



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Improving the Communication Pattern in Mat-Vec Operations for Large Scale-free Graphs by Disaggregation

V. Kuhlemann, P. S. Vassilevski

July 12, 2012

SIAM Journal on Scientific Computing

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

IMPROVING THE COMMUNICATION PATTERN IN MAT-VEC OPERATIONS FOR LARGE SCALE-FREE GRAPHS BY DISAGGREGATION*

VERENA KUHLEMANN[†] AND PANAYOT S. VASSILEVSKI[‡]

Abstract. Matrix vector multiplication (mat-vec) is the key operation in any Krylov-subspace iteration method. We are interested in Krylov methods applied to problems associated with graph Laplacians arising from large scale-free graphs. Computations with graphs of this type on parallel distributed-memory computers are challenging. This is due to the fact that scale-free graphs have a degree distribution that follows a power-law and currently available graph partitioners are not efficient for such an irregular degree distribution. The lack of a good partitioning leads to excessive inter-processor communication requirements during every mat-vec. We present an approach to alleviate this problem based on embedding the original irregular graph into a more regular one by disaggregation (splitting up) vertices in the original graph. The mat-vec operations for the original graph are performed via a factored triple matrix vector product involving the embedding graph. Even though the latter graph is larger, we are able to decrease the communication requirements considerably and improve the performance of mat-vec.

1. Introduction. Complex networks appear in various disciplines including for example mathematics, computer science, social and biological sciences. The analysis of these networks plays a crucial role in its respective fields. Biologists and physicians are interested how and how fast a disease might spread in a population, social scientists are often interested in the centrality of individuals in a social network, and computer scientists investigate the robustness of computer networks. In 1999 Barabási and Albert analyzed the topology of a portion of the world wide web and found the degree distribution of this network to follow a power law[5]. That is, the number of nodes of a certain degree decreases exponentially with the degree. If $P(k)$ denotes the number of nodes with degree k , then $P(k) \sim k^{-\gamma}$. The parameter γ typically lies in the range between 1.5 and 4. Networks with this property are called *scale-free* or *power-law* networks. These networks appear in a variety of applications including social network analysis [4, 12, 14, 30, 31, 32], web mining [9, 13, 28, 34], and bioinformatics [29, 20].

To analyze a network its topology and parameters of interest are usually modeled by a graph. A *graph* is an ordered pair $\mathcal{G} = (V, E)$ where V is a set of n nodes or *vertices*, $V = \{1, 2, \dots, n\}$, and E is a set of edges. An *edge* is a pair of nodes and describes a connection between two nodes. These pairs can be ordered or unordered. If the set of edges consists of ordered pairs of nodes, the graph is called a *directed graph* or *digraph*, otherwise the graph is called *undirected*. In the following we are only interested in undirected graphs and will omit the term undirected.

One faces many challenges when analyzing scale-free networks. The networks of interest are usually huge and are constantly increasing in size. A Google search on the number "2" in October 2010 returned about 16,250,000,000 results while the same search in January 2012 returned about 25,270,000,000 results. Thus, it is required to use distributed memory systems for computations on large scale-free graphs.

In network analysis one is often interested in finding eigenvalues and eigenvector of the graph Laplacian. Given a graph \mathcal{G} the *unnormalized graph Laplacian* is given by

$$L(\mathcal{G}) = D(\mathcal{G}) - A(\mathcal{G}),$$

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

[†]Department of Mathematics and Computer Science, Emory University, Atlanta, GA

[‡]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA

where $A(\mathcal{G})$ is the *adjacency matrix* of the graph \mathcal{G} and $D(\mathcal{G})$ is a diagonal matrix with vertex degrees on the diagonal. The computation of eigenvalues and eigenvectors of scale-free graphs is very expensive. Any Krylov method based eigensolver (the method of choice in the large-scale) spends the majority of the time in the matrix-vector multiplication. Yoo et al. identified the increased communication overhead in the mat-vec as the performance bottleneck for parallel eigensolvers for scale-free graphs [38]. A common technique to improve the communication requirements is to re-partition the matrix before starting any computations. Graph partitioners attempt to partition the nodes of a graph in roughly equal sized non-overlapping parts such that the number of edges between these parts is minimized. However, state of the art parallel graph partitioners such as ParMetis [26] and Pt-scotch [10] were designed for graphs with a more regular or uniform degree distribution. They employ multilevel partitioning algorithms [11, 18, 21, 22, 23, 24, 25] which depend on coarsening the graph until it is small enough to be efficiently partitioned. In the case of scale-free graphs these partitioners produce partitions that only slightly improve the communication behavior and require a high amount of time and memory [1]. The aforementioned graph partitioners attempt to partition the nodes of a graph. Edge or 2D partitioning has been successfully used to improve the scalability of mat-vec [38]. While mat-vec based on 2D partitioning can be easily used for matrix-free eigensolvers, available multilevel methods require a row-(or edge-) wise distribution of the matrix.

In this paper we present a method that embeds the original irregular graph into a more regular one by disaggregation (splitting up) vertices in the original graph. The mat-vec operations for the original graph are performed via a factored triple matrix-vector product involving the embedding graph. Even though the latter graph is larger, we are able to decrease the communication requirements considerably and improve the performance of mat-vec.

The remainder of the present paper is structured as follows. In Section 2, we summarize our disaggregation idea, whereas in Section 3, we apply it such that to reduce inter-processor communications in the parallel implementation of the matrix-vector products. In the last section, Section 4, we illustrate the performance of our method. At the end we give some conclusions and identify areas for future work.

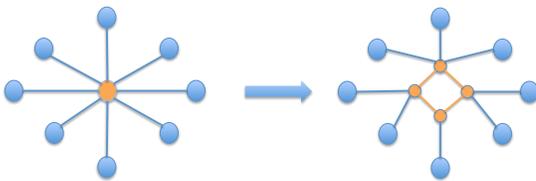


FIG. 2.1. *Disaggregating a node and using a circle as connected between the new nodes.*

2. Disaggregation. Scale-free graphs have a very irregular degree distribution. The existence of a few very high degree nodes together with many small degree nodes make computations with these types of graph particularly challenging. We attempt to tackle these challenges by disaggregation or splitting up nodes in the graph. Thus, resulting in a larger graph with a more favorable structure. Enlarging a graph of a sparse matrix with unfavorable structure to a graph with a more desired structure has been used before. In particular, in the context of matrix stretching where dense rows (columns) in a sparse matrix are split into several more sparse rows (columns) [2, 16]. For other method that use matrix enlarging refer to [3]. In the finite element literature, the popular FETI (finite element tearing and interconnecting) technique, [15], can be viewed as a specialized disaggregation method.

If a node i is disaggregated, it is split up into several new nodes i_1, \dots, i_k and every neighbor of node i in the graph is connected to exactly one of the nodes i_1, \dots, i_k . Usually, we connect the nodes i_1, \dots, i_k , called internal nodes, with a connected structure such as a cycle as seen in figure 2.1 or a complete graph as seen in figure 2.2. We can represent the disaggregated graph \mathcal{G}_f as a combination of a graph that contains the same number of edges as the original graph \mathcal{G} and a graph that contains only the new internal edges, called *internal graph*. A visualization can be seen in figure 2.3. The matrix corresponding to the disaggregated graph \mathcal{G}_f is denoted by A_f , the matrix with the same number of edges as \mathcal{G} , but nodes from \mathcal{G}_f is denoted by B , and the matrix corresponding to the internal graph is denoted by C . We can write A_f as $A_f = B + sC$, where s is the weight given to the internal edges. Under certain conditions the eigenvalues of A_f approximate the eigenvalues of A provided that the weigh on the internal edges s is chosen large enough.

Next, we construct a non-negative intergrid-transfer operator Q . Let n denote the number of nodes in the original graph \mathcal{G} and $n_{\mathcal{G}_f}$ the number of nodes in the disaggregated graph \mathcal{G}_f . The operator Q is used to prolongate a vector from \mathbb{R}^n to $\mathbb{R}^{n_{\mathcal{G}_f}}$,

$$Q : \mathbb{R}^n \rightarrow \mathbb{R}^{n_{\mathcal{G}_f}}$$

The matrix Q is constructed as follows. If $i = 1, \dots, n_{\mathcal{G}_f}$ are the nodes in \mathcal{G}_f and $j = 1, \dots, n$ are the nodes in \mathcal{G} , then $Q_{ij} = 1$ whenever node i in \mathcal{G}_f represents node j in \mathcal{G} . Thus, if node j in \mathcal{G} is not disaggregated the j th column in Q has exactly one entry. If j is disaggregated into k nodes, the j th column contains exactly k entries. Now, the original matrix A can be written in terms of the disaggregated matrix A_f by $A = Q^t A_f Q$. That is, we can replace the mat-vec Ax by the factored triple matrix vector product $Q^t(A_f(Qx))$. Note that $C \cdot Q = 0$ and the internal graph is not necessary for the factored triple matrix vector product. That is, one can use $Q^t(B(Qx))$ for the mat-vec. The transfer operator Q can be scaled such that it is orthogonal. Let $D \in \mathbb{R}^{n_{\mathcal{G}_f} \times n_{\mathcal{G}_f}}$ be a diagonal matrix such that $D_{i,i} = \frac{1}{\sqrt{k}}$ if node i is disaggregated into k nodes. Then, $\hat{Q} = Q \cdot D$ is orthogonal and has the following properties

- $\hat{Q}^t \hat{Q} = I_n$ and
- $\hat{Q} \hat{Q}^t$ is a projection onto the range of \hat{Q} .

In a linear algebra matrix form the disaggregation can be described as follows. Consider the following matrix

$$A = \begin{bmatrix} A & \begin{bmatrix} 0 \\ \mathbf{a} \\ \alpha \end{bmatrix} \\ [0, \mathbf{a}^t] & \alpha \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \mathbf{a} \\ 0 & \mathbf{a}^t & \alpha \end{bmatrix}.$$

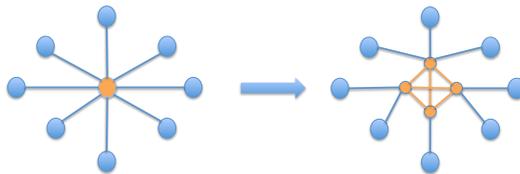


FIG. 2.2. *Disaggregating a node and using a complete graph as connected between the new nodes.*

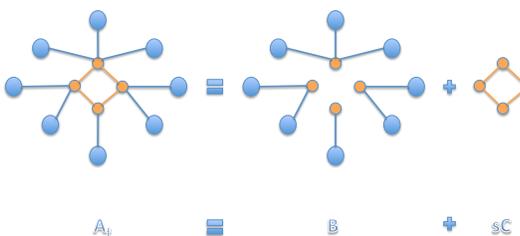


FIG. 2.3. *Presenting the disaggregated graph as a combination of the graph with internal and external edges.*

Here A is a $n \times n$ matrix, $A_{23} = \mathbf{a} \in \mathbb{R}^m$, $m \leq n$, and $\alpha \in \mathbb{R}$. In our application, the last row (and column) will correspond to a vertex of the associated sparse graph for \mathcal{A} with large degree $m \geq 1$ that we want to disaggregate.

We are interested in the following “matrix embedding”. Let $\mathbf{1} = (1) \in \mathbb{R}^m$ be the constant vector. Form the $m \times m$ diagonal matrix $D = \text{diag}(d_i)_{i=1}^m$, where $\mathbf{d} = (d_i)_{i=1}^m$ to be determined later on. Finally, let Λ be the $m \times m$ periodic tridiagonal graph Laplacian matrix

$$\Lambda = \begin{bmatrix} 2 & -1 & 0 & \dots & -1 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & -1 & 2 & -1 \\ -1 & 0 & \dots & -1 & 2 \end{bmatrix}. \quad (2.1)$$

In what follows, Λ can be any graph Laplacian matrix corresponding to a graph defined by the sparsity structure imposed on the additionally introduced nodes. This graph sometimes (in what follows) will be referred to as “*internal graph*.” For any such internal graph Laplacian, we have $\Lambda \mathbf{1} = 0$.

Given a parameter $s \geq 0$ and a given vector $\mathbf{c}_2 \in \mathbb{R}^m$, we form the $m \times m$ diagonal matrix $C_2 = \text{diag}(\mathbf{c}_2)$ (i.e., $C_2 \mathbf{1} = \mathbf{c}_2$), and consider

$$T = -DC_2 + s\Lambda.$$

We are interested in the following $(n + m) \times (n + m)$ embedding matrix

$$\mathcal{A}_f = \begin{bmatrix} A & \begin{bmatrix} 0 \\ D \end{bmatrix} \\ [0, D] & T \end{bmatrix}. \quad (2.2)$$

From now on we assume that the original matrix \mathcal{A} corresponds to the graph Laplacian. For the considerations below it is sufficient to assume that $\mathcal{A}\mathbf{c} \geq 0$ for a positive vector

$$\begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \sigma \end{bmatrix},$$

and that \mathcal{A} has non-positive off-diagonal entries. In that case, we choose the diagonal matrix $D = \text{diag}(d_i)_{i=1}^m$ as

$$d_i = \sigma a_i, \text{ where } \mathbf{a} = (a_i). \quad (2.3)$$

It is clear that $d_i < 0$ and $D\mathbf{1} = \mathbf{a}\sigma$.

We have the following result.

LEMMA 2.1. *The embedding matrix \mathcal{A}_f has non-positive off-diagonal entries and its action on the positive vector $\mathbf{c}_f = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{1} \end{bmatrix}$ is the same (non-negative) as of \mathcal{A} on \mathbf{c} for the common rows of \mathcal{A}*

and \mathcal{A}_f . For the embedding rows this action is zero. That is, if \mathbf{c} is a null-vector of \mathcal{A} , then \mathbf{c}_f is a null-vector of \mathcal{A}_f . In the latter case consider

$$Q = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & \frac{1}{\sigma} \mathbf{1} \end{bmatrix}.$$

Then, the following Galerkin relation holds,

$$\mathcal{A} = Q^t \mathcal{A}_f Q. \quad (2.4)$$

Proof. We first notice that all off-diagonal entries of \mathcal{A}_f are non-positive. What remains is to show that its action on the positive vector $\begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{1} \end{bmatrix} \in \mathbb{R}^{n+m}$ is non-negative. In fact that action for the common rows of \mathcal{A} and \mathcal{A}_f is the same as the action of the original matrix \mathcal{A} (which is non-negative by assumption). We have

$$\mathcal{A}_{21} \mathbf{c}_1 + \mathcal{A}_{22} \mathbf{c}_2 + D \mathbf{1} = \mathcal{A}_{21} \mathbf{c}_1 + \mathcal{A}_{22} \mathbf{c}_2 + \mathbf{a} \sigma = (\mathcal{A} \mathbf{c})_2.$$

For the embedding rows, the action is zero due to the choice of Λ and T . Indeed, we have

$$D \mathbf{c}_2 + T \mathbf{1} = D \mathbf{c}_2 + (-DC_2) \mathbf{1} + \Lambda \mathbf{1} = D \mathbf{c}_2 + (-D \mathbf{c}_2) = 0.$$

To prove the Galerkin relation (2.4), from the third component of $\mathcal{A} \mathbf{c} = 0$, we obtain

$$\mathbf{a}^t \mathbf{c}_2 + \alpha \sigma = 0. \quad (2.5)$$

Then, by direct computation, we have (using $C_2 \mathbf{1} = \mathbf{c}_2$, $D \mathbf{1} = \mathbf{a} \sigma$, and hence $\mathbf{1}^t D = \sigma \mathbf{a}^t$)

$$\begin{aligned} Q^t \mathcal{A}_f Q &= \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \frac{1}{\sigma} D \mathbf{1} \\ 0 & \frac{1}{\sigma} \mathbf{1}^t D & \frac{1}{\sigma} \mathbf{1}^t (-DC_2 + s\Lambda) \frac{1}{\sigma} \mathbf{1} \end{bmatrix} \\ &= \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \mathbf{a} \\ 0 & \mathbf{a}^t & \frac{1}{\sigma^2} \mathbf{1}^t (-D \mathbf{c}_2) \end{bmatrix} \\ &= \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \mathbf{a} \\ 0 & \mathbf{a}^t & -\frac{1}{\sigma} \mathbf{a}^t \mathbf{c}_2 \end{bmatrix} \\ &= \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \mathbf{a} \\ 0 & \mathbf{a}^t & \alpha \end{bmatrix} = \mathcal{A}. \end{aligned}$$

In the last equality, we used (2.5). This completes the proof. \square

In conclusion, we have the following result.

THEOREM 2.2. *The graph Laplacian matrix \mathcal{A} can be embedded into a larger in size but sparser matrix \mathcal{A}_f that is also a graph Laplacian matrix. The two graph Laplacians are related via*

a Galerkin relation, $A = Q^t \widehat{A} Q$ for a block-diagonal aggregation type matrix Q , where each column of Q contains non-zero constant entries in its rows that arise from a disaggregated vertex.

To show the first statement, we use the following well-known characteristic property of graph Laplacian (that can serve as an alternative definition).

PROPOSITION 2.3. Any symmetric matrix $M = (m_{ij})_{i,j=1}^n$ such that $M\mathbf{1} = 0$, can be characterized with the expression for any $\mathbf{u} = (u_i)$, $\mathbf{v} = (v_i) \in \mathbb{R}^n$,

$$\mathbf{v}^t M \mathbf{u} = -\frac{1}{2} \sum_{i,j} m_{ij} (v_i - v_j)(u_i - u_j). \quad (2.6)$$

The latter, for $m_{ij} \leq 0$, $i \neq j$, is the definition of (symmetric) weighted graph Laplacian matrix.

Proof. To prove this identity (which has been used before, see, e.g., formulas (2.1)-(2.2) in [36]), use the fact that $M\mathbf{1} = 0$, i.e., that for any i ,

$$m_{ii} = -\sum_{j \neq i} m_{i,j}.$$

Then

$$\begin{aligned} \mathbf{v}^t M \mathbf{u} &= \sum_{i=1}^n \sum_{j=1}^n m_{ij} u_i v_j = \sum_{i=1}^n m_{ii} u_i v_i + \sum_{i=2}^n \sum_{j=1}^{i-1} m_{ij} u_i v_j + \sum_{i=1}^{n-1} \sum_{j=i+1}^n m_{ij} u_i v_j \\ &= -\sum_{i=2}^n u_i v_i \sum_{j=1}^{i-1} m_{ij} - \sum_{i=1}^{n-1} u_i v_i \sum_{j=i+1}^n m_{ij} \\ &\quad + \sum_{i=2}^n \sum_{j=1}^{i-1} m_{ij} u_i v_j + \sum_{i=1}^{n-1} \sum_{j=i+1}^n m_{ij} u_i v_j \\ &= \sum_{i=2}^n \sum_{j=1}^{i-1} m_{ij} u_i (v_j - v_i) + \sum_{i=1}^{n-1} \sum_{j=i+1}^n m_{ij} u_i (v_j - v_i) \\ &= \sum_{i=2}^n \sum_{j=1}^{i-1} m_{ij} u_i (v_j - v_i) + \sum_{j=2}^n \sum_{i=1}^{j-1} m_{ij} u_i (v_j - v_i) \\ &= \sum_{i=2}^n \sum_{j=1}^{i-1} m_{ij} u_i (v_j - v_i) + \sum_{i=2}^n \sum_{j=1}^{i-1} m_{ji} u_j (v_i - v_j) \\ &= \sum_{i=2}^n \sum_{j=1}^{i-1} (m_{ij} u_i - m_{ji} u_j) (v_j - v_i). \end{aligned}$$

Thus for symmetric matrix M , $m_{ij} = m_{ji}$, we obtain the desired representation (2.6), which is the definition of the graph Laplacian matrix associated with the graph \mathcal{G} defined by the sparsity pattern of M when $m_{ij} \leq 0$, $i \neq j$. I.e., the quantities $-m_{ij} > 0$ play role of weights assigned to each edge (i, j) of the graph \mathcal{G} . \square

3. Parallel Disaggregation. The communication overhead during the matrix vector multiplication has been identified as the performance bottleneck for parallel eigensolvers for scale-free

graphs. In our experience large scale-free graphs require the communication between all processors during mat-vec. Even after re-partitioning with a graph partitioner such as ParMetis or Pt-scotch, communication is needed between all of the processors. Abou-Rjeili and Karypis explained the inability of these partitioners to find a suitable partitioning for scale-free graphs as follows [1]. Most of the available partitioners rely on multilevel methods where the graph is coarsen until it is small enough to employ partitioning directly. The coarsening methods use vertex matching to reduce the number of nodes, and the method depends on finding large enough matchings to coarsen the graph efficiently. Scale-free graphs have a very irregular degree distribution and a large number of low degree nodes is connected to a small number of very high degree nodes. This property leads to relatively small matchings and the partitioner needs a high number of levels to coarsen the graph enough to be able to partition it. In addition this leads to a very high memory demand, and is not suitable for large graphs. Furthermore, scale-free graphs usually also have the small-world property, that is, the graph has a small diameter. This property also makes partitioning more challenging.

A good vertex based partitioning tries to balance the computational work by assigning roughly the same number of vertices and edges to each processor while at the same time minimize the communication requirements. Diagonal banded matrices are optimal for communication pattern, as every processor only needs to communicate with a few close neighbors. We use disaggregation of nodes to embed the original graph in a larger graph whose matrix representation has a more banded structure. We thrive to achieve a given inter-processor communication structure. Any node that violates this communication pattern is disaggregated and copies of this node are distributed among the processors in such a way that the desired communication pattern is not violated. We usually restrict the communication to a percentage or fraction of the other processors.

The communication pattern that we are enforcing depends on the "distance" between two processors. We define the distance of two processors as follows.

DEFINITION 3.1. *Assume that the number of processors is np . For two processors P and Q , we define the distance between the two processors as*

$$dist(P, Q) = \min(|P - Q|, np - |P - Q|),$$

where P and Q are the indices (or ranks) of the processors. That is, $0 \leq P, Q \leq np - 1$.

In particular, the distance between processor 0 and processor $np - 1$ is one. If communication is restricted to a fraction $p \in (0, 1)$ of the number of processors np , we say that node v violates the communication pattern if for some neighbor u

$$dist(PROC(v), PROC(u)) > \lfloor \frac{p \cdot np}{2} \rfloor,$$

where $PROC(v)$ and $PROC(u)$ denote the processors that hold v and u , respectively.

The following proposition and its proof show how a node is split up and the neighbors are connected to the new nodes.

PROPOSITION 3.2. *Assume that communication should be restricted to a fraction $p \in (0, 1)$ of the number of processors np . This restriction can be fulfilled by disaggregating any node in the given graph $\mathcal{G} = (V, E)$ that violates the communication pattern into f nodes, where $f = \lceil \frac{np}{l} \rceil$, with $l = \lfloor \frac{p \cdot np}{2} \rfloor$.*

Proof. Node $i \in V$ violates the communication pattern if there exist $j \in V$ with $(i, j) \in E$ such that $dist(PROC(i), PROC(j)) > l$. We disaggregate node i into f nodes i_0, \dots, i_{f-1} . That is, in the disaggregated graph \mathcal{G}_f node i is represented by the nodes i_0, \dots, i_{f-1} , where node i_k ,

$k = 0, \dots, f - 1$ lies on processor $(PROC(i) + k \cdot l) \bmod np$. The neighbors of i in graph \mathcal{G} are connected to the nodes i_0, \dots, i_{f-1} in the disaggregated graph \mathcal{G}_f as follows. If u is a neighbor of i in \mathcal{G} , that is $(i, u) \in E$, then u is connected to i_k in \mathcal{G}_f , where

$$k = \begin{cases} \lfloor \frac{PROC(u) - PROC(i)}{l} \rfloor, & \text{if } PROC(u) \geq PROC(i) \\ \lfloor \frac{np + PROC(u) - PROC(i)}{l} \rfloor, & \text{if } PROC(u) < PROC(i). \end{cases}$$

Since i_k lies on processor $proc = (PROC(i) + k \cdot l) \bmod np$, and $dist(PROC(u), proc) \leq l$, the desired communication pattern is not violated by the edge between u and i_k . \square

The details of the parallel disaggregation method are summarized in Algorithm 1. The input matrix A can be either a (weighted) incidence matrix or a matrix corresponding to a graph Laplacian. In both cases the output matrix A_f will be a Laplacian matrix. To derive A_f we use the representation of a Laplacian matrix in terms of its *edge-vertex incidence matrix* EV and *weight matrix* W [6, 8, 17],

$$A_f = (EV)^t \cdot W \cdot (EV)$$

For a graph $\mathcal{G} = (V, E)$ the edge-vertex matrix EV is a matrix of size $|E| \times |V|$ that is set up as follows. Each edge $e \in E$ is arbitrarily directed as $e = (u, v)$. The row e in EV has two entries, $EV(e, u) = 1$ and $EV(e, v) = -1$. The weight matrix W of size $|E| \times |E|$ is a diagonal matrix with positive entries on its main diagonal equal to the edge weights from the original graph \mathcal{G} .

In our algorithm, we first count the number of nodes that violate the desired communication pattern on every processor. Those are the nodes that need to be disaggregated. Each of these nodes i is disaggregated into f nodes i_0, \dots, i_{f-1} , where i_0 replaces i on $PROC(i)$ and i_k is placed on processor $(PROC(i) + k \cdot l) \bmod np$. Thus, if n_d is the number of nodes in \mathcal{G} that need to be disaggregated, the number of nodes in the disaggregated graph \mathcal{G}_f is given by $n_{\mathcal{G}_f} = |V| + n_d \cdot (f - 1)$.

The number of edges in the disaggregated graph \mathcal{G}_f depends on the chosen internal graph that describes the connection between i_0, \dots, i_{f-1} . If the internal graph is a cycle (see matrix Λ in (2.1)), one edge is added for every i_k , that is, the number of edges in \mathcal{G}_f is given by $e_{\mathcal{G}_f} = |E| + n_d \cdot f$. After the number of nodes and edges of \mathcal{G}_f are determined, we set up the required matrices $EV \in \mathbb{R}^{e_{\mathcal{G}_f} \times n_{\mathcal{G}_f}}$, $W \in \mathbb{R}^{e_{\mathcal{G}_f} \times e_{\mathcal{G}_f}}$, and $Q \in \mathbb{R}^{n_{\mathcal{G}_f} \times n}$.

In a second sweep through the original graph \mathcal{G} the values in EV , W , and Q are being set. For every node i the process depends on if i is to be disaggregated. First, we describe the procedure if i needs to be disaggregated. For every i_k , $k = 0, \dots, f - 1$, set $Q(i_k, i) = 1$. In addition we need to connect the nodes i_0, \dots, i_{f-1} . This depends on the chosen internal graph. If a cycle is used, we connect i_k with i_{k+1} , $k = 0, \dots, f - 2$ and i_{f-1} with i_0 . That is, for every $e = (i_k, i_{k+1})$, $k = 0, \dots, f - 2$, we set $E(e, i_k) = 1$, $E(e, i_{k+1}) = -1$, and $W(e, e) = s$. We also set $E((i_{f-1}, i_0), i_{f-1}) = 1$, $E((i_{f-1}, i_0), i_0) = -1$, and $W((i_{f-1}, i_0), (i_{f-1}, i_0)) = s$. Next, the neighbors of i in \mathcal{G} have to be connected to the appropriate i_k . For $i \in V$ such that $(i, j) \in E$, k is chosen as in the proof of proposition 3.2. Let $j_{\mathcal{G}_f}$ denote the index of j in \mathcal{G}_f . For $e = (i_k, j_{\mathcal{G}_f})$ we set $E(e, i_k) = 1$, $E(e, j_{\mathcal{G}_f}) = -1$, and $W(e, e) = |A(i, j)|$. Note that if j lies on the same processor as i , then $k = 0$ and the edge $e = (i_k, j_{\mathcal{G}_f})$ lies on the same processor as the edge (i, j) in \mathcal{G} . Next, we describe the procedure if node i does not need to be disaggregated. In this case we simply set $Q(i_{\mathcal{G}_f}, i) = 1$, where $i_{\mathcal{G}_f}$ denotes the index of i in \mathcal{G}_f . For every $e = (i, j) \in E$, we determine the corresponding edge $e_{\mathcal{G}_f} = (i_{\mathcal{G}_f}, j_{\mathcal{G}_f}) \in E_{\mathcal{G}_f}$ and set $E(e_{\mathcal{G}_f}, i_{\mathcal{G}_f}) = 1$, $E(e_{\mathcal{G}_f}, j_{\mathcal{G}_f}) = -1$, and $W(e_{\mathcal{G}_f}, e_{\mathcal{G}_f}) = |A(i, j)|$. After all values in EV and W have been set, A_f can be determined by $A_f = (EV)^t \cdot W \cdot (EV)$.

Algorithm 1 Parallel Disaggregation

Input symmetric matrix $A \in \mathbb{R}^{n \times n}$, fraction $p \in (0, 1)$

Output disaggregated matrix A_f , transfer operator Q

```
1:  $np \leftarrow$  number of processors
2:  $l \leftarrow \lfloor \frac{proc \cdot np}{2} \rfloor$ ,  $f \leftarrow \lceil \frac{np}{l} \rceil$ 
3: for each processor  $proc$  do
4:    $count \leftarrow 0$ 
5:   for each row  $i$  that lies on processor  $proc$  do
6:     if  $\exists j$  with  $A(i, j) \neq 0$  AND  $dist(proc, PROC(j)) > l$  then
7:        $count \leftarrow count + 1$   $\triangleright$  disaggregate  $i$ 
8:     end if
9:   end for
10:  send count to the processors  $(proc + k \cdot l) \bmod np$ ,  $k = 1, \dots, f - 1$ 
11:     $\triangleright$  these processors receive copies of the disaggregated nodes from processor  $proc$ 
12:  receive the corresponding count values from processors  $(proc + k \cdot l) \bmod np$ ,  $k = 1, \dots, f - 1$ ,
13:     $\triangleright$  to determine the additional number of rows needed on this processor
14: end for
15:  $e_{\mathcal{G}_f} \leftarrow$  number of edges in  $\mathcal{G}_f$   $\triangleright$  this value depends on the internal graph
16:  $n_{\mathcal{G}_f} \leftarrow$  number of nodes in  $\mathcal{G}_f$ 
17: set up edge-vertex matrix  $EV$  of size  $e_{\mathcal{G}_f} \times n_{\mathcal{G}_f}$ 
18: set up matrix  $Q$  of size  $n_{\mathcal{G}_f} \times n$ 
19: set up diagonal matrix  $W$  of size  $e_{\mathcal{G}_f} \times e_{\mathcal{G}_f}$ 
20: for each processor  $proc$  do
21:   for each row  $i$  that lies on processor  $proc$  do
22:     if  $\exists j$  with  $A(i, j) \neq 0$  AND  $dist(proc, PROC(j)) > l$  then
23:       disaggregate  $i$ : represent row  $i$  by  $f$  rows  $i_0, \dots, i_{f-1}$ 
24:       for  $k = 0, \dots, f - 1$  do
25:          $Q(i_k, i) \leftarrow 1$ 
26:       end for
27:       set edges between nodes  $i_0, \dots, i_{f-1}$   $\triangleright$  depends on internal graph
28:       for all  $j$  with  $A(i, j) \neq 0$  do
29:         if  $PROC(j) \geq proc$  then
30:            $k \leftarrow \lfloor \frac{PROC(j) - proc}{l} \rfloor$ 
31:         else
32:            $k \leftarrow \lfloor \frac{np + PROC(j) - proc}{l} \rfloor$ 
33:         end if
34:          $j_{\mathcal{G}_f} =$  index of node  $j$  in  $\mathcal{G}_f$ ,  $e \leftarrow (i_k, j_{\mathcal{G}_f})$ 
35:          $EV(e, i_k) \leftarrow 1$ ,  $EV(e, j_{\mathcal{G}_f}) \leftarrow -1$ ,  $W(e, e) \leftarrow |A(i, j)|$ 
36:       end for
37:     else
38:        $i_{\mathcal{G}_f} =$  index of node  $i$  in  $\mathcal{G}_f$ ,  $Q(i_{\mathcal{G}_f}, i) \leftarrow 1$ 
39:       for all  $j$  with  $A(i, j) \neq 0$  do
40:          $j_{\mathcal{G}_f} =$  index of node  $j$  in  $\mathcal{G}_f$ ,  $e \leftarrow (i_{\mathcal{G}_f}, j_{\mathcal{G}_f})$ ,
41:          $EV(e, i_{\mathcal{G}_f}) \leftarrow 1$ ,  $EV(e, j_{\mathcal{G}_f}) \leftarrow -1$ ,  $W(e, e) \leftarrow |A(i, j)|$ 
42:       end for
43:     end if
44:   end for
45: end for
46:  $A_f \leftarrow (EV)^t \cdot W \cdot EV$ 
```

As described in the previous section, the original matrix A is given by $A = Q^t A_f Q$, where $A_f = B + s \cdot C$ is the disaggregated matrix and Q is an orthogonal intergrid-transfer operator. For the matrix vector multiplication it is not necessary to connect the disaggregated nodes, that is, the matrix C can be omitted. However, if the disaggregated matrix is to be used in a different context, for example as an auxiliary preconditioner, the connectivity of the graph might be desirable. Proposition 3.2 only shows that the graph corresponding to matrix B does not violate the communication pattern. However, if a cycle is used to connect the internal nodes i_0, \dots, i_{f-1} the graph \mathcal{G}_f corresponding to the disaggregated matrix A_f does not violate this pattern as well. This can easily be seen as subsequent nodes i_k and i_{k+1} lie on processors $proc_1 = (proc + k \cdot l) \bmod np$ and $proc_2 = (proc + (k + 1) \cdot l) \bmod np$, respectively, and $dist(proc_1, proc_2) \leq l$. In the following, we will use a cycle for the connection between internal (disaggregated) nodes and set the weight on these internal edges to one, that is $s = 1$.

In figure 3.1 we present the non-zero structure of a scale-free graph before and after disaggregation. The matrix is split up on 16 processors and has 100 nodes per processor. In the top on the left-hand side (a) the non-zero structure of the original matrix is shown. We can recognize that this matrix structure leads to communication between every single processor during mat-vec. On the top right (b) the structure of the same matrix is given after it is redistributed with

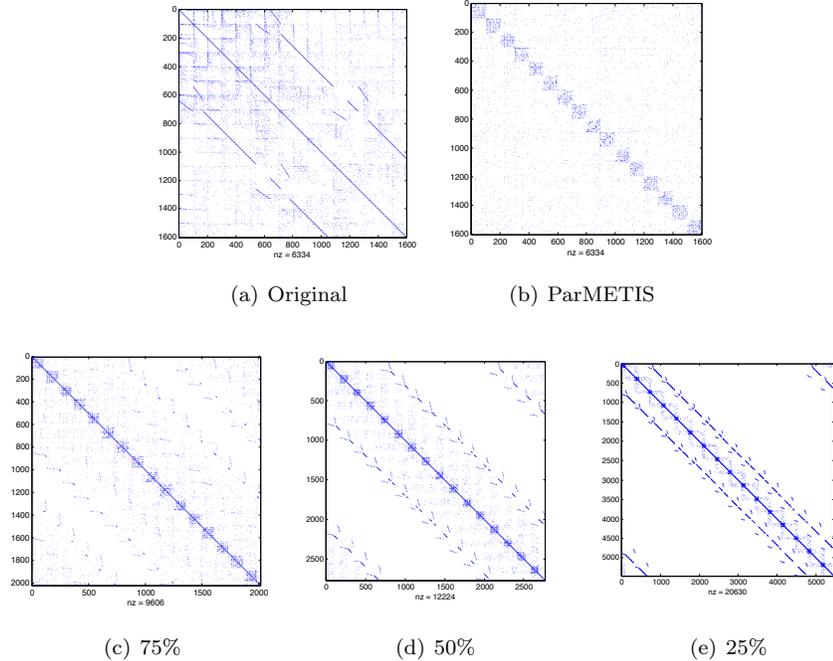


FIG. 3.1. Non-zero pattern of the original matrix, the original matrix after redistributing with ParMetis, and after using disaggregation to limit communication to 75%, 50%, or 25% of the other processors

ParMetis. The matrix is denser on the block diagonal compared to the original matrix, but communication between all processors during mat-vec is still required. ParMetis mainly reduces the size of the messages but not the number of messages that needs to be send between processors. Thus, we have a large number of small messages which is a particularly unfavorable setting for distributed systems. On the bottom row the non-zero structure of the same matrix is shown after it is disaggregated, that is the non-zero structure of A_f is shown. We restricted the communication

to 75%, 50% and 25% of the processors. Recall that every processor can communicate to processors that have a distance of at most $l = \lfloor \frac{p \cdot np}{2} \rfloor$. Thus, for the particular example with 16 processors every processor is only allowed to communicate with the closest 12, 8, or 4 processors.

While the communication volume is not or only slightly reduced with disaggregation, the number of messages that are being sent during mat-vec is significantly reduced. Thus, we have a relatively small number of large messages. In addition, each processor communicates with the same number of processors. This is a very favorable communication behavior for a distributed setting. Also, since the distribution of the disaggregated nodes is done in a very structured way, load balancing can be preserved provided that roughly the same number of nodes are disaggregated on every processor. The mat-vec will be employed by the factored triple matrix vector multiplication $Q^t(A_f(Qx))$. Note that Q is very sparse. If a node is not disaggregated Q has exactly one entry in the corresponding column. If a node is disaggregated the number of entries in the corresponding column depend on the number of nodes that this node is split up into. Next we show that this number does not increase as the number of processors increases.

PROPOSITION 3.3. *If communication is restricted to a fraction $p \in (0, 1)$ of the number of processors np with $p \cdot np > 2$, then every processor can communicate solely with its $2 \cdot l = 2 \cdot \lfloor \frac{p \cdot np}{2} \rfloor$ nearest neighbors. Every node that violates the desired communication pattern is disaggregated into $f = \lceil \frac{np}{l} \rceil$ nodes, and f is bounded by*

$$\frac{2}{p} \leq f \leq \frac{2}{p - \frac{2}{np}}.$$

Proof. With $l = \lfloor \frac{p \cdot np}{2} \rfloor$ follows

$$\frac{p \cdot np}{2} - 1 \leq l \leq \frac{p \cdot np}{2}.$$

Thus, f is bounded below by

$$f = \lceil \frac{np}{l} \rceil \geq \frac{np}{\frac{p \cdot np}{2}} = \frac{2}{p},$$

and above by

$$f \leq \frac{np}{\frac{p \cdot np}{2} - 1} = \frac{2}{p - \frac{2}{np}}.$$

□

That is, the communication requirement, meaning the number of messages, for a multiplication with Q does not increase even if the number of used processor increases.

4. Numerical Results. Our experiments were conducted on Hera, a large parallel system at Lawrence Livermore National Laboratory. Hera is a multicore Linux cluster with 864 nodes. Each node has 32 GB memory and 4 sockets with AMD Quadcore 2.3 GHz processors. The nodes are connected by Infiniband network.

In our implementation we use the PETSc library [33] for the matrix vector multiplication. We use a parallel scale-free graph generator [37] to test our method. The graph generator generates scale-free graphs using the preferential attachment method [5]. In addition, we used a real-world example from the WebGraph library [7]. This social graph, called Hollywood-2011, represents working

relationships between actors. Nodes are actors, and two actors are joined by an edge whenever they appeared in a movie together. In table 4.1 the increase in matrix size of the disaggregated matrix is given as ratio between the size of the disaggregated and original matrix. The first two matrices are generated with the scale-free graph generator and have average degree two and five. The hollywood matrix has 2,180,759 nodes, 228,985,632 edges, and average degree 105.003. As expected, the more we restrict the communication the more the disaggregated matrix increases. While the matrix sizes increase considerably we will later see that the time saved during communication is large enough to compensate for the additional computational requirement.

TABLE 4.1
Ratio of the number of nodes of the disaggregated matrix and the original matrix.

	75%	50%	25%	10%
avg=2	1.7	2.6	5.8	15.4
avg=5	2.2	3.4	7.1	18.3
hollywood	1.7	2.4	5.0	12.8

TABLE 4.2
Ratio of the number of nodes of the disaggregated matrix and the original matrix. The matrix was re-partitioned with ParMetis before applying disaggregation.

	75%	50%	25%
avg=2	1.4	1.9	3.9
avg=5	2.0	3.0	6.4
hollywood	1.66	2.4	4.77

TABLE 4.3
Ratio of the number of edges of the disaggregated matrix and the original matrix.

	75%	50%	25%	10%
avg=2	1.5	2.2	4.2	10
avg=5	1.4	1.8	2.8	5.9
hollywood	1.03	1.06	1.13	1.36

We compared the size of the disaggregated matrix if re-partitioning with ParMetis is applied before using disaggregation. The results are given in table 4.2. Note that re-partitioning the matrix leads to slightly smaller disaggregated matrices. In table 4.3 we provide the ratio of the number of non-zeros of the disaggregated matrix to the number of non-zeros in the original matrix. The increase in number of non-zeros results purely from adding internal edges.

In figure 4.1 the time needed to perform 10 matrix vector multiplications with the matrices generated by [37] are given. We consider two different matrix vector multiplications, the factored triple matrix vector product $Q^t(A_f(Qx))$ (right column) and A_fx (left column). The matrices have 10,000 nodes per processor, and have average degree 2 (top row) and 5 (bottom row). During disaggregation communication is restricted to 75%, 50%, and 25%. In both cases the matrix was re-partitioned with ParMetis before applying disaggregation. First, we can note that re-partitioning with ParMetis (solid red line) only gives a small advantage over mat-vec with the original matrix

(dotted red line) if a small number of processors are used. For a very small number of processors disaggregation does not give an advantage. The time saved during communication does not offset the increased workload generated by working with a larger matrix. However, as the number of processors increases the reduced communication becomes more prevalent. As the number of processors increases it becomes evident that restricting communication brings a large advantage even though the matrix size increases considerably. Note that we used a rather small number of processors. For a larger number of processors the memory requirement for ParMetis became too large.

In our next experiment we omitted repartitioning the matrix. The results are given in figure 4.2. The matrices used are of the same type as in the previous experiment. That is, the matrices are generated with [37] and have 10,000 nodes per processor and average degree two or five. Besides restricting the communication to 75%, 50%, and 25%, we included results where communication was restricted even further. These experiments, similar as the ones before, are weak scaling experiments. That is, ideally the time should stay constant as the number of processors increase as the matrix size per processor

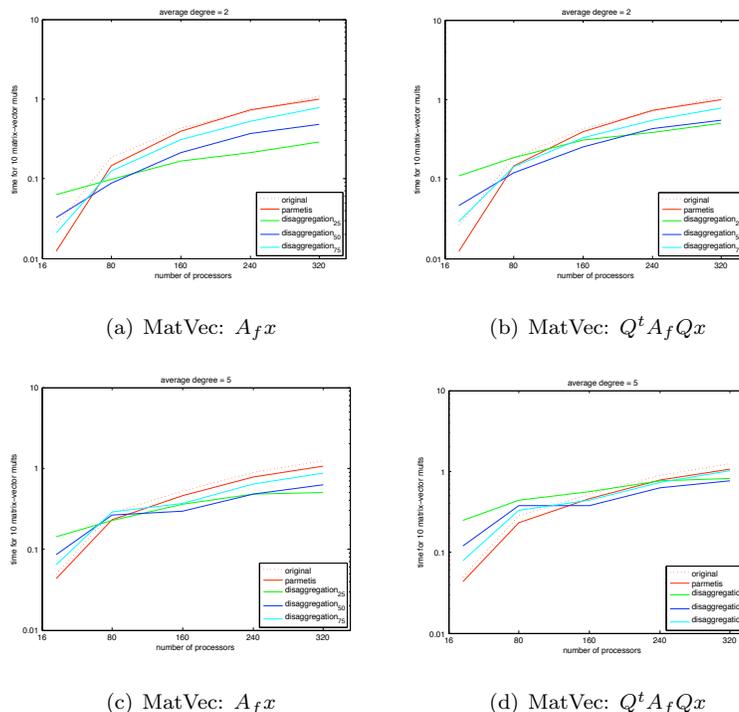


FIG. 4.1. Numerical results for matrices with average degree of two (top row) and five (bottom row). The matrices have 10,000 nodes per processor. Before disaggregating the matrices, we first repartitioned them with ParMetis. The time needed for 10 mat-vec is given in seconds.

stays the same. For the mat-vec with the original matrix (red line) this is clearly not the case. Instead the time needed for 10 mat-vecs increases very rapidly as the number of processors increases. While the timings for the 75% and 50% case also increase at a fairly large rate, the 25% and 10% case bring a clear advantage. Note that for multiplication with the disaggregated matrix A_f restricting the communication to 10% eventually outperforms disaggregation with 25% restriction. However, for the factored triple matrix vector product $Q^t(A_f(Qx))$ the increases communication and computation requirements for the multiplication with Q and Q^t offset this advantage and 25% restriction usually outperform the 10% case. This observation suggests that restricting the communication even further does not give an additional advantage.

We also give the result of an experiment with a real world graph in figure 4.3. Note that this is a strong scaling experiment and ideally the time needed should decrease as the number of processors increases. We note that for this real-world problem the behavior is similar to the synthetic problem and restricting the communication brings a huge advantage over the performance of the mat-vec with the original matrix.

Lastly, we provide a small experiment in figure 4.4 to demonstrate the effect of the improved mat-vec on the performance of an eigensolver. We used the Lanczos eigensolver from the SLEPc library [19] for our experiment. The matrix was generated with [37] and has 100 nodes per processor and average degree 2. Matrix vector multiplication was done via the factored triple matrix vector product $Q^t(A_f(Qx))$. A considerable time reduction when using the factored triple matrix vector product can be observed.

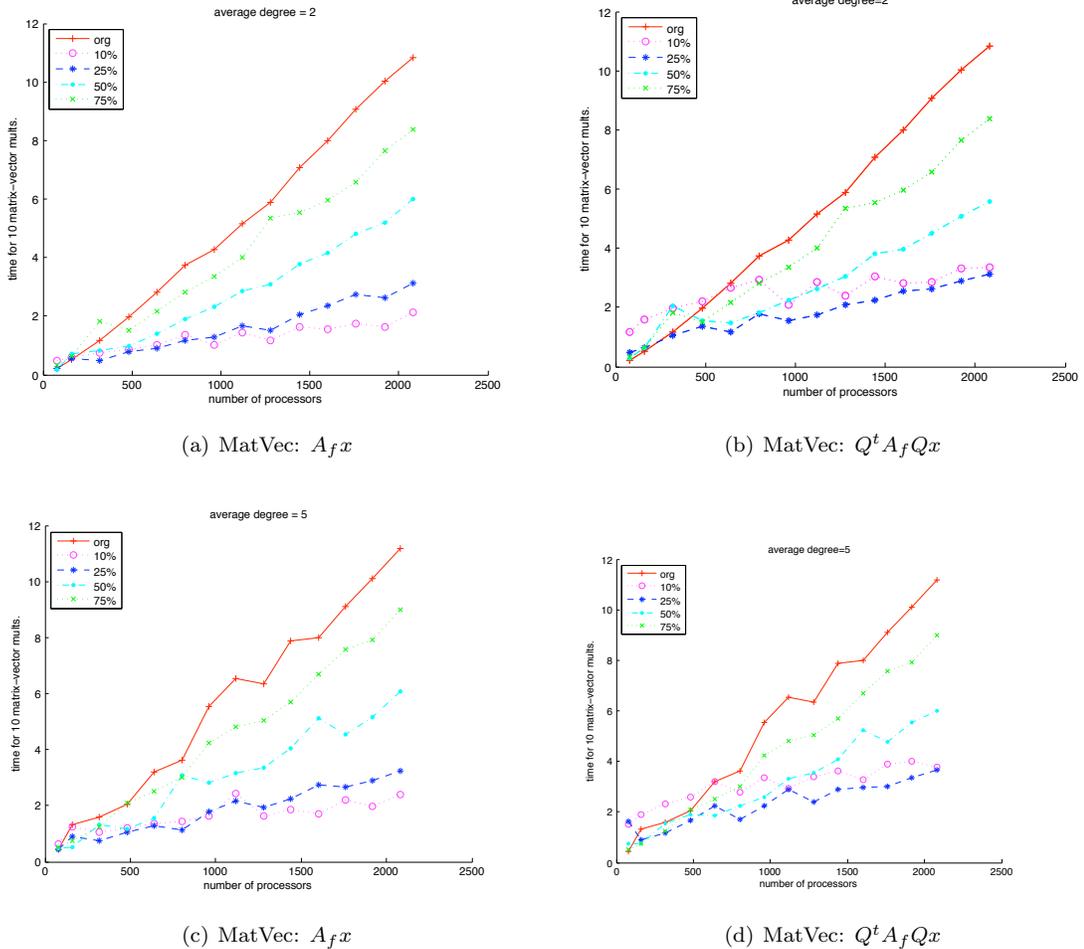
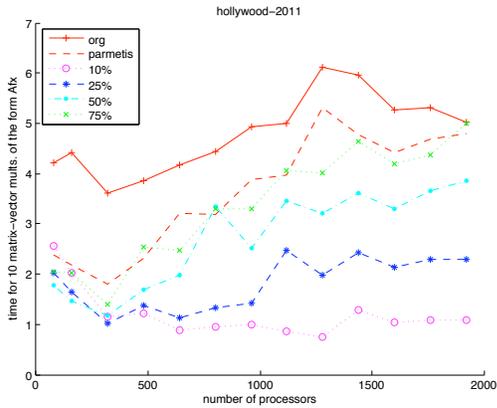
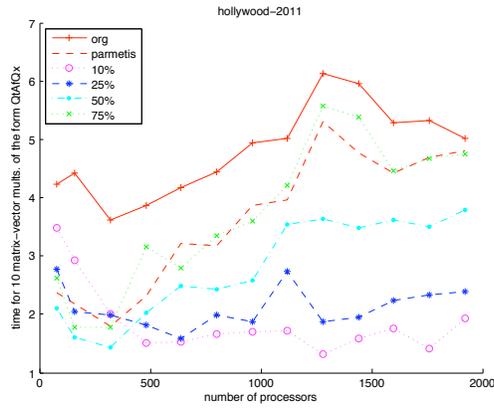


FIG. 4.2. Numerical results for matrices with average degree of two (top row) and five (bottom row). The matrices have 10,000 nodes per processor. No repartitioning is used. The time needed for 10 mat-vec is given in seconds.



(a) MatVec: $A_f x$



(b) MatVec: $Q^t A_f Q x$

FIG. 4.3. Numerical results for the hollywood-2011 matrix. Before disaggregating the matrix, we first repartitioned it with ParMetis. The time needed for 10 mat-vec is given in seconds.

Conclusions and Future Work. The matrix vector multiplication is the bottleneck for the parallel computation of eigenvalues and eigenvectors of large scale-free graphs. Currently, no parallel method is available to partition a scale-free graph in such a way that matrix vector multiplication can be completed in a sufficient way. The lack of good partitioners for scale-free graphs arises mainly from the irregular degree distribution and the existence of very large degree nodes. We provided a method to embed a scale-free graph into a more regular graph. The structure of the resulting graph is favorable for distributed environments. Even though the resulting graph is larger, we are able to improve the regular matrix vector multiplication with a factored triple matrix vector product using the disaggregated matrix and a transfer operator. While in this paper we focused on disaggregating scale-free graphs, the method described can also be used for other graphs with irregular structure that cannot be successfully partitioned.

For future work we are interested in using the disaggregated matrix with large internal weights to compute its spectrum. For large s the spectrum of the disaggregated matrix approximates the spectrum of the original matrix. For this to be feasible, a scalable (such as AMG) preconditioner is needed. In addition, we are working to use the disaggregated matrix with $s = O(1)$, to construct an “auxiliary space” (AMG) preconditioner for the original matrix. It has the form $B^{-1} = M^{-1} +$

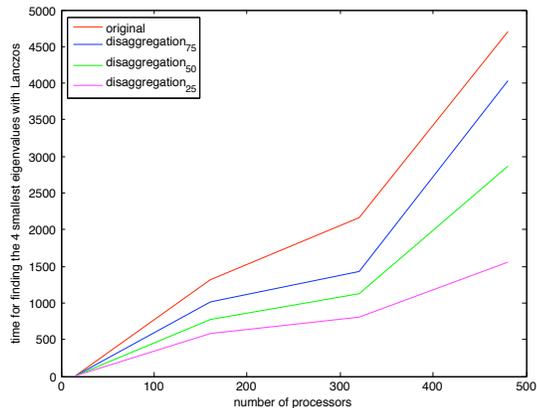


FIG. 4.4. Time in seconds needed to find the 4 smallest eigenvalues with Lanczos algorithm. The matrix has 100 nodes per processor and average degree 2.

$Q^t B_{\text{disaggr.}}^{-1} Q$ where M is a standard smoother for the original matrix and $B_{\text{disaggr.}}^{-1}$ is a preconditioner for the embedding matrix $A_f = B + sC$. This approach can be used within the effective 2D matrix storage [38] of the original matrix. In either case, these preconditioners can be used in the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) eigensolver [27]. More detailed studies on the latter topics are in progress and will be presented elsewhere.

REFERENCES

- [1] A. Abou-Rjeili, G. Karypis, *Multilevel algorithms for partitioning power-law graphs*, Parallel and Distributed Processing Symposium, 2006.
- [2] M. Adlers, and Å. Björck, *Matrix stretching for sparse least squares problems*, Num. Lin. Alg. App., 7(2):51-65, March 2000.
- [3] F. L. Alvarado, *Matrix enlarging methods and their application*, BIT, vol. 37, no. 3, 1997, pp. 473-505.
- [4] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. *Group formation in large social networks: membership, growth, and evolution*. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '06, pages 44-54, New York, NY, USA, 2006. ACM.
- [5] A.-L. Barabási, R. Albert (October 15, 1999), *Emergence of scaling in random networks*, Science 286 (5439): 509-512, Manhattan, USA, 2004, ACM Press.
- [6] N. Biggs, *Algebraic Graph Theory*, Cambridge University Press, Cambridge, 1974. Second edition 1993.
- [7] P. Boldi and S. Vigna, *The WebGraph Framework I: Compression Techniques*, Proc. of the Thirteenth International World Wide Web Conference (WWW 2004), pp 595-601.
- [8] B. Bollobás, *Graph Theory: An Introductory Course*, Springer-Verlag, New York, 1979.
- [9] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, and A. Tomkins. *Graph structure in the web: Experiments and models*. In 9th World Wide Web Conference, 2000.
- [10] C. Chevalier and F. Pellegrini, *Pt-scotch: A tool for efficient parallel graph ordering*, Parallel Computing, 34(6-8):318-331, 2008.
- [11] H. Brunst, H.-C. Hoppe, W. E. Nagel, and M. Winkler, *Performance optimization for large scale computing: the scalable VAMPIR approach*, ICCS'01: Proceedings of the International Conference on Computational Science-Part II, pages 751-760, 2001.
- [12] A. Clauset, M. E. J. Newman, and C. Moore. *Finding community structure in very large networks*. Phys. Rev. E, 70(6):066111, Dec. 2004.
- [13] R. Cooley, B. Mobasher, and J. Srivastava. *Web mining: Information and pattern discovery on the world wide web*. Tools with Artificial Intelligence, IEEE International Conference on, 0:0558, 1997.
- [14] J. Duch and A. Arenas. *Community detection in complex networks using extremal optimization*. Physical Review E, 72:027104, Jan. 2005.
- [15] C. Farhat and F. X. Roux, *A method of finite element tearing and interconnecting and its parallel solution algorithm*, Internat. J. Numer. Meths. Engrg. **32**: 1205-1227, 1991.
- [16] J. F. Grear, *Matrix stretching for linear equations*, Technical Report SAND90- 8723 Sandia National Laboratories, November 1990.
- [17] S. Guattery and G. L. Miller, *Graph Embeddings and Laplacian Eigenvalues*, SIAM Journal on Matrix Analysis and Applications, v.21 n.3, p.703-723, Feb.-March 2000
- [18] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs*, Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM), page 28, New York, NY, USA, 1995. ACM.
- [19] V. Hernandez, J. E. Roman, and V. Vidal, *Slepc: A scalable and exible toolkit for the solution of eigenvalue problems*. ACM Trans. Math. Softw., 31(3):351-362, 2005.
- [20] Y. Ji, X. Xu, and G. D. Stormo, *A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences*, Bioinformatics, 20(10):1603-1611, 2004.
- [21] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, Technical Report 95-035, University of Minnesota, Dept. of Computer Science, 1995.
- [22] G. Karypis and V. Kumar, *Multilevel k-way partitioning scheme for irregular graphs*, Technical Report 95-064, University of Minnesota, Dept. of Computer Science, 1995.
- [23] G. Karypis and V. Kumar, *Parallel multilevel k-way partitioning scheme for irregular graphs.*, Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM), page 35, Washington, DC, USA, 1996. IEEE Computer Society.
- [24] G. Karypis and V. Kumar, *A coarse-grain parallel formulation of multilevel k-way graph partitioning algorithm*,

- PPSC, 1997.
- [25] G. Karypis and V. Kumar, *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, J. Parallel Distrib. Comput., 48(1):71-95, 1998.
 - [26] G. Karypis and V. Kumar, *A coarse-grain parallel formulation of multilevel k-way graph-partitioning algorithm*, Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing, 1997.
 - [27] A. V. Knyazev, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23(2):517-541, 2001.
 - [28] R. Kosala and H. Blockeel. *Web mining research: a survey*. SIGKDD Explor. Newsl., 2:1-15, June 2000.
 - [29] E. Nabieva, K. Jim, A. Agarwal, B. Chazelle, and M. Singh, *Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps*, ISMB (Supplement of Bioinformatics), pages 302-310, 2005.
 - [30] M. E. J. Newman. *Detecting community structure in networks*. European Physical Journal B, 38:321-330, May 2004.
 - [31] M. E. J. Newman. *Fast algorithm for detecting community structure in networks*. Phys. Rev. E, 69(6):066133, June 2004.
 - [32] M. E. J. Newman and M. Girvan. *Finding and evaluating community structure in networks*. Phys. Rev. E, 69(2):026113, Feb. 2004.
 - [33] Portable, Extensible Toolkit for Scientific Computation. <http://www.mcs.anl.gov/petsc/petsc-as>.
 - [34] A. Schenker. *Graph-theoretic techniques for web content mining*. PhD thesis, Tampa, FL, USA, 2003. AAI3182715.
 - [35] J. Scott. *Social Network Analysis: A Handbook*. SAGE Publications, London, UK, 1991.
 - [36] J. Xu and L. Zikatanov, "A monotone finite element scheme for convection-diffusion equations," Mathematics of Computation **68**:1429–1446, 1999.
 - [37] A. Yoo and K. Henderson, *Parallel massive scale-free graph generators*, 2010. <http://arxiv.org/pdf/1003.3684v1>.
 - [38] A. Yoo, A. Baker, R. Pearce, and V.E. Henson *A scalable eigensolver for large scale-free graphs using 2D partitioning*, SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis.
 - [39] S. Wasserman and K. Faust. *Social network analysis: methods and applications*. Cambridge University Press, 1994.