2007－HPC－112 (4)
2007／9／9

Web-Site-Based Partitioning Techniques for Efficient Parallelization of the PageRank Computation

ALI CEVAHIR† and SATOSHI MATSUOKA†,††

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 enourmous size of the Web matrix to be multiplied, PageRank computations are usually carried out on parallel systems. Graph and hypergraph partitioning techniques are widely used for efficient parallelization of matrix-vector multiplications. These techniques suffer from high preprocessing overhead for PageRank algorithm. In this work, we propose Web-site-based partitioning techniques to reduce the preprocessing overhead of Parallel PageRank computation.

1. Introduction

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[17]. Many researchers proposed different acceleration techniques after the proposal of the basic model. Algorithmic/numeric optimizations that try to reduce the number of iterations[11),12),14)] and I/O efficient out-of-core algorithms for reducing the disk swapping time for single processor[7),10)] are some of the proposed techniques for improving the PageRank computation performance. PageRank should be calculated repeatedly with the change of the Web. 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[8),16)].

The focus of this work is on reducing the per iteration time through parallelization. Among several PageRank formulations[11),12),14)] widely used formulation of Kamvar et al.[11)] is selected for parallelization. In this formulation, which is based on the power method, the kernel operations are sparse-matrix vector multiply and linear vector operations. We are going to deal with row-parallel[19)] implementation of the PageRank algorithm.

Recently Bradley et al.[2)] utilized the hypergraph-partitioning-based models proposed by Catalyurek and Aykanat[3),4)] directly for parallel PageRank computation. Unfortunately, because of the huge size of the Web graph, hypergraph-partitioning-based models are not scalable, when applied directly over the Web matrix. Even though the computations reported in[2)] are fairly fast; the preprocessing time for partitioning takes even longer than the sequential PageRank computation. To avoid this problem, Cevahir et al.[6)] suggested site-based hypergraph partitioning models which reduces the sizes of the hypergraphs used in partitioning, considerably. In addition to reduced preprocessing time, they offer parallelization of 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 Bradley et al. only consider matrix-vector multiplies. The proposed site-based partitioning scheme reduces the preprocessing time drastically compared to the page-based scheme while producing better partitions in terms of communication volume.

In[6)], preprocessing time is reduced by partitioning site-to-page iteration matrix, instead of partitioning page-to-page matrix. In this work, we demonstrate that the preprocessing time can be further decreased by compressing the transition matrix site-to-site. Unfortunately, by partitioning the site-to-site com-

† Tokyo Institute of Technology
†† National Institute of Informatics

pressed iteration matrix, we are unable to minimize the correct metric for total communication volume of the PageRank iterations, but an approximation. Nevertheless, our experiments of both site-to-page and site-to-site compression schemes on TSUBAME supercomputer demonstrate that site-to-site partitioning is still viable on parallel computers with high speed networks.

The rest of the paper is organized as follows. In Section 2, background information is given for PageRank algorithm. Section 3 explains the parallel PageRank algorithm used. Site-based partitioning models are explained in Section 4. Experimental results are explained in Section 5. Section 6 concludes the paper and make discussions of some future work.

2. PageRank Algorithm

PageRank can be explained with a probabilistic model, called random surfer model[17]. Consider an Internet surfer randomly visiting pages by following links within pages or jumping to a random page. Let the surfer visit page i at a particular time step. In the next time step, the surfer chooses to visit one of the pages pointed by the links of page i at random or visit a random page within the Web. If page i is a dangling page (a page without links), then the only way for surfer to leave page i is to jump to a random page. There is a fix probability of randomly jumping to a page instead of following the links within pages.

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

A row-stochastic transition matrix $\mathbf{P'}$ is constructed from $\mathbf{P}$ as $\mathbf{P'} = \mathbf{P} + \mathbf{dt}^T$ to handle dangling pages according to the random surfer model. Here, $\mathbf{d} = (d_i)$ and $\mathbf{t} = (t_i)$ are column vectors of size m. $\mathbf{d}$ identifies dangling pages, i.e., $d_i = 1$ if row i of $\mathbf{P}$ corresponds to a dangling page, and 0, otherwise. $\mathbf{t}$ is the teleportation vector which denotes the probability distribution of destination pages for a random jump. Generally, uniform teleportation vector $\mathbf{t}$, where $t_i = 1/m$ for all i, is used for PageRank computation. However, non-uniform teleportation vectors can be used for personalized PageRank computation[17].

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

Fig. 1 Power method solution for PageRank: $\mathbf{A} = \mathbf{P}^T$ is the transition matrix, $\mathbf{v}$ is the teleportation vector and ε is the convergence threshold.

Although $\mathbf{P'}$ is row-stochastic, it may not be irreducible. An irreducible Markov matrix $\mathbf{P''}$ is constructed as $\mathbf{P''} = \alpha\mathbf{P'} + (1-\alpha)\mathbf{e}\mathbf{t}^T$, where $\mathbf{e}$ is a column vector of size m containing all ones. Here, α represents the probability that the surfer chooses to follow one of the links of the current page, and $(1 - \alpha)$ represents the probability that surfer makes a random jump instead of following the links.

Given $\mathbf{P''}$, PageRank vector $\mathbf{r}$ can be determined by solving the equation $(\mathbf{P''})^T\mathbf{r} = \mathbf{r}$. That is, by finding the principal eigenvector of matrix $\mathbf{P''}$. Applying the power method directly for the solution of this eigenvector problem leads to a sequence of matrix-vector multiplies $\mathbf{p}^{k+1} = (\mathbf{P''})^T\mathbf{p}^k$, where $\mathbf{p}^k$ is the kth iterate towards the PageRank vector $\mathbf{r}$. However, matrix $\mathbf{P''}$ is completely dense, whereas original $\mathbf{P}$ is sparse. Fortunately, by reformulating dense matrix multiplication in terms of sparse matrix computation, Kamvar et al.'s[11] algorithm given in Fig. 1 can efficiently compute PageRank refraining any dense matrix operations.

3. Parallel PageRank Algorithm

We consider the parallelization of the computations of the PageRank algorithm given in Fig. 1 through rowwise partitioning of the $\mathbf{A}$ matrix as $\mathbf{A} = [\mathbf{A}_1^T \cdots \mathbf{A}_k^T \cdots \mathbf{A}_K^T]^T$, where pro-

Parallel-PageRank($\mathbf{A}_k$, $\mathbf{v}_k$)

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

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

cessor P_k stores row stripe $\mathbf{A}_k$. All vectors (e.g., $\mathbf{p}$ and $\mathbf{q}$) used in the algorithm are partitioned conformably with the row partition of $\mathbf{A}$ to avoid communication of the vector components during linear vector operations. That is, the $\mathbf{p}$ and $\mathbf{q}$ vectors are partitioned as $[\mathbf{p}_1^T \cdots \mathbf{p}_K^T]^T$ and $[\mathbf{q}_1^T \cdots \mathbf{q}_K^T]^T$, respectively. Processor P_k is responsible for performing the local matrix-vector multiply $\mathbf{q}_k \leftarrow \alpha \mathbf{A}_k \mathbf{p}$ while holding $\mathbf{p}_k$. Processor P_k is also responsible for the linear vector operations on the kth blocks of the vectors.

In this work, we are going to work on partitioning models for the parallel PageRank algorithm given in Fig. 2[6]. The algorithm is called coarse-grain parallel PageRank algorithm, since it reduces the number of global communication operations at each iteration from two to one by rearranging the computations. Namely, two global norms are accumulated at all processors in a single all-to-all reduction operation performed at step 5(c) in Fig. 2. Hence, the proposed coarse-grain formulation halves the latency overhead while keeping the communication volume the same. In Fig. 2, a superscript k denotes the partial result computed by processor P_k, e.g., γ^k is the partial result for global scalar γ, where $\gamma = \sum_{k=1}^{K} \gamma^k$. *Expand* is the multicast-like operation to exchange the $\mathbf{p}_k$ vector entries before the row-parallel sparse matrix-vector multiplication[19].

4. Partitioning Models

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

Rowwise partitioning of irregularly sparse matrices for the parallelization of matrix-vector multiplies is formulated using the hypergraph-partitioning model[3),4)]. In the column-net model proposed for rowwise partitioning[3),4)], a given matrix is represented as a hypergraph which contains a vertex for each row and a net for each column. The net corresponding to a column connects the vertices corresponding to the rows that have a non-zero at that column. Vertices are associated with weights which are set equal to the number of non-zeros in the respective rows. A K-way vertex partition on the hypergraph is decoded as assigning the rows corresponding to the vertices in each part of the partition to a distinct processor. Partitioning constraint on balancing the part weights corresponds to balancing the computational loads of processors, whereas partitioning objective of minimizing the cutsize corresponds to minimizing the total communication volume during a parallel matrix-vector multiply.

Page-based and site-based hypergraph partitioning models, considering the whole iterative PageRank algorithm, are discussed in[6]. It is observed that partitioning site-to-page compressed iteration matrix, instead of original page-to-page matrix drastically reduces the preprocessing time, while achieving better communication volume values.

In the site-based hypergraph model, a compressed version, $\bar{\mathbf{A}}$, of the page-to-page matrix $\mathbf{A}$ is partitioned. Site-to-page $\bar{\mathbf{A}}$ matrix is generated exploiting the fact that Web sites form a natural clustering of pages. In matrix $\bar{\mathbf{A}}$, rows correspond to Web sites while columns corresponds to pages. The union of the non-zeros of the $\mathbf{A}$-rows that correspond to the pages residing in a site form the non-zeros of the $\bar{\mathbf{A}}$-row corresponding to that site. Then, column-net hypergraph partitioning model[4)] is applied on $\bar{\mathbf{A}}$, and partition the site-based hypergraph $\mathcal{H}(\bar{\mathbf{A}})$ instead of $\mathcal{H}(\mathbf{A})$. In $\mathcal{H}(\bar{\mathbf{A}})$, the number of vertices is equal to the number of sites and the number of nets is equal to the number of pages. To encapsulate the weight of the whole iteration, the weight of a vertex j in $\mathcal{H}(\bar{\mathbf{A}})$ cor-

responding to site j is set equal to

$$\sum_{\text{page i} \in \text{site j}} (2 \times nnz(\text{row i of } [\mathbf{A}]) + 7),$$

where nnz means number of nonzeros. Here, the first term accounts for the number of flops of the matrix-vector multiplication and the second term accounts for the vector operations.

The preprocessing time can be further improved by compressing the matrix $\mathbf{A}$ in both dimensions and partitioning this matrix. To achieve this goal, we generate site-to-site matrix $\bar{\bar{\mathbf{A}}}$. The advantage of site-to-page compression over site-to-site compression is, column-net hypergraph partitioning model for the matrix $\bar{\mathbf{A}}$ correctly encapsulates the total communication volume during PageRank computation. Since the sparsity pattern of $\bar{\bar{\mathbf{A}}}$ does not correctly summarize the communication requirement of the PageRank computation, advantage of the hypergraph-partitioning models is lost for partitioning $\bar{\bar{\mathbf{A}}}$ matrix. However, with the proper edge costs, graph-partitioning model can be adopted with the same approximation factor as it achieves in the page-based partitioning. For rowwise graph-partitioning, the weight of the vertex j of the model graph $\mathcal{G}(\bar{\bar{\mathbf{A}}})$ of site-to-site matrix $\bar{\bar{\mathbf{A}}}$ is the same as the vertex weight of the vertex j of $\mathcal{H}(\bar{\mathbf{A}})$. Edge cost of edge e_{ij}, which is the edge corresponding to the edge between vertex i and j, is computed as the sum of links between site i and site j.

5. Experimental Results

Experimental results presented in[6] prove that hypergraph partitioning model for site-to-page compressed matrix drastically reduces the preprocessing time, so that preprocessing takes time equal to only a few sequential PageRank iterations. Experiments presented in this section demonstrates that site-to-site compression achieves less preprocessing time with slightly bad total communication volume. Hence, site-to-site compression is still viable with high speed networked supercomputers, such as TSUBAME. In our experiments, two datasets with different sizes are used. The `Google`☆ dataset is provided by Google and includes .edu domain pages in the US. The `Balkan` dataset is crawled by Larbin crawler☆☆and includes pages from the Web of Balkan countries. The properties of these datasets are given in Table 1. The convergence threshold is set to $\epsilon = 10^{-8}$ and the damping factor α is set to 0.85 during computations .

Table 1 Properties of datasets.

	`Google`	`In-2004`
# of pages	913,569	771,895
# of sites	15,819	37,722
# of links	4,480,218	31,794,117

Direct K-way hypergraph partitioning tool kPaToH[1),5)] is used, with default parameters and an imbalance tolerance of 3%, for partitioning the hypergraphs. Graph partitioning tool MeTiS is used for partitioning graphs, with default parameters.

Fig. 3 displays the variation of the preprocessing times of site-to-page and site-to-site compression schemes with increasing number of processors. Preprocessing time involves both compression and partitioning times. It has been shown that site-based hypergraph partitioning scheme achieve drastic reduction in the preprocessing time compared to the page-based scheme[6]. As seen in Fig. 3, partitioning site-to-site compressed matrix achieves even better preprocessing time compared to hypergraph partitioning model for site-to-page compression scheme. Per iteration time of sequential PageRank algorithm is 85.5 ms for `Google` dataset and 273.5 ms for `Balkan` dataset. Hence, preprocessing time required for site-to-page compression scheme is equal to only a few sequential iterations, whereas it is less than two iterations for site-to-site compression scheme.

Fig. 4 displays the quality of the partitions obtained by the site-to-page and site-to-site matrix partitioning schemes in terms of total communication volume. As seen in the figure, the partitions obtained by hypergraph partitioning model for site-to-page matrix incurs significantly less communication volume than those of the graph partitioning model for site-to-site matrix, for both of the datasets. Note that, it was already observed that site-based partitioning achieves less total communication volumes than page-based partitioning since sites consti-

☆ `http://www.google.com/programming-contest/`

☆☆ `http://larbin.sourceforge.net/index-eng.html/`

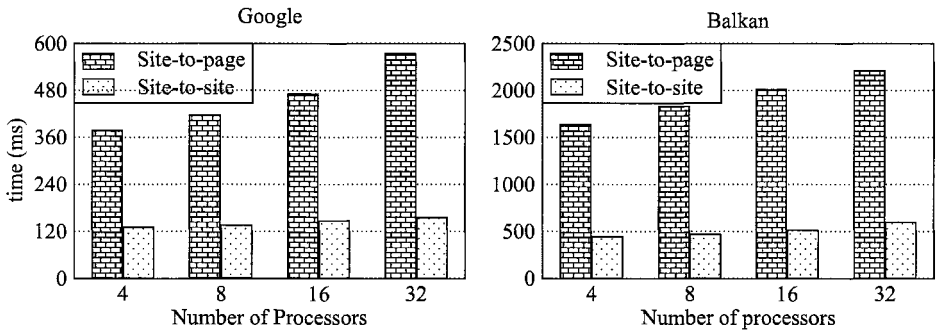

Fig. 3 Preprocessing times in miliseconds.

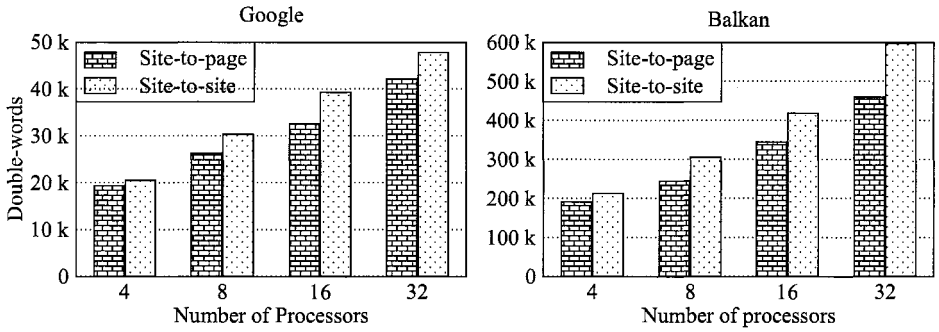

Fig. 4 Total communication volumes in Kbytes.

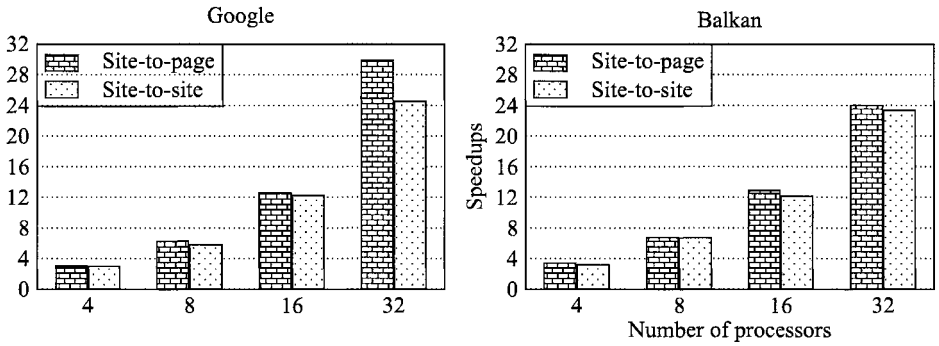

Fig. 5 Speedup curves.

tute natural clusters of pages[6].

Paralel sparse matrix-vector multiplication library ParMxvLib[18] is used for implementation of the parallel PageRank algorithm. Experiments are held using TSUBAME supercomputer in TITECH. TSUBAME includes 16 processors for each computing node. Experiments are held using one processor from each node. Note that, we have given speedups upto 32 processors. Load imbalance problem occurs with greater number of processors, because of the sites that contain large number of pages.

The speedup curves are given in Fig. 5. As seen in the figure, the site-to-page partitioning scheme leads to slightly higher speedup values than the site-to-site partitioning scheme, in accordance with the reduction in the communication volumes. TSUBAME has high speed in-

fiband communication network, therefore total communication volume has less effect than it has for megabit or gigabit PC clusters. Therefore site-to-site compression scheme can be applied for matrix partitioning as an alternative for site-to-page partitioning scheme since it achives less preprocessing time with the drawback of worse speedup values.

6. Conclusion and future work

In this work, we have compared performances of partitioning site-to-page and site-to-site compressed iteration matrices of PageRank algorithm for efficient parallelization using TSUBAME supercomputer. Experimental results confirm that site-based partitioning achieve reasonable preprocessing time for PageRank. Despite its worse performance in parallelization, site-to-site compression is still valuable for parallel systems with high speed networks, since it achieves considerably less preprocessing time.

One disadvantage of the site-based schemes is the load imbalance problem due to very large sites. Actually, the datasets are relatively small when compared to computation power of the parallel system used. The algorithm is expected to be more scalable with bigger datasets. However, load imbalance still remains to be a problem to be solved. We plan to attack this problem in future. Also, with the increasing number of processors, communication between processes becomes a bottleneck. Developing scalable parallel algorithms for next generation supercomputers remains as another future work.

References

1) Aykanat, C., Cambazoglu B.B., Ucar, B.: Multilevel hypergraph partitioning with multiple constraints and fixed vertices. Submitted to J. Parallel and Distributed Computing.
2) Bradley, J.T., Jager, D.V., Knottenbelt, W.J., Trifunovic, A.: Hypergraph partitioning for faster parallel PageRank computation. Lecture Notes in Computer Science **3670** (2005) 155–171
3) Catalyurek, U.V., Aykanat, C.: Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. Lecture Notes in Computer Science **1117** (1996) 75–86
4) Catalyurek, U.V., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Transactions on Parallel and Distributed Systems **10**(7) (1999) 673–693
5) Catalyurek U.V., Aykanat, C.: A multilevel hypergraph partitioning tool, version 3.0. Tech. Rep., Bilkent University. (1999)
6) Cevahir, A., Aykanat, C., Turk, A., Cambazoglu B.B.: A web-site-base partitioning technique for reducing preprocessing overhead of parallel PageRank computation. In: Proc. Workshop on State-of-the-Art in Scientific and Parallel Computing, PARA06 (2006)
7) Chen, Y., Gan, Q., Suel, T.: I/O efficient techniques for computing PageRank. In: Proc. 11th Conf. Information and Knowlegde Management. (2002) 549–557
8) Gleich, D., Zhukov, L., Berkhin, P.: Fast parallel PageRank: A linear system approach. Tech. Rep. YRL-2004-038, Yahoo!. (2004)
9) Gyöngyi, Z., Garcia-Molina, H., Pedersen, J.: Combating Web spam with TrustRank. In: Proc. 30th Int'l Conf. on VLDB. (2004) 576–587
10) Haveliwala, T.: Efficient computation of PageRank. Tech. Rep. 1999-31, Stanford Digital Library Project. (1999)
11) Kamvar, S., Haveliwala, T., Manning, C., Golub, G.: Extrapolation methods for accelerating PageRank computations. In: Proc. 12th Int'l WWW Conf. (2003) 261–270
12) Kamvar, S., Haveliwala, T., Golub, G.: Adaptive methods for computation of PageRank. In: Proc. Int'l Conf. on the Numerical Solution of Markov Chains. (2003)
13) Kamvar, S., Haveliwala, T., Manning, C., Golub, G.: Exploiting the block structure of the Web for computing PageRank. Tech. Rep., Stanford Univ. (2003)
14) Langville, A.N., Meyer, C.D.: Deeper inside PageRank. Internet Mathematics **1**(3) (2005) 335–380
15) Langville, A.N., Meyer, C.D.: A reordering for the PageRank problem. SIAM J. Scientific Computing **27**(6) (2006) 2112–2120
16) Manaskasemsak, B., Rungsawang, A.: Parallel PageRank computation on a gigabit PC cluster. In: Proc. AINA'04. (2004) 273–277
17) Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the Web. Tech. Rep. 1999-66, Stanford Univ. (1999)
18) Ucar, B., Aykanat, C.: A library for parallel sparse matrix-vector multiplies. Tech. Rep. BU-CE-0506, Department of Computer Engineering, Bilkent University, Ankara, Turkey (2005)
19) Ucar, B., Aykanat, C.: Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for matrix-vector multiplies. SIAM J. Scientific Computing. **25**(6) (2004) 1837–1859