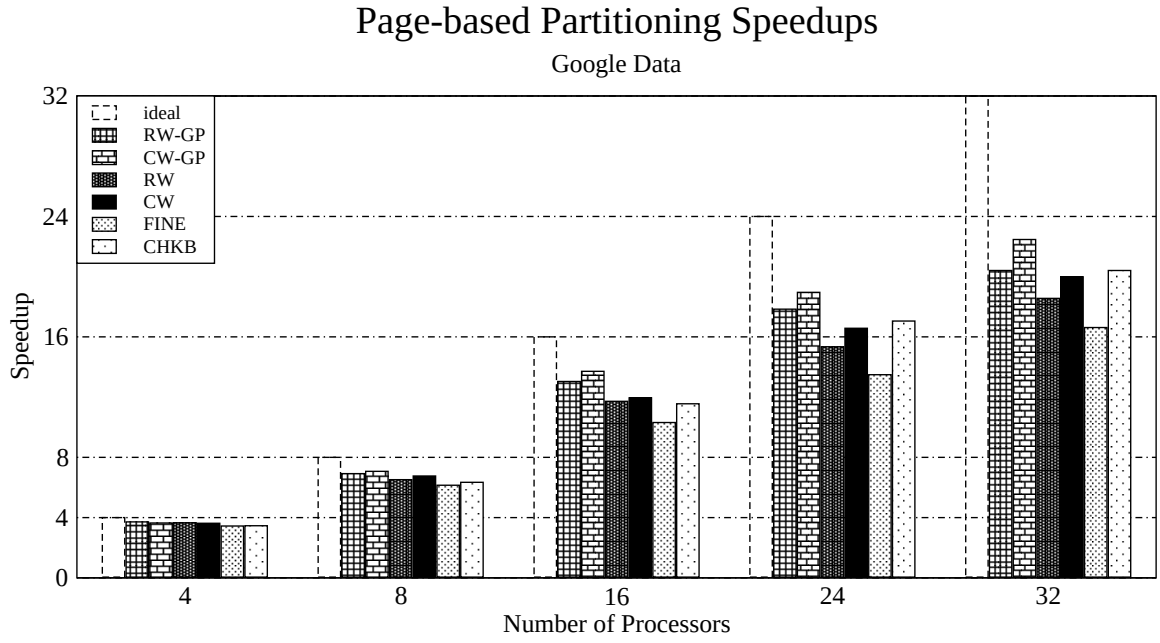


Figure 6.11: Average speedups for page-based partitionings of `Google` data.

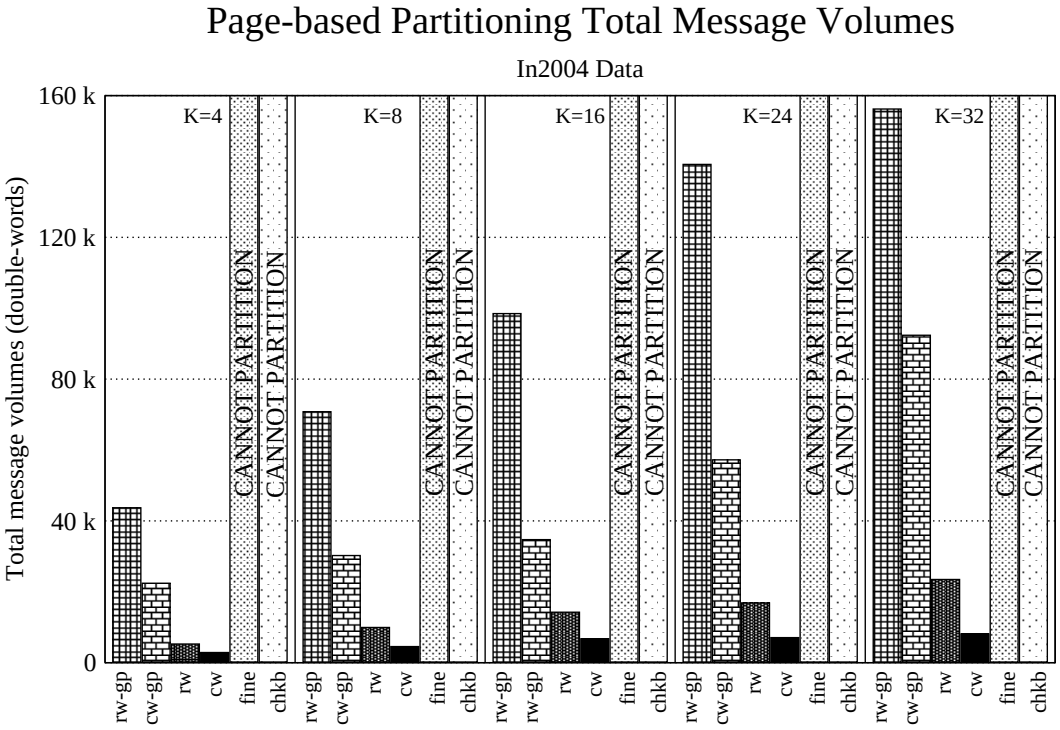


Figure 6.12: Average total communication volumes for page-based partitionings of `in-2004` data.

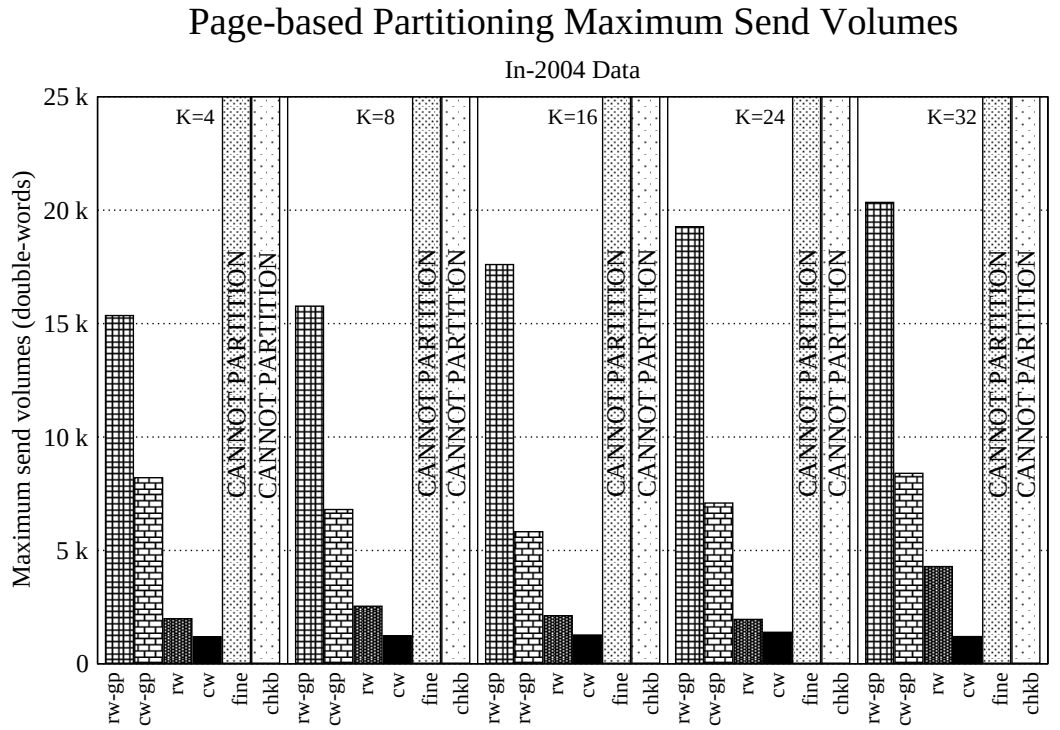


Figure 6.13: Average maximum per-processor communication volumes for page-based partitionings of `in-2004` data.

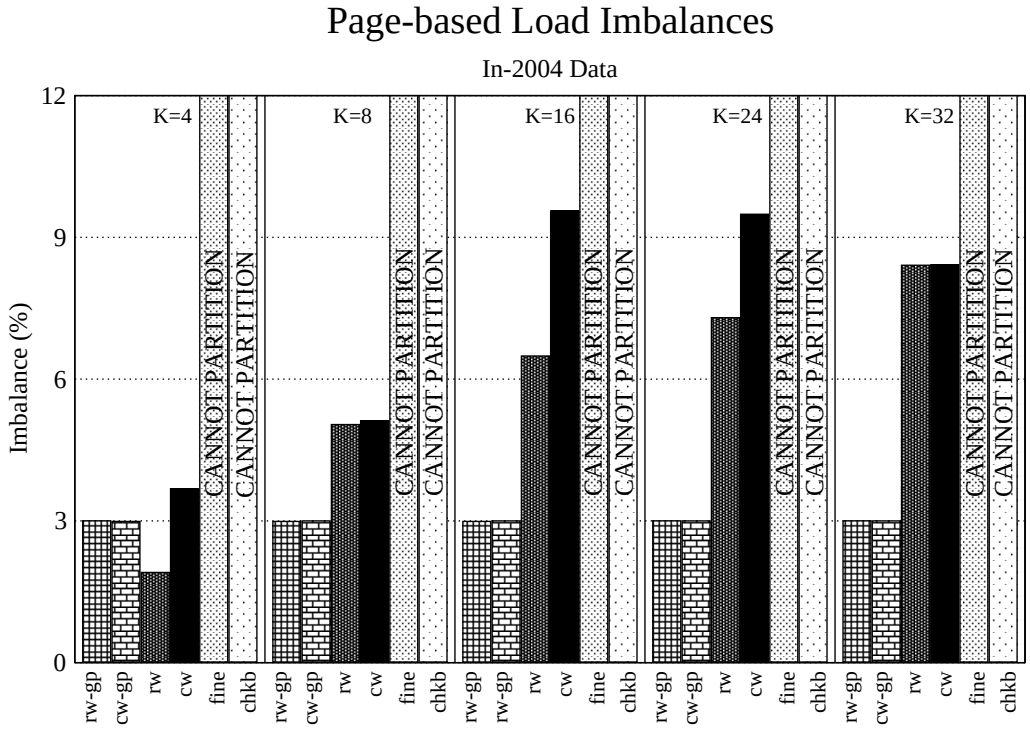


Figure 6.14: Average load imbalances for page-based partitionings of `in-2004` data.

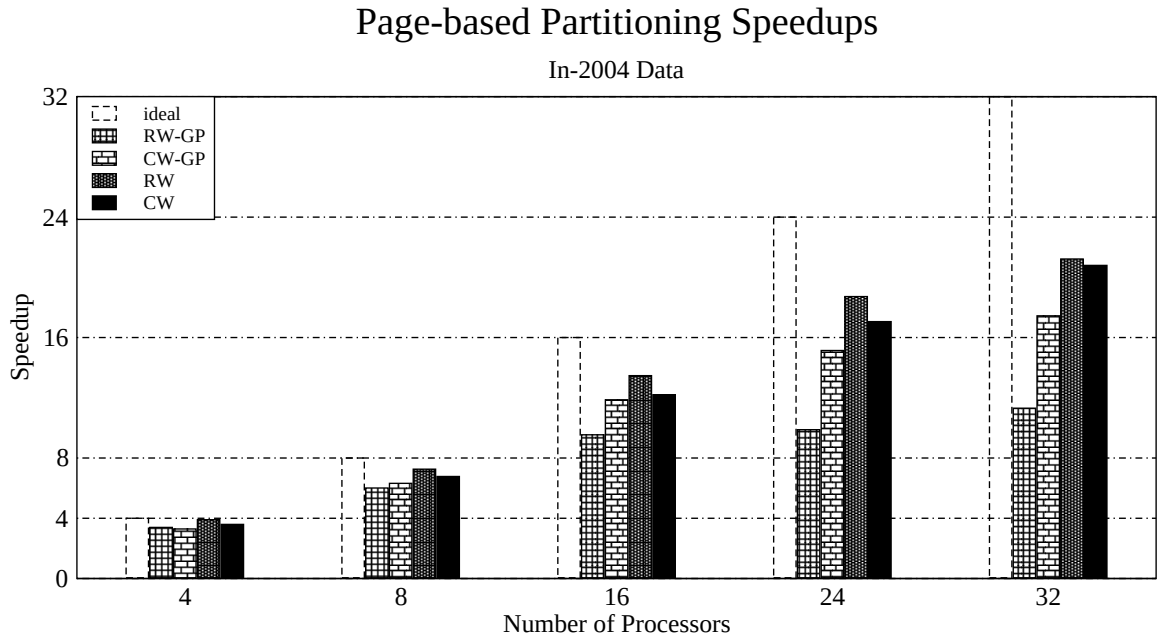


Figure 6.15: Average speedups for page-based partitionings of `in-2004` data.

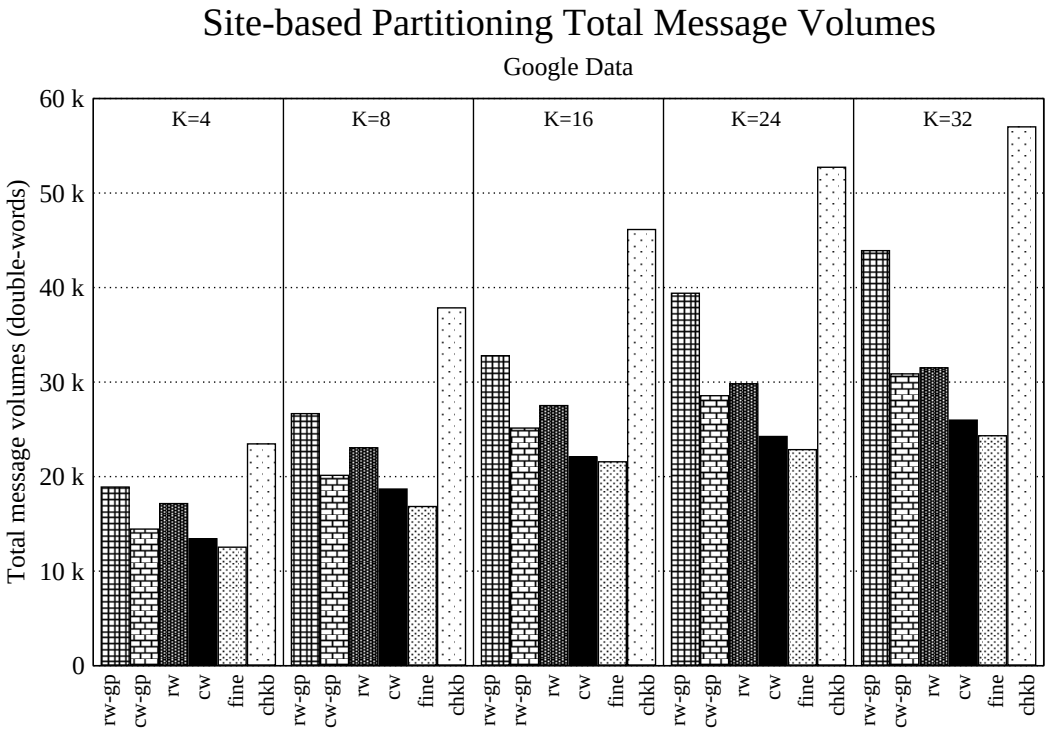


Figure 6.16: Average total communication volumes for site-based partitionings of `Google` data.

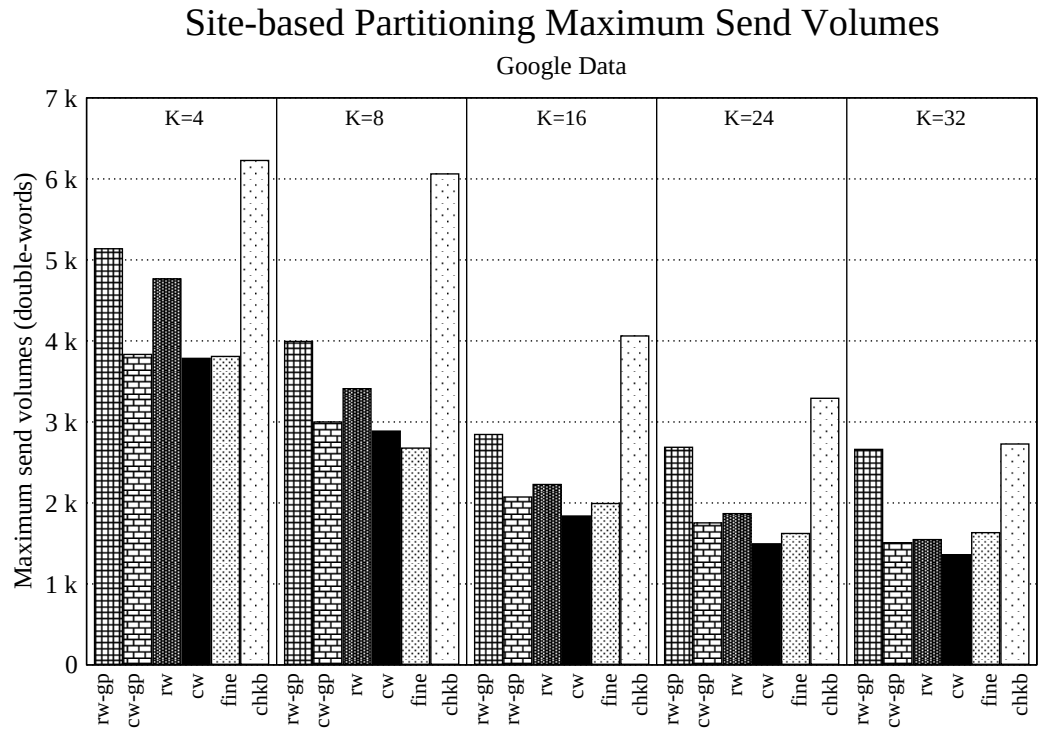


Figure 6.17: Average maximum per-processor communication volumes for site-based partitionings of `Google` data.

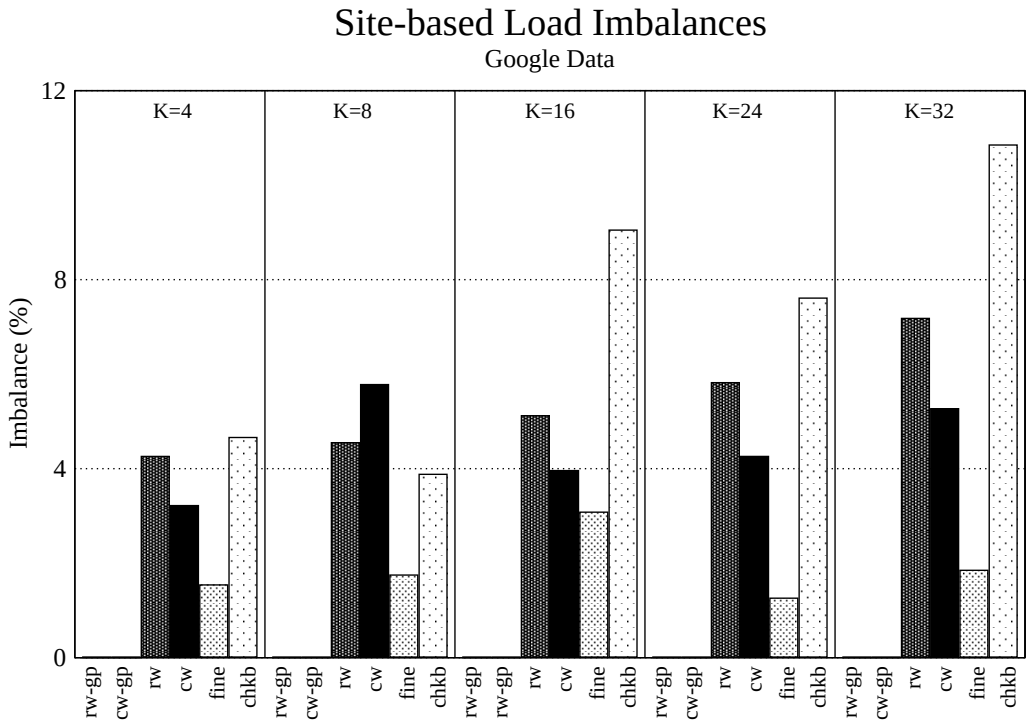


Figure 6.18: Average load imbalances for site-based partitionings of `Google` data.

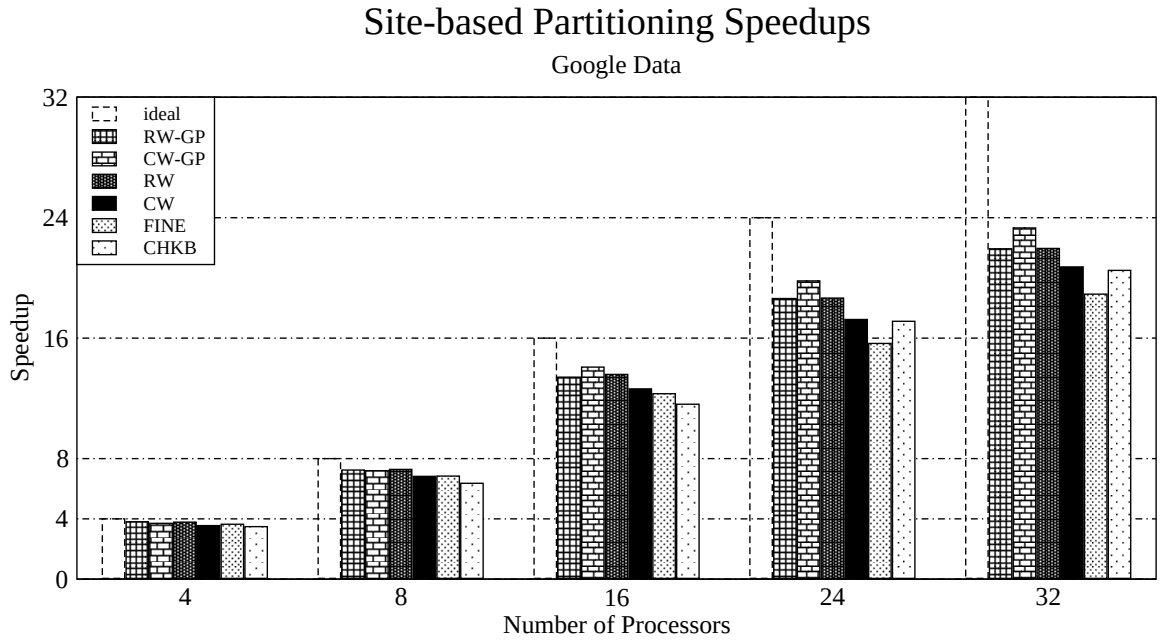


Figure 6.19: Average speedups for site-based partitionings of `Google` data.

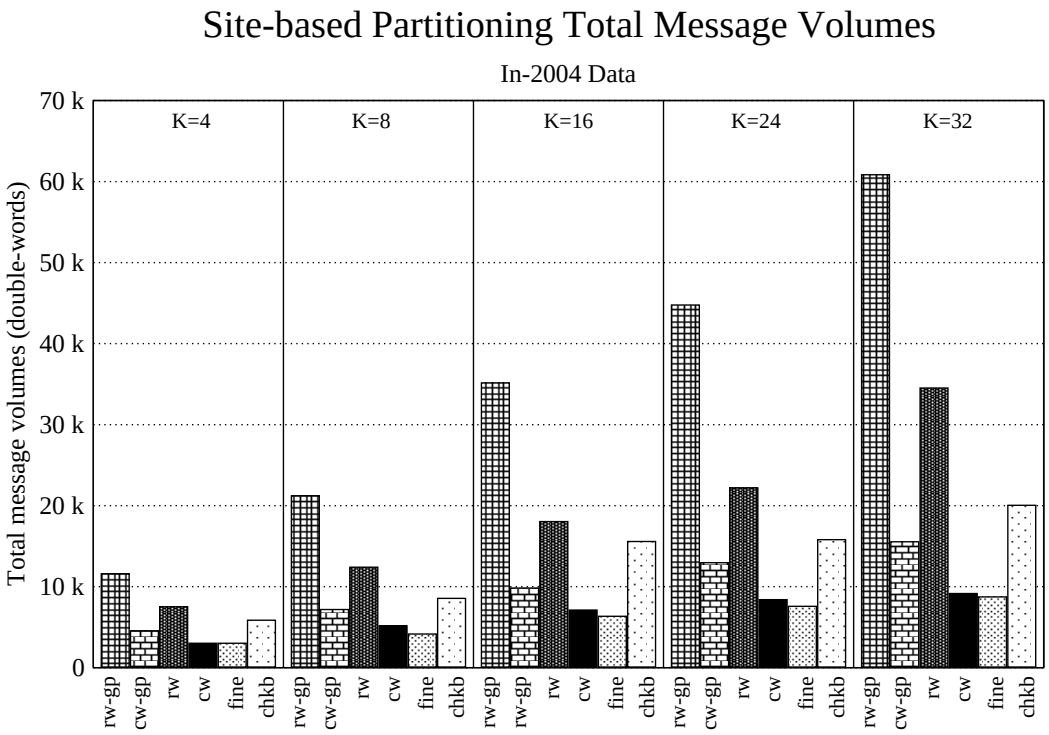


Figure 6.20: Average total communication volumes for site-based partitionings of `in-2004` data.

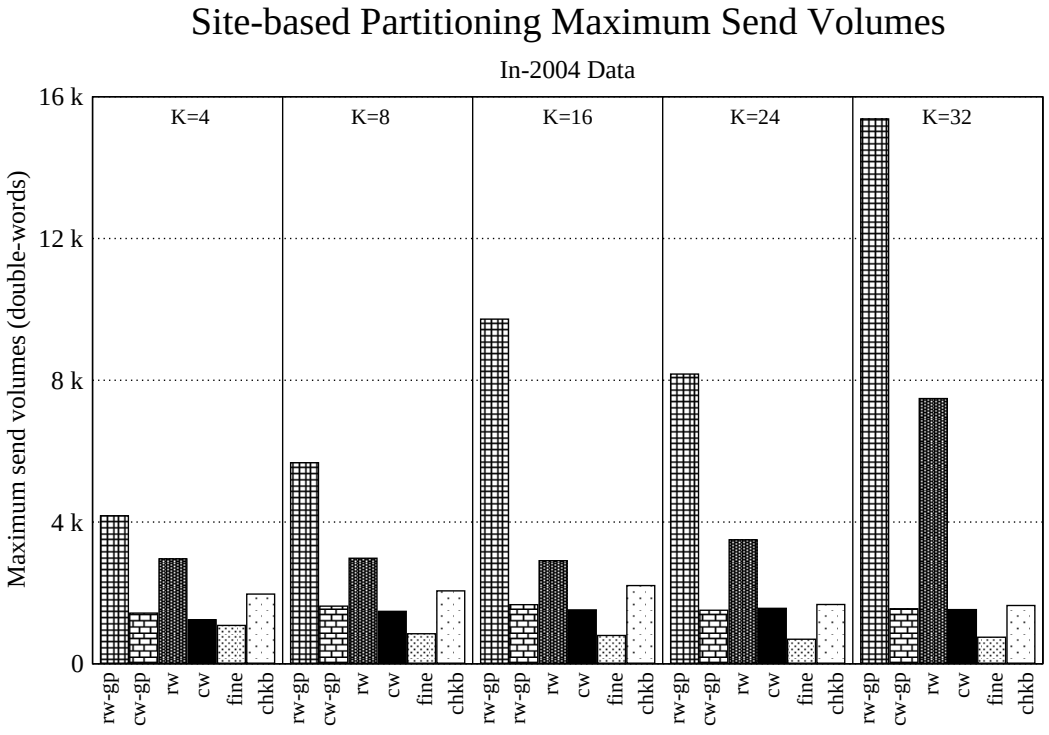


Figure 6.21: Average maximum per-processor communication volumes for site-based partitionings of `in-2004` data.

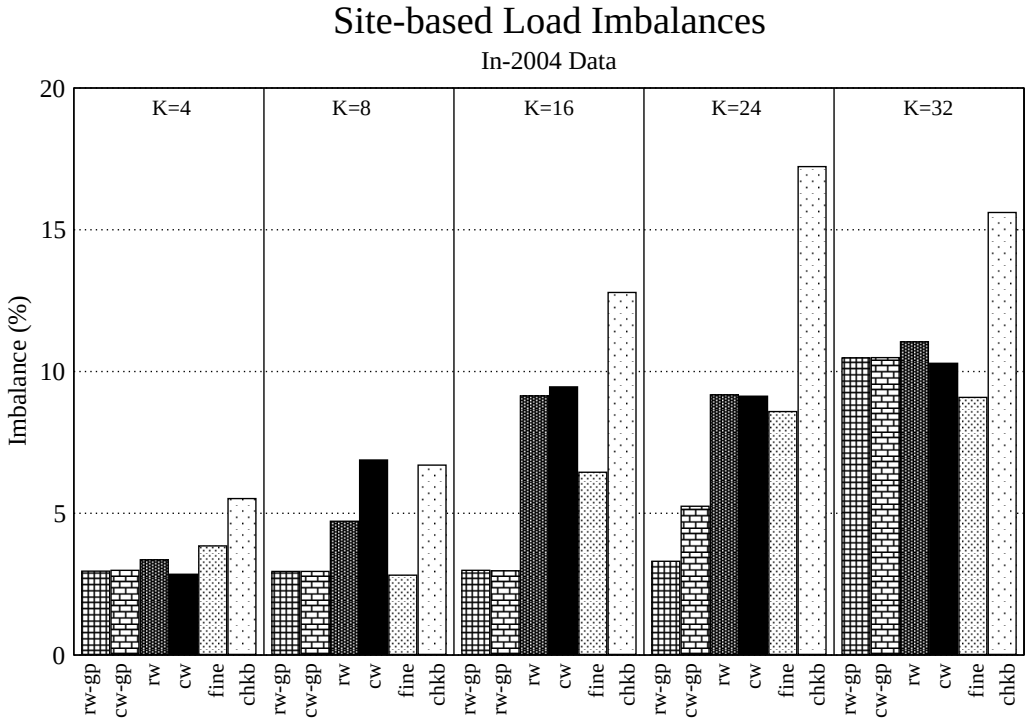


Figure 6.22: Average load imbalances for site-based partitionings of `in-2004` data.

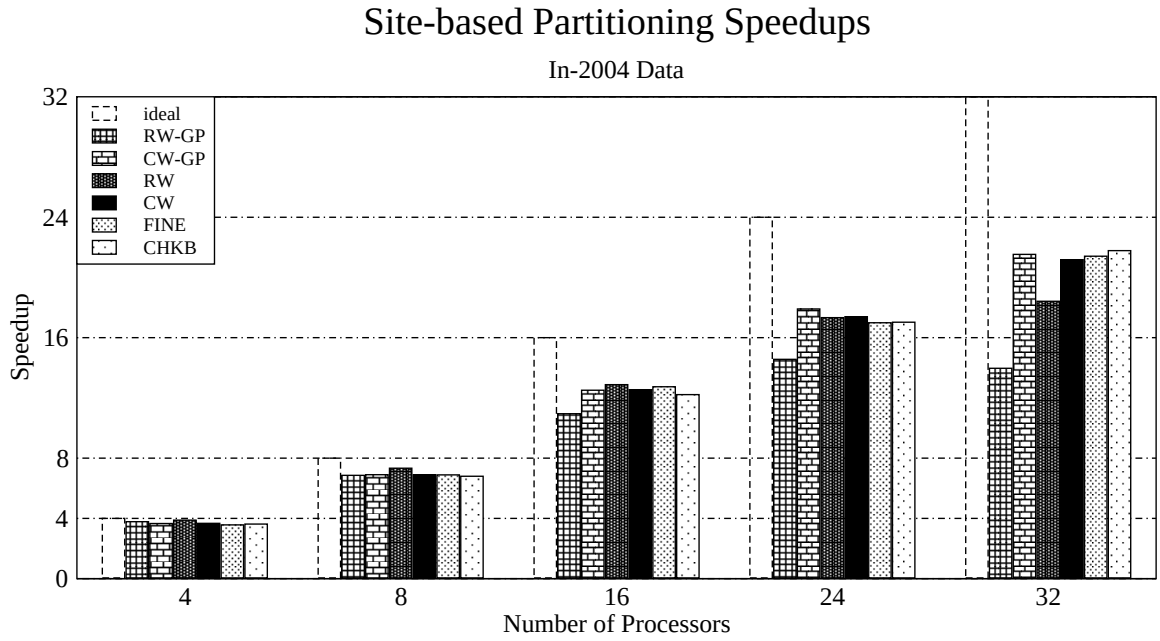


Figure 6.23: Average speedups for site-based partitionings of `in-2004` data.

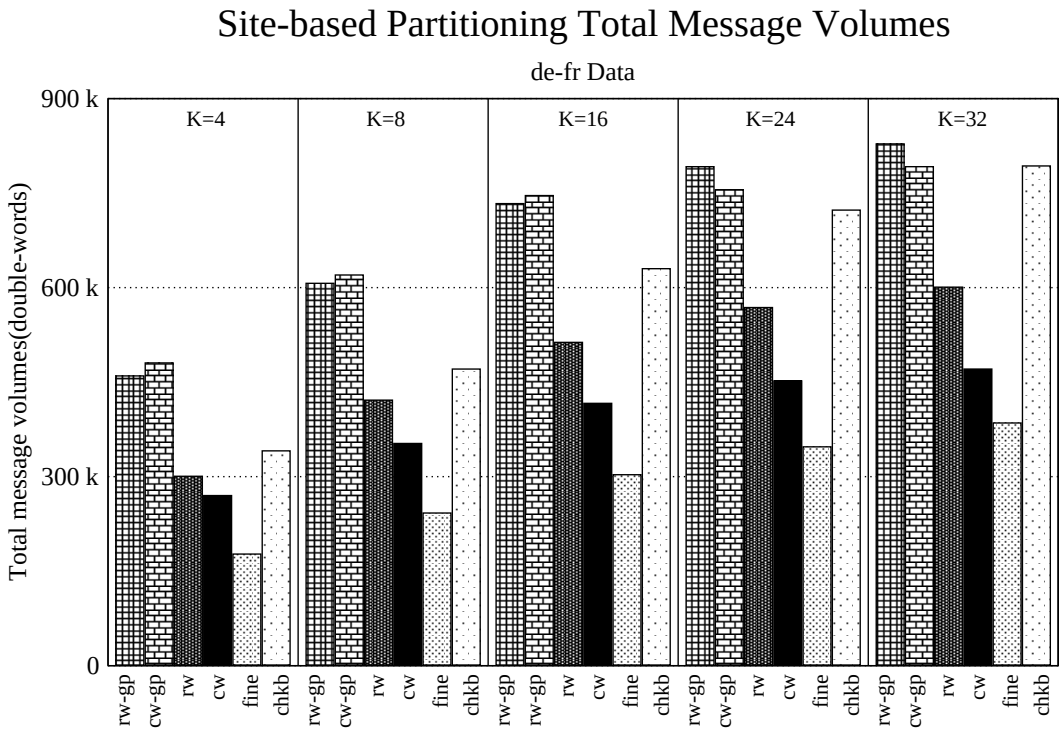


Figure 6.24: Average total communication volumes for site-based partitionings of `de-fr` data.

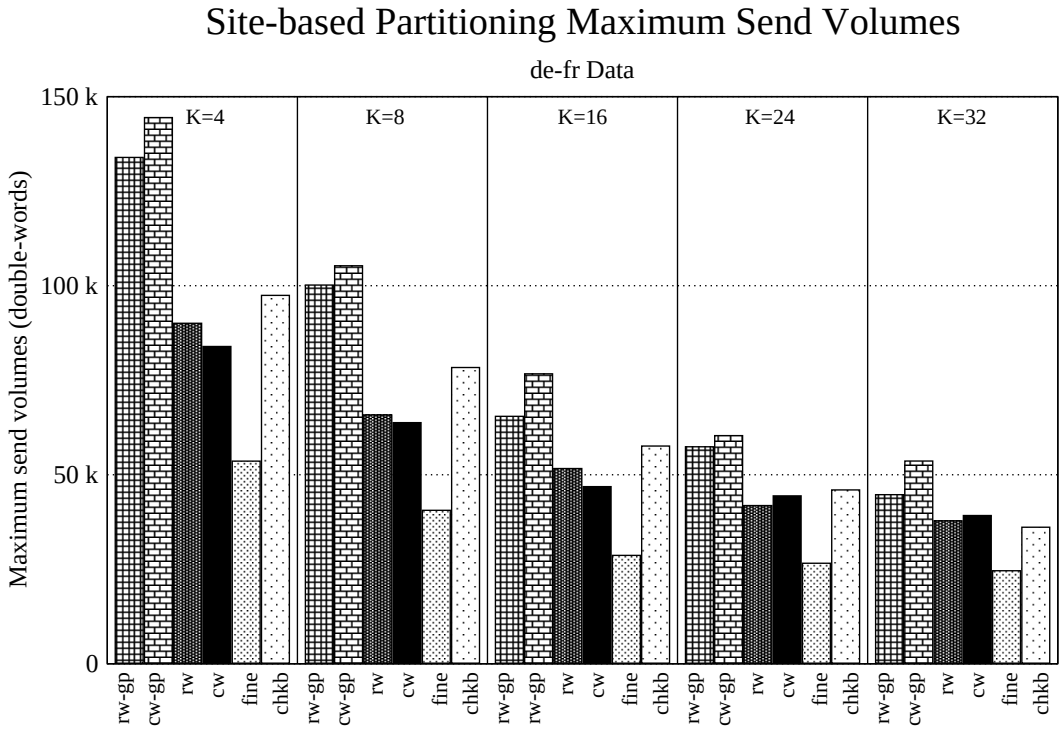


Figure 6.25: Average maximum per-processor communication volumes for site-based partitionings of `de-fr` data.

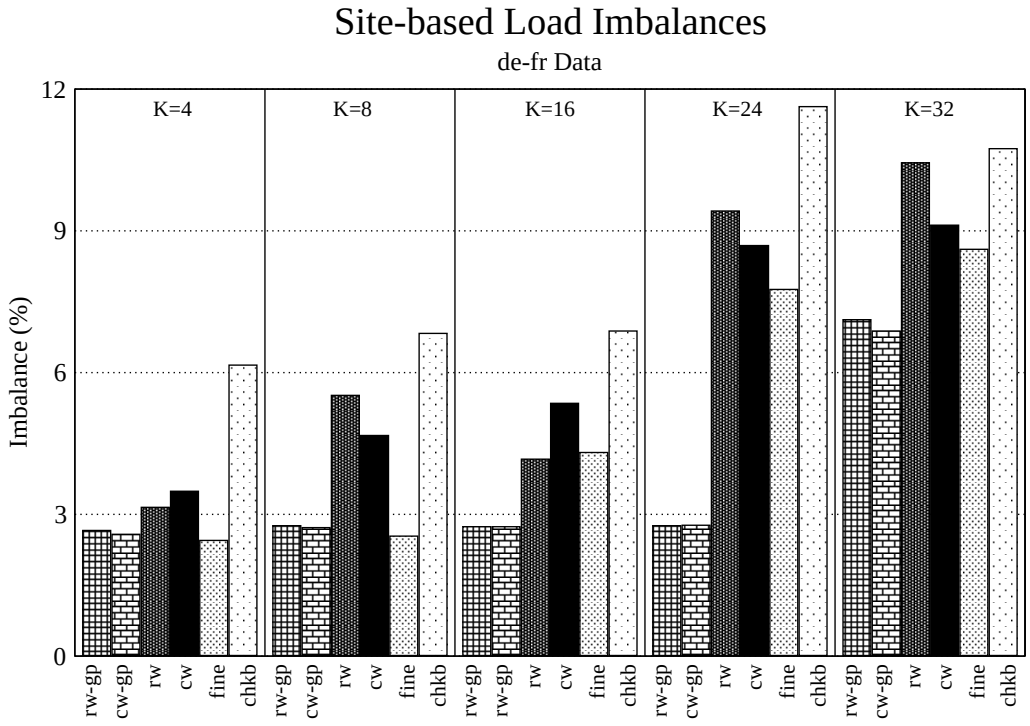


Figure 6.26: Average load imbalances for site-based partitionings of `de-fr` data.

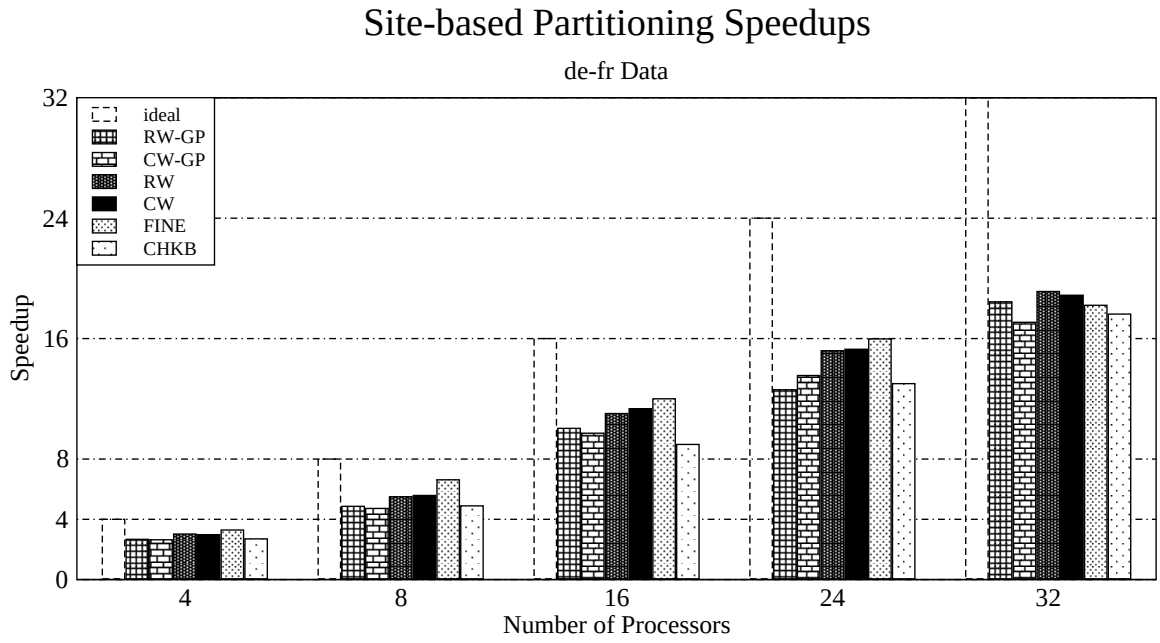


Figure 6.27: Average speedups for site-based partitionings of `de-fr` data.

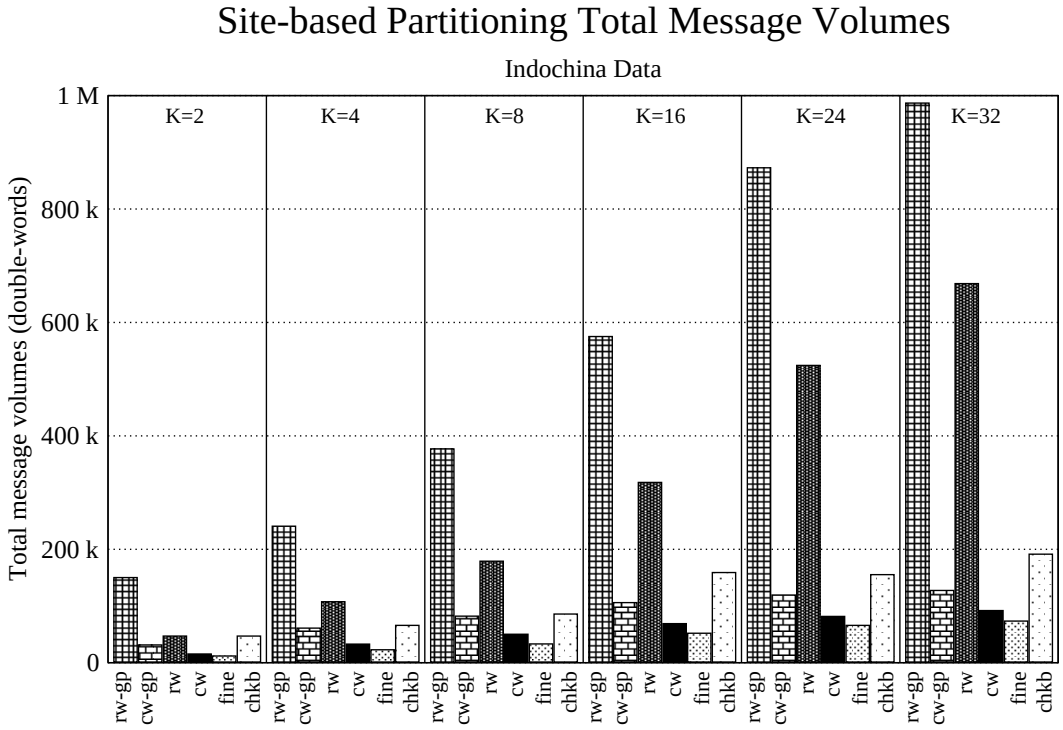


Figure 6.28: Average total communication volumes for site-based partitionings of `indochina` data.

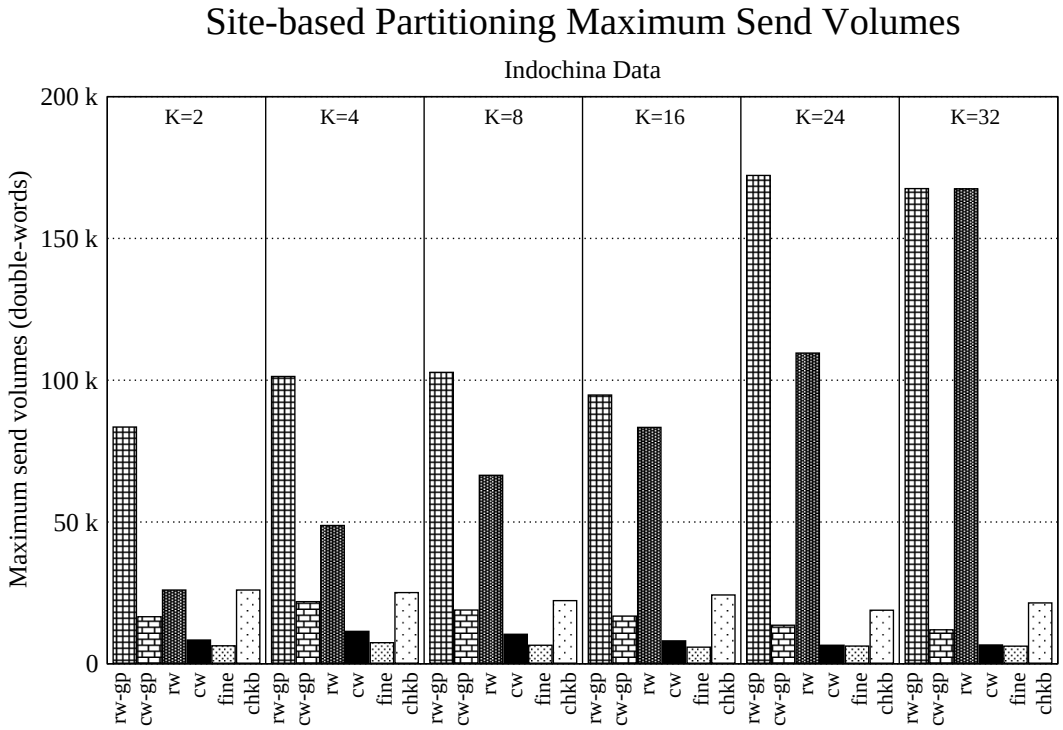


Figure 6.29: Average maximum per-processor communication volumes for site-based partitionings of `indochina` data.

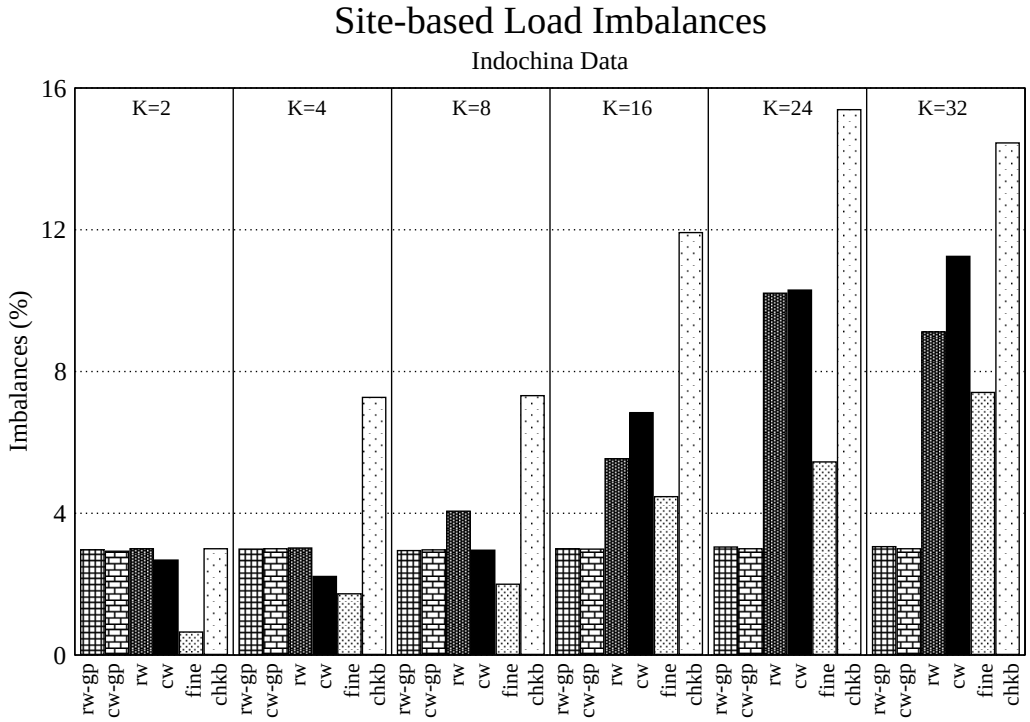


Figure 6.30: Average load imbalances for site-based partitionings of `indochina` data.

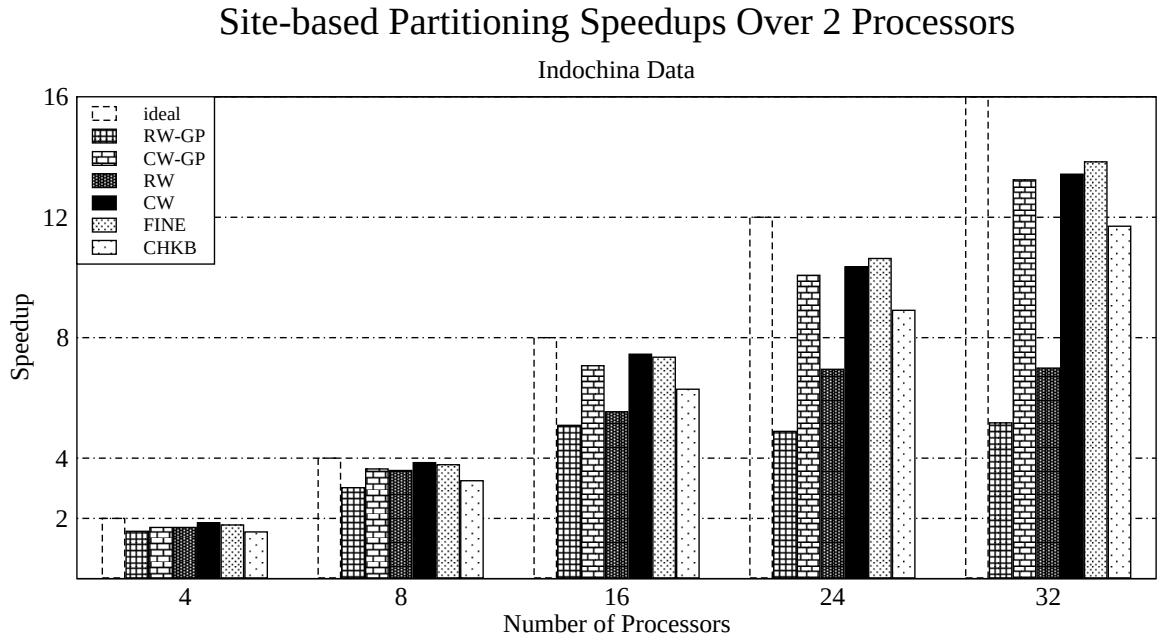


Figure 6.31: Average speedups for site-based partitionings of `indochina` data.

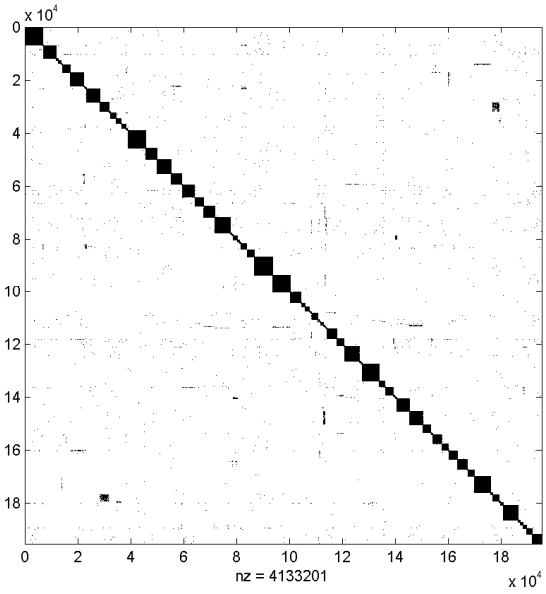


Figure 6.32: Upper-left corner of site-ordered `Google` data transition matrix.

6.3.3 Site-based ordering and Cache Coherency

Cache coherency is an important factor that dramatically affects the running time of PageRank computation. Since most of the links are intra-site links for the Web, site-based ordering of the transition matrix increases the cache coherency, hence reduces the running time of PageRank computation. By site-based ordering, we mean the ordering of transition matrix, so that pages inside a site corresponds to consecutive rows and columns of the matrix. Figure 6.32 shows the upper-left corner non-zero pattern of the site-ordered transition matrix of `Google` data. Diagonal blocks of site-ordered matrix, which corresponds to intra-site links, is denser.

Parallel computation of PageRank with an unordered matrix may cause observing superlinear speedup values, since sequential execution with unordered matrix is slower. As the number of processors increase, cache becomes more coherent for site-based partitioning. Figure 6.33 depicts the superlinear speedups obtained by multiplication of random ordered matrix and speedups for site-ordered matrix for HP-based rowwise decomposition of `Google` data transition matrix. The speedups we provided in previous pages are obtained by multiplying site-ordered matrices. We do not report superlinear speedups for this work.

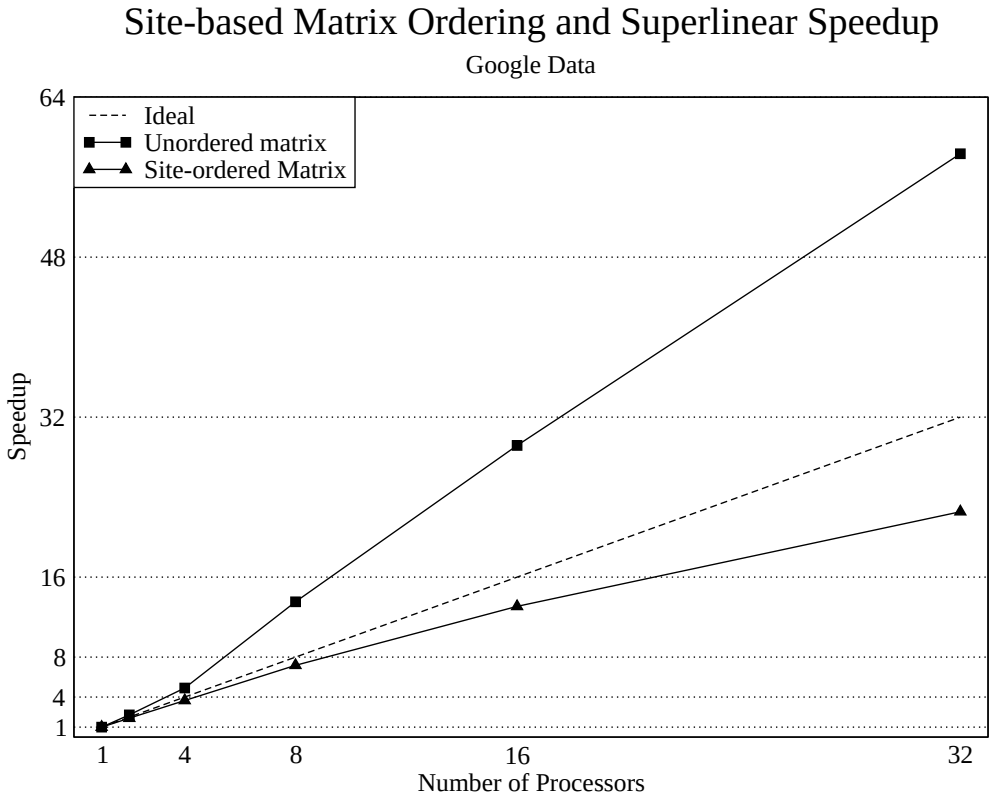


Figure 6.33: Speedups for HP-based rowwise model of `Google` data with site-ordered and unordered transition matrices.

Chapter 7

Conclusion and Future Work

In this thesis, we have mainly focused on reducing the preprocessing overhead before the parallel PageRank computations without degrading the parallel computation time. To achieve this goal, we have proposed Web-site-based graph and hypergraph partitioning models for rowwise, columnwise, fine-grain and checkerboard workload partitioning. We do not only partition the workload incurred by matrix-vector multiplication, which is the key operation for PageRank computation, but we partition the workload of overall iterative PageRank algorithm. The models correctly handle the pages without incoming links, for which we prevent extra communication. We have presented our models for the power method, which is most widely used for computing PageRank, but models can be applied to other iterative PageRank computation methods.

Compared with a previous work which employs rowwise and fine-grain hypergraph partitioning models for parallel PageRank computation [13], our site-based hypergraph partitioning models provide much smaller preprocessing overhead with comparable parallel PageRank computation time. Graph-partitioning-based models achieve even smaller preprocessing time, with higher communication overhead during parallel PageRank computation. Columnwise partitioning models usually obtain lower communication volume than rowwise models. Fine-grain partitioning models are well-suited for PageRank transition matrices which require high communication volume during parallel computation. Checkerboard

partitioning is advisable for PageRank computation on large number of processors.

Although the proposed models achieve fast parallel PageRank computation with a low preprocessing time, there is still room for improvement. We restrict the solution space with sites in site-based models. Hence, sites containing large number of pages may cause load imbalance. We work on models to cope with the load imbalance problem caused by large-size sites.

State-of-the-art graph and hypergraph partitioning algorithms may fail to achieve desired performance for power-law graphs, such as Web graphs [7, 23]. This is mostly caused by the failure of coarsening algorithms in multilevel partitioning paradigm [7]. During our experiments, we observed that partitioning quality dramatically changes according to the coarsening algorithm used in kPaToH. Site-based clustering performs better than most of the coarsening algorithms. In other words, for most the coarsening algorithms used for hypergraph partitioning, site-based models provide lower communication volume than page-based models. For better partitioning of large power-law transition matrix, coarsening algorithms for hypergraph partitioning may be improved.

Bibliography

[1] Larbin home page. URL reference: `http://larbin.sourceforge.net/index-eng.html/`.

[2] Google Programming Contest. URL reference: `http://www.google.com/programming-contest/`, 2004.

[3] Google Technology. URL reference: `http://www.google.com/technology`, 2004.

[4] Laboratory for Web Algorithmics. URL reference: `http://law.dsi.unimi.it/`, 2006.

[5] Web Graph Benchmark. URL reference: `http://hipercom.inria.fr/~viennot/webgraph/`, 2006.

[6] Webgraph. URL reference: `http://webgraph.dsi.unimi.it/`, 2006.

[7] A. Abou-rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. Technical Report TR 05-034, Multilevel Algorithms for Partitioning Power-Law Graphs, Department of Computer Science and Engineering, October 2005.

[8] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. Pagerank computation and the structure of the web: Experiments and algorithms. In *Proc. of the 11th International Conference on World Wide Web, Alternate Poster Tracks*, 2002.

[9] C. Aykanat, B. B. Cambazoğlu, and B. Uçar. Multi-level hypergraph partitioning with multiple constraints and fixed vertices. Submitted to the Journal of Parallel and Distributed Computing.

[10] C. Aykanat, A. Pınar, and Ü. V. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing*, 25(6):1860–1879, 2004.

[11] P. Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1):73–120, 7 2005.

[12] M. Bianchini, M. Gori, and F. Scarselli. Inside pagerank. *ACM Transactions on Internet Technology*, February 2005.

[13] J. Bradley, D. Jager, W. Knottenbelt, and A. Trifunovic. Hypergraph partitioning for faster parallel pagerank computation. *Lecture Notes in Computer Science*, 3670:155–171, 2005.

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

[15] T. N. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, Philadelphia, 1993.

[16] B. B. Cambazoglu. *Models and Algorithms for Parallel Text Retrieval*. PhD thesis, Bilkent University, Department of Computer Engineering, January 2006.

[17] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, 1999.

[18] Ü. V. Çatalyürek and C. Aykanat. Patoh: A multilevel hypergraph partitioning tool, version 3.0. Technical Report BU-CE-9915, Bilkent University, Department of Computer Engineering, 1999.

[19] Ü. V. Çatalyürek and C. Aykanat. A fine grain hypergraph model for 2d decomposition of sparse matrices. In *Proc. 8th International Workshop on Solving Irregularly Structured Problems in Parallel*, San Fransisco, USA, April 2001.

[20] Ü. V. Çatalyürek and C. Aykanat. A hypergraph partitioning approach for coarse grain decomposition. In *Proc. of Scientific Computing 2001*, pages 10–16, Denver, Colorado, November 2001.

[21] A. Cevahir, C. Aykanat, A. Türk, and B. B. Cambazoğlu. Web-site-based partitioning techniques for reducing the preprocessing overhead before the parallel pagerank computations. In *Workshop on State-of-the-Art in Scientific and Parallel Computing, PARA'06*, Umea, Sweden, June 2006.

[22] Y. Chen, Q. Gan, and T. Suel. I/O efficient techniques for computing pagerank. In *Proc. 11th Conf. Information and Knowledge Management*, pages 549–557, 2002.

[23] D. Gleich, L. Zhukov, and P. Berkhin. Fast parallel pagerank: A linear system approach. Technical Report YRL-2004-038, Yahoo!, 2004.

[24] J.-L. Guillaume, M. Latapy, and L. Viennot. Efficient and simple encodings for the web graph. In *Proc. of the 11th international conference on the World Wide Web*, 2002.

[25] A. B. Gürdağ. A parallel implementation of a link-based ranking algorithm for web search engines. Master's thesis, Boğaziçi University, 2002.

[26] T. Haveliwala. Efficient computation of pagerank. Technical Report 1999-31, Stanford Digital Library Project, 1999.

[27] T. Haveliwala. Efficient encodings for document ranking vectors. Technical report, Stanford University, 2002.

[28] B. Hendrickson, R. Leland, and S. Plimpton. An efficient parallel algorithm for matrix-vector multiplication. *International Journal of High Speed Computing (IJHSC)*, 7(1):73–88, 1995.

[29] B. A. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? *Lecture Notes in Computer Science*, 1457:218–225, 1998.

[30] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for computation of pagerank. In *Proc. Int'l Conf. on the Numerical Solution of Markov Chains*, 2003.

[31] S. Kamvar, T. Haveliwala, and G. Golub. Extrapolation methods for accelerating pagerank computations. In *Proc. 12th Int'l World Wide Web Conf.*, pages 261–270, 2003.

[32] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank. Technical report, Stanford University, 2003.

[33] G. Karypis and V. Kumar. *MeTiS: A software package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0*. University of Minnesota, Department of Computer Science / Army HPC Research Center, Minneapolis, September 1998.

[34] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[35] G. Karypis, V. Kumar, R. Aggarwal, and S. Shekhar. *hMeTiS a hypergraph partitioning package version 1.0.1*. University of Minnesota, Department of Computer Science / Army HPC Research Center, Minneapolis, 1998.

[36] J. Kleinberg. Authoritive sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[37] A. Langville and C. Meyer. Updating pagerank with iterative aggregation. In *Proc. the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, pages 392–393, New York, 2004. ACM Press.

[38] A. Langville and C. Meyer. Deeper inside pagerank. *Internet Mathematics*, 1(3):335–380, 2005.

[39] A. Langville and C. Meyer. A survey of eigenvector methods of web information retrieval. SIAM R*eview*, 47(1):135–161, 2005.

[40] R. Lempel and S. Moran. The stochastic approach for link structure analysis (salsa) and the tkc effect. In *Proc. of the 9th International Conference on World Wide Web*, pages 387–401, Amsterdam, 2000. North Holland Publishing Co.

[41] R. Lempel and S. Moran. Salsa: The stochastic approach for link-structre analysis. *ACM Transactions on Information Systems*, 19(2):131–160, 2001.

[42] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley-Teubner, Chichester, U.K., 1990.

[43] B. Manaskasemsak and A. Rungsawang. Parallel pagerank computation on a gigabit pc cluster. In *Proc. 18th International Conference on Advanced Information Networking and Application (AINA'04)*, pages 273–277. IEEE, 2004.

[44] C. Moler. The world's largest matrix computation. *Matlab News and Notes*, pages 12–13, October 2002.

[45] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[46] K. Sankaralingam, S. Sethumadhavan, and J. Browne. Distributed pagerank for p2p systems. In *Proc. 12th IEEE Int'l Symp. High Performance Distributed Computing*, pages 58–69, 2003.

[47] A. Trifunovic and W. J. Knottenbelt. Par*k*way2.0: A parallel multilevel hypergraph partitioning tool. In C. Aykanat, T. Dayar, and İ. Körpeoğlu, editors, *Proc. 19th International Symposium on Computer and Information Sciences*, volume 3280 of *Lecture Notes in Computer Science*, pages 789–800, Antalya, Turkey, October 2004. Springer.

[48] B. Uçar. Parallel sparse matrix multiplies and iterative solvers. PhD Thesis Presentation, 2005.

[49] B. Uçar. *Parallel Sparse Matrix-Vector Multiplies and Iterative Solvers*. PhD thesis, Bilkent University, Department of Computer Engineering, August 2005.

[50] B. Uçar and C. Aykanat. Parmxvlib: A parallel library for sparse-matrix vector multiplies. In *Proc. 7th World Multiconference on Systemics, Cybernetics and Informatics*, pages 393–398, 2003.

[51] B. Uçar and C. Aykanat. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for matrix-vector multiplies. *SIAM Journal on Scientific Computing*, 25(6):1837–1859, 2004.

[52] B. Uçar and C. Aykanat. A library for parallel sparse matrix vector multiplies. Technical Report BU-CE-0506, Bilkent University, Department of Computer Engineering, August 2005.

[53] Y. Wang and D. J. Dewitt. Computing pagerank in a distributed internet search system. In *Proc. of the 30th VLDB Conference*, Toronto, Canada, 2004.

[54] Y. Zhu, S. Ye, and X. Li. Distributed pagerank computation based on iterative aggregation-disaggregation methods. In *Proc. 14th ACM international Conference on Information and Knowledge Management.*