Scalable Matrix Computations on Large Scale-Free Graphs Using 2D Graph Partitioning

Erik G. Boman, Karen D. Devine, Sivasankaran Rajamanickam
Sandia National Laboratories [*]
Scalable Algorithms Department
Albuquerque, NM 87185
{egboman,kddevin,srajama}@sandia.gov

ABSTRACT

Scalable parallel computing is essential for processing large scale-free (power-law) graphs. The distribution of data across processes becomes important on distributed-memory computers with thousands of cores. It has been shown that two-dimensional layouts (edge partitioning) can have significant advantages over traditional one-dimensional layouts. However, simple 2D block distribution does not use the structure of the graph, and more advanced 2D partitioning methods are too expensive for large graphs. We propose a new two-dimensional partitioning algorithm that combines graph partitioning with 2D block distribution. The computational cost of the algorithm is essentially the same as 1D graph partitioning. We study the performance of sparse matrix-vector multiplication (SpMV) for scale-free graphs from the web and social networks using several different partitioners and both 1D and 2D data layouts. We show that SpMV run time is reduced by exploiting the graph's structure. Contrary to popular belief, we observe that current graph and hypergraph partitioners often yield relatively good partitions on scale-free graphs. We demonstrate that our new 2D partitioning method consistently outperforms the other methods considered, for both SpMV and an eigensolver, on matrices with up to 1.6 billion nonzeros using up to 16,384 cores.

Keywords

parallel computing, graph partitioning, scale-free graphs, sparse matrix-vector multiplication, two-dimensional distribution

[*]Sandia is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

SC13 November 17-21, 2013, Denver, CO, USA
Copyright 2013 ACM 978-1-4503-2378-9/13/11 ...$15.00
http://dx.doi.org/10.1145/2503210.2503293.

1. INTRODUCTION

The need for computations on large data sets has grown rapidly. We are concerned with data that can be represented as graphs, where typically vertices represent data objects and edges represent relationships. Very large data sets are common in data mining, social network analysis, and communication networks. The graphs are highly irregular, and are often *scale-free* with a power-law degree distribution. These properties make them significantly different from graphs in scientific computing, which come mostly from meshes and discretizations such as finite elements.

Computations on scale-free graphs fall mainly into two categories: graph algorithms and linear algebra analysis. Both are important tools to analyze large graphs and networks, but we focus on the latter. A well-known algorithm for web graphs is PageRank [26], which in its simplest form is the power method applied to a matrix derived from the web-link adjacency matrix. A more computationally challenging area is spectral graph analysis. Eigenvalues and eigenvectors of various forms of the graph Laplacian are commonly used in clustering, partitioning, community detection, and anomaly detection. However, computing eigenvalues is computationally intensive. For large problems, iterative methods are needed, which rely heavily on the sparse matrix-vector product (SpMV). SpMV time can dominate solve time in an eigenvalue computation. For example, for a representative social network graph (com-orkut) with a commonly used row-wise block layout on 64 processes, SpMV took 95% of the eigensolver time. Any reduction in SpMV time, then, can significantly reduce eigensolve time. We show later that by improving the data layout for this problem, we can reduce SpMV time by 69% and overall solve time by 64%. Therefore, we focus on parallel SpMV performance. Although the eigenvalue problem is our primary target, our work applies immediately to iterative methods for linear and nonlinear systems of equations as well.

Parallel computing on scale-free graphs is very challenging. Such graphs have little locality, so partitioning them for distributed-memory computers is hard. Often the structure of the graph is ignored in the partitioning (as in [34]), as traditional graph partitioners are thought to not work well. We show that in many cases, there is enough structure that it can be exploited.

Our main contributions are:

- A new algorithm for 2D (edge) partitioning,
- An empirical comparison of several 1D and 2D distributions for scale-free graphs and the effect on SpMV,

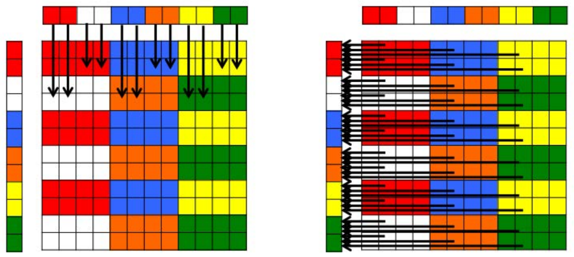

Figure 1: 2D SpMV: Expand phase (left) and fold phase (right). Arrows represent communication.

- Evidence that current graph and hypergraph partitioners are useful for scale-free graphs, and
- A demonstration of the impact of data distribution in a parallel eigensolver for scale-free graphs.

2. BACKGROUND

Partitioning (load balancing) is the problem of assigning data and computation to processes. Its goal is to balance the load (data, computation) while also reducing interprocess communication. For SpMV, both the matrix and vectors must be distributed and considered in the data layout. Since we focus on numerical algorithms using SpMV, we mainly use matrix terminology in our discussion below. However, since an undirected graph corresponds to a symmetric sparse matrix, we present both graph and matrix views.

2.1 Parallel SpMV

Suppose both the matrix A and the vectors x, y have been distributed among processes in some way. The parallel SpMV $y = Ax$ generally has four phases:

1. Expand: Send x_j to the processes that own a nonzero a_{ij} for some i.
2. Local compute: $y_i^{loc} += a_{ij}x_j$.
3. Fold: Send y_i^{loc} to the owner of y_i.
4. Sum: Add up local contributions received, $y = \sum y^{loc}$.

The *matrix partitioning* problem [5] is to decide how to distribute both the matrix and the vectors among processes. For 1D distributions, only the first two phases are necessary. For 2D distributions, all phases are required and, notably, there are two separate communication phases (expand and fold). This communication is illustrated in Figure 1. We do not discuss the details of the expand and fold phases, but note that the communication can be implemented in several ways [18].

Our focus is on iterative methods, such as eigenvalue solvers or PageRank. In this case, the vectors x and y should have the same distribution; otherwise, communication is incurred to remap one of the vectors in each iteration.

2.2 1D (Vertex) Partitioning

The most common way to partition a sparse matrix is by rows. Each process owns a set of rows and the nonzeros within those rows. This row-based approach is known as a *1D* partition for matrices, or a *vertex* partition for graphs. The vector entries are distributed in the same manner as the matrix rows.

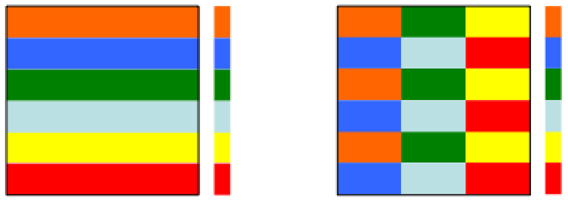

Figure 2: 1D and 2D block layouts for 6 processes. Each color represents a process.

The simplest 1D partition evenly distributes the n rows among p processes in a block fashion, such that each process owns approximately n/p consecutive rows. This distribution is the default in many matrix libraries, for example, Epetra [1] in Trilinos [19]. Although this partitioning balances the rows, it does not balance the nonzeros. For mesh-based computing, typically the variation in vertex degrees (nonzeros per row) is low, so disregarding the nonzeros is not an issue. Scale-free graphs usually have a power-law degree distribution, meaning some rows have many more nonzeros than others. This disparity can cause some processes to run out of memory, or to be much slower than other processes since the SpMV work is proportional to the number of nonzeros.

The preferred partitioning method for mesh-based simulations is graph partitioning. Here, the objective is to balance the load and reduce communication at the same time. Mathematically, we wish to minimize the edge cuts in the graph subject to a balance constraint (rows or nonzeros). Unless stated otherwise, we will always balance the nonzeros (as this is most important for SpMV). The graph partitioning problem has been well studied, and good parallel software (e.g., ParMetis [22], Scotch [27]) is available.

A drawback of graph partitioning is that it does not accurately model communication cost. Hypergraphs generalize graphs, and hypergraph partitioning can be used to accurately model communication volume. However, hypergraph partitioning is more expensive to compute than graph partitioning and yields similar results in practice for symmetric problems [28]. Hypergraph partitioning is available in Zoltan [7], PaToH [11], and hMetis [21].

2.3 2D (Edge) Partitioning

Two-dimensional matrix distributions have long been used in dense linear algebra [25]. Typically, these distributions are *Cartesian*, so each process owns the intersection of a subset of rows with a subset of the columns. Examples are the 2D block and cyclic distributions, which are used in the ScaLAPACK library [6]. These distributions limit the communication and the number of messages. Suppose the processes are arranged in a $\sqrt{p} \times \sqrt{p}$ grid. Then the maximum number of messages per process is $O(\sqrt{p})$, as compared to $O(p)$ for 1D distributions.

This approach also applies to sparse matrices [18]. In sparse 2D (edge) partitioning, the nonzeros of the matrix, or equivalently, the edges of the graph are assigned to processes. The most common 2D partitionings are Cartesian, but in the most general case, one could assign each nonzero a_{ij} independently to a process. The advantage of nonzero-based (edge-based) 2D partitions, is that one can exploit

Algorithm 1 2D Graph/Hypergraph Partitioning

```
Input: Sparse matrix $A$ and vector $x$ of dimension $n$. $p_r$, #processes in the row dimension and $p_c$, #processes in the column
    dimension, of the $p_r \times p_c$ process grid. ($p = p_r p_c$).
Output: $part$, Sparse matrix of dimension $n$, $part_{ij}$ is the part assignment for the nonzero $a_{ij}$.
1: Partition the rows and columns of $A$ into $p$ parts using graph or hypergraph partitioning.
2: Let $rpart$ denote the partition of rows and columns. {Comment: rpart also defines the vector distribution.}
3: $[procrow, proccol]$ = NONZERO-PARTITION $(rpart, p_r, p_c)$
4: for each nonzero $a_{ij}$ do
5:     {Comment: Assign $a_{ij}$ to process $(procrow(i), proccol(j))$ in the $p_r \times p_c$ process grid.}
6:     $part_{ij} = procrow(i) + proccol(j) * p_r$ {Comment: using column-major process numbering}
7: end for
```

the structure of the matrix, but the downside is that the number of messages is generally higher than for Cartesian distributions.

In the 2D block distribution both rows and columns are divided into $\sqrt{p}$ parts. However, using the block partitioning along both rows and columns creates a subtle difficulty: there is no good compatible vector distribution over p processes. The most natural choice is to align one vector with the rows and another with the columns, but different distributions of x and y would then be needed in SpMV. Instead, we desire a matrix distribution in which the diagonal entries are spread among all p processes. Such a partition can be obtained by partitioning the rows into p "stripes" as shown in Figure 2, which corresponds to a block cyclic distribution along the rows but a block partitioning along columns. For further details, see [5, Ch.4]. This method was successfully used in [34] for scale-free graphs.

Several other 2D methods have been proposed for scientific computing. In the fine-grain method [12], a hypergraph model is employed where each nonzero is a vertex. This model is optimal in terms of communication volume, but the number of messages may be high, and such partitions are expensive to compute. The coarse-grain method [13] is a Cartesian 2D method that also uses hypergraphs to reduce communication volume. As such, it is closely related to our new method, but it requires two partitioning steps, one for rows and another for columns. The second step requires multiconstraint hypergraph partitioning, which is currently available only in the serial PaToH [11] library. The Mondriaan method [33] uses hypergraph partitioning to do recursive bisection on the matrix in different directions (along rows or columns). The resulting distribution is not Cartesian, so does not have the $O(\sqrt{p})$ bound on the number of messages per process. A 2D partitioning method based on vertex separators and nested dissection was recently proposed [8, 9]. These partitioning methods have different strengths and weaknesses [14]. Although all these methods reduce communication (in some metric) compared to the block and random methods, they can be quite expensive to compute so they are often not suitable for very large scale-free graphs. There is no parallel software publicly available for these methods.

2.4 Randomization for Load Balancing

A significant issue when partitioning scale-free graphs is that it is difficult to load balance both the number of rows and number of nonzeros at the same time. The basic 1D and 2D block methods balance the number of rows, but can have significant load imbalance in the nonzeros. (We have observed up to 130x imbalance.) The nonzero imbalance causes load imbalance in the local SpMV computation, and may cause some processes to run out of memory.

Randomization is a simple but powerful technique. Each row (and corresponding vector entry) is assigned to a random process. Since the expected number of rows and nonzeros is uniform for all processes, this method generally achieves good load balance in both metrics. The drawback of this method is that communication volume increases if the given graph/matrix has some locality. Therefore, randomization is a poor load balancing method for finite elements and mesh-based computations. Nevertheless, it is a viable approach for highly irregular graphs such as scale-free graphs.

2.5 Related Work

Yoo et al. [34] considered both SpMV and eigensolvers for scale-free graphs. They compared 1D-block and 2D-block partitions, demonstrating the scalability benefits of 2D distribution for scale-free graphs up to an impressive 5B edges and 32K processors. Their partitions did not exploit the structure of the graphs to reduce communication volume. In addition, the preferential-attachment graphs generated for their experiments [35] did not require load balancing, as their graph generator produced an equal number of nonzeros on each processor. Thus, they showed, for scale-free graphs, the benefit that can be achieved solely by reducing message counts via 2D distribution. We present a comparison of the most common matrix partitioning schemes for scale-free graphs with graph and hypergraph partitioners that exploit graph structure. We also present a new 2D partitioning algorithm and show that it almost always outperforms the existing methods.

Partitioning for web graphs and PageRank was studied in [10, 15]. They compared several partitioning methods, both 1D and 2D. When additional information is available, such as the host and domain names for web pages, this information can be exploited [15]. The recent GPS graph processing system [29] has an option to use Metis graph partitioning, which was shown to reduce run-time for PageRank and some graph algorithms.

Multilevel partitioning for scale-free graphs has been considered in [3]. This work indicates that current graph partitioners need to be modified to work better on scale-free graphs. To the best of our knowledge, no such software is currently publicly available, so we use only general-purpose partitioners.

3. 2D CARTESIAN GRAPH PARTITIONING

For mesh-based applications, 1D graph partitioning reduces communication volume and cost by exploiting the structure of the graph underlying the mesh to maintain

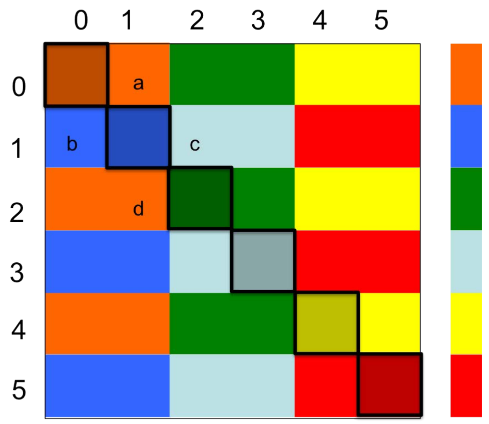

Figure 3: 2D layout of a matrix, after graph partitioning and reordering. The diagonal blocks correspond to the parts from the partitioner, and generally have more nonzeros than off-diagonal blocks. The stripes do not necessarily have the same height.

locality of dependencies within processors. The same 1D graph-partitioning methods are often assumed to be less effective for highly irregular problems such as scale-free graphs. For such graphs, the 2D-block and 2D-random distributions are effective because they limit the number of messages per process. Remarkably, we show it is possible to combine the best of these two approaches. First, we use graph (or hypergraph) partitioning to lower the communication volume. Then we impose a Cartesian 2D structure on the edges (nonzeros) to limit the number of messages. We call this new data distribution *2D Cartesian Graph Partitioning*, although other partitioning methods (such as hypergraph partitioning) may also be used. Note that we use this phrase to indicate that graph partitioning is used to exploit the structure of the graph, as opposed to the block approach used in [34].

Since we are interested in matrix computations based on a graph, we will adopt below the sparse matrix point of view. Since the sparsity structure of a matrix corresponds to a graph (or hypergraph), everything could alternatively have been phrased in terms of graphs (hypergraphs).

3.1 The 2D Partitioning Algorithm

Algorithm 1 describes our approach at a high level, while Algorithm 2 formally defines the nonzero mapping in terms of two functions ϕ and ψ.

Algorithm 1 can be explained succinctly in matrix terms: Let P be the permutation matrix associated with the graph partition of A. Partition the permuted matrix P^TAP by the block 2D method from Section 2.3, where the block sizes correspond to the part sizes from the graph partition. We do not explicitly permute the matrix in an implementation; this permutation is a purely conceptual point of view. Figure 3 shows the distribution on the permuted matrix, where the labels on the rows and columns correspond to the part numbers from the partitioning step. The (ϕ, ψ) pair in Algorithm 2 maps nonzeros to a process in a logical 2D grid. There are other possible choices for ϕ and ψ, in particular, ϕ and ψ could be interchanged.

Algorithm 1 is a two-step method where, in graph terms,

Algorithm 2 NONZERO-PARTITION($rpart, p_r, p_c$)

```
Input: rpart, 1D-part assignment for the rows and columns; $p_r$, #processes in the row dimension; and $p_c$, #processes in the column dimension, of the $p_r \times p_c$ process grid.
Output: procrow, proccol, vectors of dimension $n$ that give the mapping to process rows and columns, respectively.
1: for k= 0 to $n-1$ do
2:     procrow(k) = $\phi(k) \equiv rpart(k) \mod p_r$
3:     {Comment: procrow(k) = i ⟹ row k in the matrix is assigned to row i of the 2d process grid.}
4:     proccol(k) = $\psi(k) \equiv \lfloor rpart(k)/p_r \rfloor$
5:     {Comment: proccol(k) = j ⟹ column k in the matrix is assigned to column j of the 2d process grid.}
6: end for
```

we first partition the vertices and then the edges. For the first step, any vertex partitioning method can be used. We recommend standard graph or hypergraph partitioning, with weights to balance the number of nonzeros. Graph partitioners are generally faster than hypergraph partitioners, but hypergraph partitioners usually provide lower communication volume. After partitioning, all boundary vertices incur communication. However, which process needs to communicate with which other process is determined by the edge partitioning. Thus, in the second step, we partition the edges to minimize the number of messages. This is accomplished by a 2D Cartesian approach. Figure 4 shows a graph partitioned into six parts, represented by colored circles. Each color represents a part, consisting of both vertices and edges. The first phase in our algorithm determines the part assignment (color) for all vertices and edges within a part (circle). For simplicity, internal vertices and edges are not shown. In the second step, we assign the inter-cluster (cut) edges to parts. In Figure 4, we show exactly two edges between each pair of subgraphs, in reality there could be more or less but they will all have the same assignment. If the matrix is symmetric, it corresponds to an undirected graph; otherwise we interpret it as a directed graph. Typically, in matrix form undirected edges are stored twice ((i, j) and (j, i)), and these may be assigned differently. Figure 4 shows the part assignments (colors) for all possible cut edges. To minimize communication volume, it is beneficial to assign an edge (i, j) to the part of either vertex i or vertex j. We see that is the case for edges that go between vertices in parts that have been aligned either horizontally or vertically. See, for example, the edges labeled **a** and **b** in Figure 4 between the orange and blue parts. Some of the edges that go diagonally have counter-intuitive assignments. For example, edges **c** and **d** between the blue and green parts are orange and light blue, respectively. The edge assignment is prescribed by the 2D distribution defined in Algorithm 2. This assignment reduces the number of messages, but increases communication volume.

The most expensive step by far is the (hyper)graph partitioning; other operations are simple integer operations that take insignificant time. Therefore, 2D graph partitioning takes no more time to compute than the standard 1D graph partitioning. We also expect the data redistribution (migration) time to be similar to 1D partitioning. A significant advantage of our approach is that current software for partitioning can be re-used without modifications.

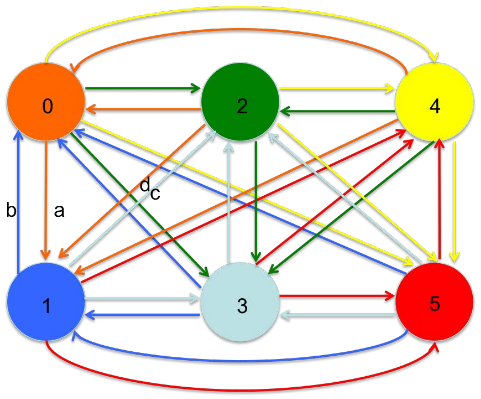

Figure 4: A graph partitioned into six parts. Colors indicate the assignment of vertices and edges. The filled circles represent internal vertices and edges in the parts defined by the partitioner. The part numbers correspond to Figure 3. Cut edges are shown twice (with different directions), as undirected edges are stored twice in the matrix.

We note that our algorithm does not guarantee balance in the number of nonzeros even if the 1D partition is balanced. However, under the (reasonable) assumption that the diagonal blocks have more nonzeros than the off-diagonal blocks, we expect that the 2D distribution also has a fairly good load balance. This idea is shown in Figure 3, where the framed diagonal blocks indicate that they are relatively dense compared to the off-diagonal blocks.

One possible improvement is to simply evaluate several 2D distributions based on the same graph partitioning and pick the "best" (for example, best load balance). As noted earlier, the functions ϕ and ψ in Algorithm 2 can be interchanged, giving two different distributions to evaluate. The additional time to do this evaluation would be small relative to the graph partitioning time.

3.2 Analysis and Comparison

Our 2D graph partitioning has several nice properties.

Number of messages: For a $p_r \times p_c$ process grid, the number of messages per process is $p_r + p_c - 2 = O(\sqrt{p})$, just as in the 2D block distribution.

Communication volume: The communication volume is similar to that of 1D graph partitioning, but may vary depending on the sparisty pattern.

Load balance: The load balance in the vector is the same as for the 1D partitioning method (graph or hypergraph). Each process swaps some off-diagonal blocks with other processes. Assuming the distribution of nonzeros is uniform across all off-diagonal blocks, the load balance in nonzeros is roughly the same as the 1D method.

Overall, we expect the 2D graph partitioning method to be very competitive versus other distributions. Compared to 1D graph partitioning, it has similar communication volume but fewer messages per process. Compared to 2D block or 2D random, it has the same number of messages but lower communication volume. Load balance may be somewhat worse than 1D graph partitioning, but for large core counts the communication time dominates SpMV, so some imbalance is tolerable.

4. IMPLEMENTATION

Our numerical tests use the Trilinos [19, 20] framework. Trilinos is a collection of inter-operable C++ packages for scientific computing. Parallel computing is supported via MPI.

We use Trilinos' Epetra package [1] for sparse matrix operations. Epetra provides classes for matrices and vectors in distributed memory parallel environments. Every data object has a *map* that describes its parallel distribution over MPI ranks. For scalability, maps themselves are distributed objects. Vectors have a single map, indicating which process owns each entry. We store the adjacency matrix of a graph as an `Epetra_CrsMatrix`. Sparse matrices in Epetra have four maps: row map, column map, range map, and domain map. These maps allow Epetra to handle both 1D and 2D data distributions in a flexible and powerful way, a significant advantage over other scientific computing software.

The row and column maps describe which nonzeros in a sparse matrix may be owned locally by a process. Specifically, row i is in the row map on a process if that process owns any nonzero entry a_{ij}. Similarly, column j is in the column map if there is a locally owned a_{ij} for some i. The domain map specifies the distribution of the input vector and the range map describes the output vector. Altogether, from these four maps Epetra can determine exactly what communication is needed in SpMV. In fact, Epetra implements the four phases as in Section 2.1. The expand communication corresponds to an *import* and the fold to an *export*. Epetra automatically constructs the Importer and Exporter based on the sparse matrix and its maps. The communication is therefore essentially point-to-point, which may not be optimal (see [18]).

In our implementation, we first read the sparse matrix from a file, and send each nonzero a_{ij} to its intended owner based on the specified distribution scheme. Once each process has its nonzero values, we construct the row map with entries indicated by the process' nonzeros. We then begin matrix construction, inserting the nonzeros into the matrix. We also construct the vector map, using it as both range and domain map for the matrix. Finally, we call `FillComplete()` for the matrix, which sets up the column map, importer and exporter transparently to the user.

To compute eigenvalues and eigenvectors, we use Trilinos' Anasazi [4] package. Anasazi is templated on the matrix and vector, so can be used with a variety of data types; we use it with Epetra. Anasazi contains a collection of different eigensolvers, including Block Krylov-Schur (BKS) and LOBPCG. Preliminary experiments indicate BKS is effective for scale-free graphs, so we use it in our experiments. BKS is a block version of the Krylov-Schur method [32] and is closely related to the implicitly restarted Arnoldi method. We use block size one, as we did not observe any advantage of larger blocks on scale-free graphs.

5. EXPERIMENTS AND RESULTS

5.1 Setup and Data

To demonstrate the effectiveness of the various partitioning strategies, we ran experiments to evaluate the performance of both SpMV and actual eigensolves on real scale free graphs from the University of Florida Matrix Collection [17] and the Stanford Network Analysis Platform (SNAP) [24]. We also used generated data, including a BTER matrix used in community detection [31] and R-MAT [16] graphs with parameter settings from the Graph500 benchmark [2]. Three R-MAT sizes were selected to provide some intuition about the scalability of the parallel distributions. Details of the matrices are in Table 1. Because we are interested on eigenanalysis of graph Laplacians, we used symmetric matrices in our experiments; for unsymmetric matrices A, we constructed the symmetric matrix as $A + A^T$.

Our 64- to 4096-process experiments were run on the cab cluster at Lawrence Livermore National Laboratory. This cluster has Intel Xeon processors with 16 cores per node; in our experiments, we assigned one MPI rank to each core. The nodes are connected by an Infiniband QDR network. Our 16K-process experiments were run on NERSC's Hopper Cray XE6. Each node has two 12-core AMD Magny-Cours processors; nodes are connected by a custom mesh network.

For all experiments, we report time only for SpMV or eigensolves, not including time to read, partition or distribute matrices. Graph/hypergraph partitioning was done as a pre-processing step on a stand-alone workstation, using only 1-64 cores. This pre-processing approach is relevant to production analysis environments, where partitions might be reused for several analyses with different goals (clustering, commute-time, bipartite detection, etc.), and where parallel-computing resources are reserved for such analyses. For use-cases requiring very few matrix operations, one must consider the partitioning cost relative to the SpMV time reduction achieved by using a more effective partition.

5.2 Impact of data layout on SpMV

We first assessed the impact of data layout on sparse matrix-vector multiplication. For each matrix, we distributed the data according to one of the following algorithms:

- 1D-Block (row-based in n/p-row blocks)
- 1D-Random (row-based with rows randomly (uniformly) distributed to processes)
- 1D-GP/HP (row-based with graph (GP) or hypergraph (HP) partitioning)
- 2D-Block (Cartesian in n/p-row blocks [34])
- 2D-Random (Cartesian with rows/columns randomly (uniformly) distributed to processes)
- 2D-GP/HP (Cartesian with nonzero distribution determined by 1D graph (GP) or hypergraph(HP) partitioning as in Algorithm 2)

We used ParMETIS 4.0.2 (through Zoltan's interface) for graph partitioning of the smaller matrices (hollywood-2009, com-orkut, cit-Patents, com-liveJournal, wb-edu, bter). Thus, for these matrices, 1D-GP is a traditional graph-partitioning approach to matrix distribution. For the larger matrices on which ParMETIS struggled to obtain a partition, we used Zoltan's parallel hypergraph partitioner in Trilinos v11.0 for the 1D-HP and 2D-HP methods. We used the same row-based graph or hypergraph partition *rpart* for 1D-GP/HP and for 2D-GP/HP as described in Algorithm 2.

We then recorded the time to do 100 SpMV per matrix per distribution, as well as statistics on the load balance, total communication volume, and maximum number of messages required for both the expand and fold phases. We examined strong scaling, varying the number of processes from 64 to 4096. To show how the methods extend to larger core counts, we ran some experiments with 16,384 cores. Complete timing results are in Table 2. Based on these results, we make several observations.

First, we note that, in all but one test case, the 2D-GP and 2D-HP methods produced faster SpMV than all other methods tested. The reduction in SpMV time varied from -5.9% (the only negative result) obtained on 64 processes with the uk-2005 matrix to 81.6% on 4096 processes with the rmat_24 matrix. Average and median reductions are 33% and 29%, respectively. The reported improvements are conservative, as the comparisons are between 2D-GP/HP and the *best of the remaining methods* for a given matrix and processor configuration. For example, had we instead chosen to compare against 2D-Random (the second best method) for all instances, we would have removed the one negative result and shown higher improvements than reported in Table 2.

The profile plot in Figure 6 graphically shows the same result. This figure shows the fraction of all problems (y-axis) for which each method requires less SpMV time than a specified multiple of the best method (x-axis), where the best method is determined separately for each problem. If a single method is the best for all the problems, it will be a line parallel to the y axis at the x value of 1. In other words, the closer the slope is to the y axis, the better the method is. Specifically in Figure 6, the (x, y) pair (2, 0.4) shows for 40% of the problems, 1D-GP/HP results in SpMV times within 2x of the time of the best method for those problems. Conversely, for 60% of the problems 1D-GP/HP results in SpMV times that are worse than 2x of the best method, which in our case is always 2D-GP/HP. The figure also shows that for 97.5% of the problems, 2D-GP/HP is the best method. Thus, we conclude that incorporating both graph or hypergraph partitioning with a 2D data distribution can yield significant reductions in SpMV time on large numbers of processes.

Second, in general, 2D methods provided lower SpMV execution times than 1D methods for large process counts. When we consider a profile plot for runs on 1000 or more processes (Figure 7), we see the 1D distributions clearly result in slower SpMV than 2D methods. Reductions in SpMV time can be attributed to the reduced number of messages needed in 2D distributions. Indeed, the number of messages appears to be more important than the total communication volume in reducing the SpMV execution time. The details for the com-liveJournal matrix in Table 3 support this claim. In this table, we show the maximum number of messages per process per SpMV and the total communication volume (number of doubles sent) per SpMV. For all 1D distributions, the maximum number of messages approaches the number of processes p, while for all 2D layouts, the number of messages approaches $2\sqrt{p}$. 2D data layouts have smaller SpMV times than their 1D counterparts due to this reduced numbers of messages, even though the total communication volume is often higher in 2D than 1D.

The strong-scaling plots in Figure 5 further demonstrate the benefit of 2D data layouts. For the test matrices displayed, both 1D and 2D methods scale to 1024 processes. However, above 1024 processes, 1D's scaling disappears even for the large rmat_26 matrix, due to the increased number of messages needed. Scalability of all 2D methods, however, is maintained to 4096 processes.

Third, while graph and hypergraph partitioning often have been thought to be ineffective for scale-free graphs, we found them almost always to be beneficial, even with 1D distributions. The results in Table 2 and the profile plot in Figure 6

Matrix	Description	# rows	# nonzeros	Max nonzeros/row
hollywood-2009	Hollywood movie actor network	1.1M	114M	12K
com-orkut	Orkut social network	3.1M	237M	33K
cit-Patents	Citation network among US patents	3.8M	37M	1K
com-liveJournal	LiveJournal social network	4.0M	73M	15K
wb-edu	Crawl of *.edu web pages	9.8M	102M	26K
uk-2005	Crawl of *.uk domain	39.5M	1.6B	1.8M
bter	Block Two-Level Erdös-Rényi (power-law deg dist $\gamma = 1.9$) [31]	3.9M	63M	790K
rmat_22	Graph 500 benchmark (a=0.57, b=c=0.19, d=0.05) [16]	4.2M	38M	60K
rmat_24		16.8M	151M	147K
rmat_26		67.1M	604M	359K

Table 1: Input matrices used in our experiments

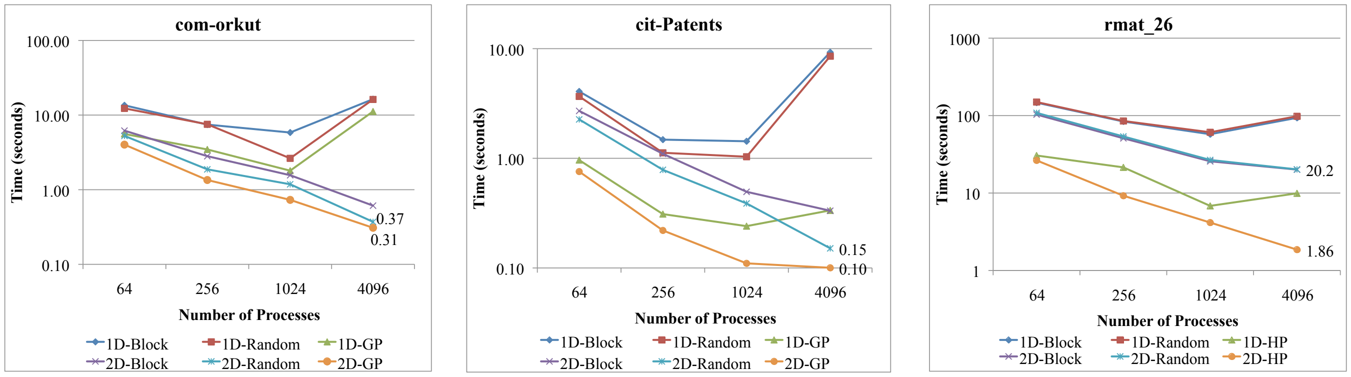

Figure 5: Strong scaling results for 100 SpMV operations. The labels compare 2D-Random vs 2D-GP/HP.

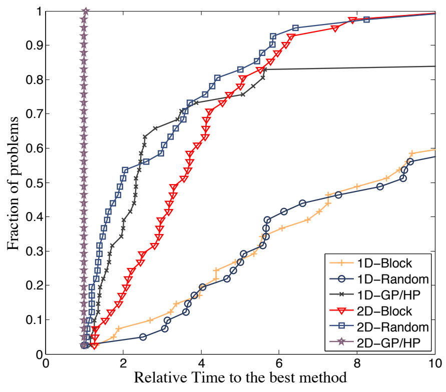

Figure 6: Performance profile comparing the time for 100 SpMV operations when using data layouts from different algorithms for all instances.

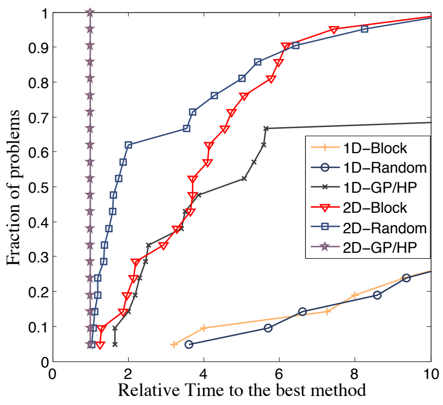

Figure 7: Performance profile comparing the time for 100 SpMV operations when using data layouts from different algorithms for instances with 1024 or more processes.

show that 1D-GP is competitive with methods 2D-Block and 2D-Random for moderate process counts. The detailed results in Table 3 provide an example. In this table, imbalance is defined to be the maximum number of nonzeros per process divided by the average number of nonzeros per process; thus, an imbalance of 1.0 is perfect balance. Applying graph partitioning in 1D-GP partitioning lowered the imbalance in the number of nonzeros per process compared to 1D-Block. It also reduced the total communication volume and, in some cases, the maximum number of messages compared to 1D-Block and 1D-Random. As a result, 1D-GP reduced the SpMV time. Graph partitioning provided similar benefits in 2D-GP. While all 2D distributions have similar message counts, 2D-GP reduced communication volume and SpMV time compared to 2D-Block and 2D-Random.

Finally, we note that simple randomization of the assignment of data to processes can often (but not always) have benefit to SpMV performance in both 1D and 2D. While randomization increases total communication volume, the improvements it yields in load balance often compensate for increased communication costs. This effect can be seen in the com-liveJournal results in Table 3. Load imbalance

Matrix	Num.	SpMV Execution time (seconds) on cab.llnl.gov cluster						Reduction in
(GP/HP)	Procs	1D-Block	1D-Random	1D-GP/HP	2D-Block	2D-Random	2D-GP/HP	SpMV time
hollywood-2009	64	4.96	4.33	2.76	2.76	**1.34**	**1.13**	15.7%
(GP)	256	2.67	2.15	1.07	1.20	**0.51**	**0.38**	25.5%
	1024	1.90	1.46	0.95	0.63	**0.23**	**0.17**	26.1%
	4096	7.14	13.04	5.81	0.41	**0.12**	**0.10**	16.7%
com-orkut	64	13.53	12.30	5.59	6.19	**5.27**	**4.02**	23.7%
(GP)	256	7.45	7.53	3.46	2.81	**1.88**	**1.35**	28.2%
	1024	5.84	2.63	1.80	1.56	**1.18**	**0.73**	38.1%
	4096	16.35	16.21	11.19	0.61	**0.37**	**0.31**	16.2%
cit-Patents	64	4.07	3.67	**0.96**	2.71	2.26	**0.76**	20.8%
(GP)	256	1.48	1.12	**0.31**	1.10	0.79	**0.22**	29.0%
	1024	1.43	1.03	**0.24**	0.50	0.39	**0.11**	54.2%
	4096	9.28	8.53	0.34	0.33	**0.15**	**0.10**	33.3%
com-liveJournal	64	6.36	4.55	**2.15**	4.75	2.77	**1.45**	32.6%
(GP)	256	3.41	2.19	**0.74**	1.94	0.96	**0.47**	36.5%
	1024	2.14	1.52	0.53	0.95	**0.43**	**0.23**	46.5%
	4096	11.13	11.58	5.01	0.41	**0.15**	**0.14**	6.7%
wb-edu	64	1.58	6.77	**1.05**	1.14	5.01	**0.90**	14.3%
(GP)	256	0.47	3.59	**0.34**	0.42	1.46	**0.25**	26.5%
	1024	0.32	1.75	0.16	**0.15**	0.66	**0.08**	46.7%
	4096	1.80	14.61	0.28	**0.10**	0.16	**0.08**	20.0%
uk-2005	64	21.43	–	**15.37**	22.12	47.68	**16.28**	-5.9%
(HP)	256	12.11	46.88	**9.31**	12.65	14.53	**4.85**	47.9%
	1024	9.95	33.96	13.02	6.86	**5.40**	**4.02**	25.6%
	4096	12.13	24.27	5.46	6.19	**2.65**	**1.71**	35.5%
bter	64	12.43	3.31	3.07	8.33	**1.94**	**1.32**	32.0%
(GP)	256	10.83	2.74	1.81	5.91	**0.90**	**0.75**	16.7%
	1024	11.34	2.68	2.65	3.50	**0.65**	**0.47**	27.7%
	4096	23.29	13.75	13.24	1.97	**0.34**	**0.33**	2.9%
rmat_22	64	5.33	5.46	**2.71**	3.36	3.49	**1.35**	50.2%
(HP)	256	3.83	3.86	**1.23**	1.87	1.92	**0.63**	48.8%
	1024	2.81	3.09	**0.66**	1.23	1.30	**0.26**	60.6%
	4096	17.91	19.76	8.71	0.71	**0.60**	**0.14**	76.7%
rmat_24	64	27.54	28.29	**7.13**	17.97	18.85	**5.64**	20.9%
(HP)	256	18.48	19.38	**4.79**	8.90	9.32	**2.11**	55.9%
	1024	25.39	24.52	**2.29**	5.08	5.16	**1.39**	39.3%
	4096	34.57	39.38	9.07	3.06	**2.88**	**0.53**	81.6%
rmat_26	64	147.06	149.73	**30.53**	103.28	108.54	**26.42**	13.5%
(HP)	256	83.94	85.24	**21.47**	51.01	53.80	**9.24**	57.0%
	1024	57.52	60.92	**6.83**	25.71	26.75	**4.16**	39.1%
	4096	93.64	98.01	**9.92**	20.09	20.20	**1.86**	81.3%
		SpMV Execution time (seconds) on NERSC Hopper Cray XE6						
com-liveJournal	16,384	87.93	47.76	19.41	0.98	**0.85**	**0.76**	10.6%
uk-2005	16,384	35.35	–	–	5.09	**2.14**	**2.05**	4.2%

Table 2: Comparison of the time for 100 SpMV operations. For the 1D-GP/HP and 2D-GP/HP methods, an indication of whether graph partitioning (GP) or hypergraph partition (HP) was used for the matrix is included with the matrix name. The reduction in SpMV time using 2D-GP/HP versus the next lowest time in the row is in the last column. Note that, because the 16K-process runs were done on a different platform from the 64- to 4096-process runs, the execution times are not directly comparable. The – indicate runs for which the time to read the matrix from file and assemble it in memory exceeded 15 minutes.

was reduced by randomization in both 1D and 2D, but randomization increased the total communication volume. For the 4096- and 16K-process experiments, randomization increased the number of messages as well. Still, SpMV time is reduced in almost all cases compared to 1D-Block and 2D-Block. For severely imbalanced problems such as this one, randomization can be a fast and effective tool for quick load balancing. However, if the original distribution is nearly balanced, the increased communication volume associated with randomization can cause SpMV execution times to increase. The wb-edu matrix, for example, exhibits a maximum load imbalance of 9.1 for 2D-Block on 4096 processes. Randomization reduces this load imbalance to 1.1, but increases communication volume from 6M doubles to 87M doubles. As a result, SpMV time increases from 1.14 seconds to 5.01 seconds. The reduction in imbalance is not sufficient to com-

Num Proc	Method	Imbal (nz)	Max Msgs	Total CV	SpMV Time
64	1D-Block	5.5	63	22.6M	6.36
	1D-Random	1.0	63	45.4M	4.55
	1D-GP	1.1	63	10.9M	2.15
	2D-Block	5.6	14	22.3M	4.75
	2D-Random	1.0	14	30.7M	2.77
	2D-GP	1.4	14	11.2M	1.45
256	1D-Block	9.1	255	29.1M	3.41
	1D-Random	1.1	255	59.8M	2.19
	1D-GP	1.1	255	14.9M	0.74
	2D-Block	9.0	30	32.6M	1.94
	2D-Random	1.0	30	47.0M	0.96
	2D-GP	1.4	30	16.4M	0.47
1024	1D-Block	12.8	1023	34.5M	2.14
	1D-Random	1.3	1023	66.3M	1.52
	1D-GP	1.2	1011	18.9M	0.53
	2D-Block	11.4	62	43.4M	0.95
	2D-Random	1.0	62	64.2M	0.43
	2D-GP	1.4	62	22.4M	0.22
4096	1D-Block	16.9	3508	39.1M	11.13
	1D-Random	1.9	4093	68.5M	11.58
	1D-GP	1.2	2927	23.6M	5.01
	2D-Block	10.1	126	53.7M	0.41
	2D-Random	1.1	126	79.6M	0.15
	2D-GP	1.9	125	30.5M	0.14
16384	1D-Block	29.5	8738	43.2M	87.93*
	1D-Random	4.2	11062	69.2M	47.76*
	1D-GP	3.3	6205	27.4M	19.41*
	2D-Block	19.8	254	63.1M	0.98*
	2D-Random	1.3	254	91.5M	0.84*
	2D-GP	1.9	250	35.6M	0.76*

Table 3: Metrics including nonzero imbalance, maximum number of messages per process per SpMV, total volume of communication (doubles) per SpMV, and time for 100 SpMV for the com-liveJournal matrix. *Note: times for 16K processes were obtained on a different platform than those for 64-4096 processes, so they are not directly comparable to lower process counts. Other metrics are not platform specific and, thus, can be compared.

pensate for the increased communication.

While most of our experiments focused on strong-scaling, we ran experiments that approximate weak scaling. The three R-MAT matrices in Table 1 were generated with 2^{22}, 2^{24}, and 2^{26} vertices and nearly constant average degree so that the number of nonzeros increased by roughly a factor of four for each larger matrix. Note, however, that weak-scaling in scale-free graphs differs from that in, say, a finite element simulation. In a finite-element simulation, the structure of the mesh and the resulting matrix does not change significantly when the problem size is increased by, say, mesh refinement. Thus, the number of neighboring processes remains nearly constant and "straight-line" weak scaling is an achievable goal. For scale-free graphs, however, the properties of the matrix change more significantly when the problem size is increased. For example, the maximum number of nonzeros/row (i.e., the maximum vertex degree) increases when the problem size is increased. For our R-MAT matri-

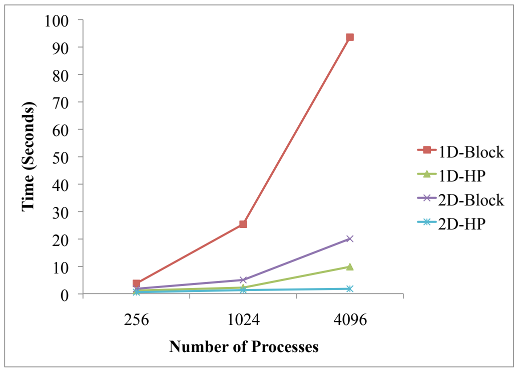

Figure 8: Weak scaling experiments for SpMV operations with the rmat_22, rmat_24 and rmat_26 matrices on 256, 1024, and 4096 processes, respectively.

ces, the maximum number of nonzeros/row increases from 60K for rmat_22 to 359K for rmat_26. This increase can affect the communication costs, making traditional straight-line scaling elusive. Still, the results in Figure 8 give a sense of how our methods perform for problems with similar structure but varying sizes. We show the times for 100 SpMV for 256 processes with rmat_22, 1024 processes with rmat_24, and 4096 processes with rmat_26, using the 1D-Block, 1D-HP, 2D-Block and 2D-HP methods. 2D-HP maintained the best weak scalability, with execution times increasing from 0.63 seconds on 256 processes to 1.86 seconds on 4096 processes. 1D-HP also maintained reasonable weak scalability to 4096 processes. The block-based methods, however, lose scalability due primarily to imbalance in the number of nonzeros per process. For example, the 2D-Block method has imbalance of 24.5 on 256 processes, 56.4 on 1024 processes, and 130.5 on 4096 processes. 2D-HP maintains load imbalance between 1.2 and 2.5 for all process configurations, while increasing communication volume relative to 2D-Block by no more than 13%.

5.3 Impact of data layout on eigensolvers

In addition to evaluating the performance of SpMV, we evaluated the performance of our data distributions in actual eigensolver computations for a subset of the matrices. For each matrix A (or, for directed graphs, $A + A^T$), we used the Block-Krylov Schur (BKS) [32] method with block size set to one to find the ten largest eigenvalues and associated eigenvectors of the normalized Laplacian matrix $\hat{L} = I - D^{-1/2}AD^{-1/2}$, where D is the diagonal matrix of degrees (i.e., d_{ii} is the degree of vertex i), and I is the identity matrix. This computation is motivated, for example, by the search for bipartite subgraphs of graphs [23]. For each experiment, we solved the eigenproblem to tolerance 10^{-3} using randomly generated initial vectors. To reduce the impact of differences in the random vector generation on the number of iterations required for convergence and, thus, on the timing comparisons, we repeated each solve ten times with different initial vectors. For each matrix and distribution, we reported the average solution time over ten solves.

While SpMV is a key kernel of BKS, other operations (e.g., orthogonalization) are also significant and have work proportional to the length of the vectors. Thus, an ideal partition would balance both the number of matrix nonze-

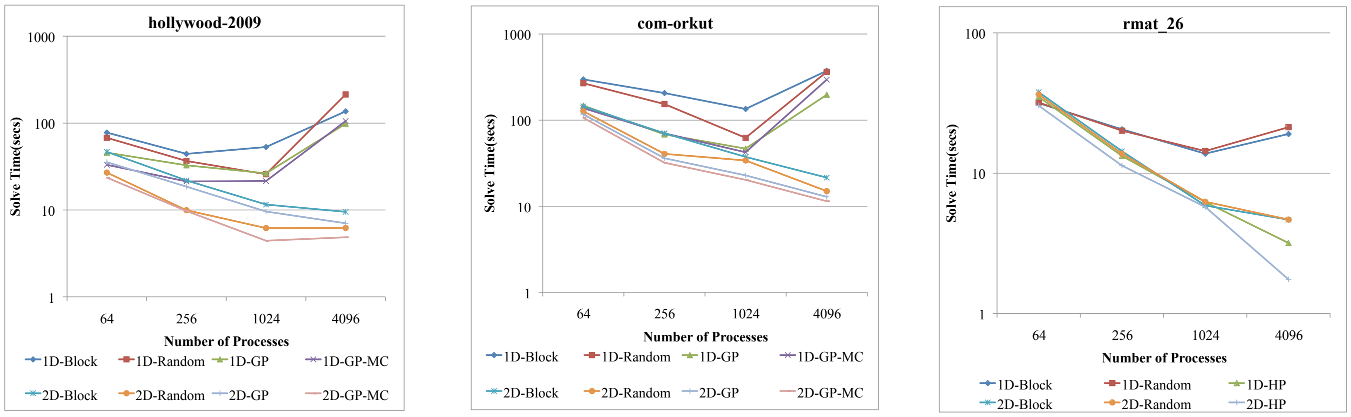

Figure 9: Strong scaling results for the eigensolver experiments with hollywood-2009, com-orkut and rmat_26 matrices.

		Eigensolve execution time (seconds)								
Matrix (GP/HP)	Num Procs	1D-Block	1D-Random	1D-GP/HP	1D-GP-MC	2D-Block	2D-Random	2D-GP/HP	2D-GP-MC	Reduction in Solve time
hollywood-2009 (GP)	64	7.78	6.78	4.57	3.31	4.67	**2.69**	3.55	**2.35**	12.6%
	256	4.42	3.67	3.27	2.13	2.19	**0.99**	1.86	**0.97**	2.0%
	1024	5.30	2.58	2.64	2.15	1.15	**0.62**	0.96	**0.44**	29.0%
	4096	13.65	21.38	9.83	10.55	0.95	**0.62**	0.70	**0.48**	22.6%
com-orkut (GP)	64	29.72	26.83	14.74	13.78	14.46	**12.66**	11.68	**10.64**	16.0%
	256	20.62	15.35	6.83	7.03	7.03	**4.06**	3.61	**3.20**	21.2%
	1024	13.48	6.23	4.66	4.23	3.74	**3.40**	2.28	**2.02**	40.6%
	4096	37.39	36.34	19.69	29.49	2.15	**1.50**	1.29	**1.14**	24.0%
rmat_26 (HP)	64	**31.55**	31.87	34.97		37.78	36.40	**30.29**		4.0%
	256	20.54	20.10	**13.28**		14.33	13.76	**11.32**		14.8%
	1024	13.77	14.37	6.23		**5.88**	6.26	**5.75**		2.2%
	4096	19.03	21.27	**3.18**		4.66	4.67	**1.75**		45.0%

Table 4: Comparison of the average eigensolve time over ten solves (in seconds) using 1D and 2D data distributions. Methods 1D-GP-MC and 2D-GP-MC use multiple partitioning constraints to balance both matrix nonzeros and matrix rows. The reduction in eigensolve time using 2D-GP-MC or 2D-HP versus the next lowest time in the row (excluding 2D-GP) is in the last column.

ros and the number of vector entries in each process. Toward that end, we used ParMETIS' multiconstraint graph partitioner [30], setting two weights per row: unit weight for the row, and the number of nonzeros in the row. The results, labeled 1D-GP-MC and 2D-GP-MC, are compared with the other 1D and 2D methods in Table 4. For the rmat_26 matrix, we used hypergraph partitioning for 1D-HP and 2D-HP due to the matrices' size; multiconstraint partitioning was not available with hypergraph partitioning.

In Table 4, the reductions in solve time shown in the last column compare 2D-GP-MC (or, for rmat_26, 2D-HP) with the next lowest execution time for the matrix and process configuration. We exclude 2D-GP from these comparisons, to more clearly compare with other methods. For all matrices, 2D-HP or 2D-GP-MC reduces overall solve time. The hollywood-2009 matrix, in particular, benefits greatly from using 2D-GP-MC to balance the vector entries as well as the matrix nonzeros. Details are in Table 5. For 2D-Block, SpMV time dominates the solve time, due to imbalance in the number of nonzeros per process. For 2D-GP, however, SpMV time is a small fraction of solve time (down to only 25% of solve time). The remaining time is dominated by vector-based calculations, which are highly imbalanced. For 2D-Random and 2D-GP-MC, both nonzeros and vector entries are balanced, resulting in lower overall solve time. The lower communication volume achieved with 2D-GP-MC relative to 2D-Random further reduces the solve time.

The com-orkut matrix also benefits from multiconstraint partitioning, but the reductions are less dramatic. This matrix experiences less severe imbalance in both nonzeros (maximum imbalance is 6.9) and vector entries (maximum imbalance is 3.9) than the hollywood-2009 matrix, so multiconstraint partitioning has less effect.

As with SpMV, overall strong scaling is improved by using 2D data layouts, as shown in Figure 9. This figure is very similar to Figure 5, in that scalability for 1D methods is lost above 1024 processes, due to the increased number of messages needed for scale-free graphs. Scalability is maintained above 1024 processes for the 2D data layouts.

6. CONCLUSIONS

We have presented a new 2D partitioning method that uses graph or hypergraph partitioning to exploit the struc-

Num Proc	Method	Nonzero Imbal	Vector Imbal	Max Msgs	Total CV	SpMV Time	Total Solve Time
64	2D-Block	5.5	1.0	14	6.4M	4.07	4.68
	2D-Random	1.0	1.0	14	12.7M	2.12	2.69
	2D-GP	1.4	4.9	14	6.1M	1.87	3.55
	2D-GP-MC	1.5	1.1	14	6.2M	1.75	2.36
256	2D-Block	16.5	1.0	30	10.3M	1.80	2.19
	2D-Random	1.0	1.0	30	22.4M	0.77	0.99
	2D-GP	1.5	14.7	30	10.9M	0.69	1.86
	2D-GP-MC	1.7	1.1	30	10.7M	0.59	0.97
1024	2D-Block	26.0	1.0	61	15.7M	0.93	1.15
	2D-Random	1.1	1.0	62	35.6M	0.44	0.62
	2D-GP	1.6	30.3	62	17.2M	0.33	0.96
	2D-GP-MC	1.6	1.1	62	17.5M	0.27	0.44
4096	2D-Block	29.0	1.0	123	23.3M	0.52	0.95
	2D-Random	1.3	1.0	126	51.2M	0.19	0.62
	2D-GP	1.9	45.6	126	25.7M	0.17	0.70
	2D-GP-MC	2.1	1.1	126	27.8M	0.16	0.48

Table 5: Metrics including imbalance in the matrix nonzeros and vector entries, the maximum number of messages per process, the total communication volume (doubles per SpMv), the amount of solve time spent in SpMV, and the total eigensolve time are shown for the hollywood-2009 matrix.

ture of scale-free graphs. We have compared our new method to other existing 1D and 2D partitioning methods on a collection of ten large graphs. Our results show that the new method reduces the run time for SpMV in almost all (41 out of 42) test cases. The run time is reduced by up to 81% compared to the best other method. We also observed that graph/hypergraph partitioning can be helpful even with the traditional 1D data distribution. We conclude that it is worth exploiting the structure of scale-free graphs, and even general-purpose partitioning tools can do so. We believe that better (1D) partitioning algorithms and software should be developed for scale-free graphs. Our 2D method can immediately use any such improved algorithm.

We studied the impact of partitioning on an eigensolver. We observed that scalability improved with 2D partitioning in general, and with our method in particular. The reduction in run time was up to 45% compared to the best other method. These results were for computing ten eigenpairs. We expect the results for fewer eigenpairs would have been even better as the SpMV is then a larger fraction of the total computation. In most of our test cases, the SpMV no longer dominates run time.

Although our test matrices were structurally symmetric, our approach extends to nonsymmetric matrices. We plan to describe and compare nonsymmetric extensions in future work. It would also be interesting to compare against the Mondriaan [33] and coarse-grain [13] methods (for problems that can be partitioned in serial).

Acknowledgements

The authors thank Rich Lehoucq, Nicole Slattengren, and Heidi Thornquist for helpful discussions on Trilinos and the Anasazi eigensolvers. We thank Van Henson, David Hysom, Geoff Sanders, and Andy Yoo for helpful discussions on 2D distributions and use of the LLNL clusters, and for providing time on the LLNL clusters for our experiments. We also thank Rob Bisseling, Umit Catalyurek, Bruce Hendrickson and Steve Plimpton for educating us over the past decade on the benefits of 2D distributions. We thank the Advanced Scientific Computing Research (ASCR) office at the U.S. Department of Energy, Office of Science, for funding support. Support for computer time at LLNL was provided by the Laboratory Directed Research and Development Graph Factorization Project (13-ERD-072) at Lawrence Livermore National Laboratory, which is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344. This research also used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

7. REFERENCES

[1] Epetra home page. `http://trilinos.sandia.gov/packages/epetra`.
[2] The Graph500 list. `http://www.graph500.org/`.
[3] A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 16–575. IEEE Press, 2006.
[4] C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Trans. Math. Softw.*, 36(3):13:1–13:23, 2009.
[5] R. H. Bisseling. *Parallel Scientific Computing: A structured approach using BSP and MPI*. Oxford University Press, 2004.
[6] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK user's guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.
[7] E. Boman, Ü. Çatalyürek, C. Chevalier, and K. Devine. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring. *Scientific Programming*, 20(2):129–150, 2012.

[8] E. G. Boman. A nested dissection approach to sparse matrix partitioning. *Proc. Applied Math. and Mechanics*, 7(1):1010803–1010804, 2007. Presented at ICIAM'07, Zürich, Switzerland, July 2007.

[9] E. G. Boman and M. M. Wolf. A nested-dissection partitioning method for parallel sparse matrix-vector multiplication. In *Proc. of the IEEE High-Performance Extreme Computing Conf. (HPEC)*, 2013. To appear.

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

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

[12] Ü. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2d decomposition of sparse matrices. In *Proc. IPDPS 8th Int'l Workshop on Solving Irregularly Structured Problems in Parallel (Irregular 2001)*, April 2001.

[13] Ü. Çatalyürek and C. Aykanat. A hypergraph-partitioning approach for coarse-grain decomposition. In *Proc. Supercomputing 2001*. ACM, 2001.

[14] U. V. Çatalyürek, C. Aykanat, and B. Uçar. On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. *SIAM J. Sci. Comput.*, 32(2):656–683, Feb. 2010.

[15] A. Cevahir, C. Aykanat, A. Turk, and B. Cambazoglu. Site-based partitioning and repartitioning techniques for parallel pagerank computation. *IEEE Trans. on Parallel and Distributed Systems*, 22(5):786–802, 2011.

[16] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *SIAM Data Mining*, 2004.

[17] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, Dec. 2011.

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

[19] M. Heroux and D. Rouson. Special issue on Trilinos. *Scientific Programming*, 20(2–3), 2012.

[20] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[21] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: application in VLSI domain. In *Proc. 34th Design Automation Conf.*, pages 526 – 529. ACM, 1997.

[22] G. Karypis and V. Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. Technical Report 97-060, Dept. Computer Science, University of Minnesota, 1997. `http://www.cs.umn.edu/~metis`.

[23] S. Kirkland and D. Paul. Bipartite subgraphs and the signless Laplacian matrix. *Appl. Anal. Discrete Math.*, 5:1–13, 2011.

[24] J. Leskovec. SNAP (Stanford Network Analysis Platform) network data sets. `http://snap.stanford.edu/data/index.html`.

[25] D. P. O'Leary and G. W. Stewart. Data-flow algorithms for parallel matrix computation. *Commun. ACM*, 28(8):840–853, Aug. 1985.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[27] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking*, volume 1067 of *Lecture Notes in Computer Science*, pages 493–498. Springer Berlin Heidelberg, 1996.

[28] S. Rajamanickam and E. G. Boman. Parallel partitioning with Zoltan: Is hypergraph partitioning worth it? In D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors, *Graph Partitioning and Graph Clustering*, volume 588 of *AMS Contemporary Mathematics*, pages 37–52. AMS, 2013.

[29] S. Salihoglu and J. Widom. GPS: A graph processing system. Technical report, Stanford University, 2012.

[30] K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multi-constraint partitioning. *Concurrency and Computation: Practice and Experience*, 14:219–240, 2002.

[31] C. Seshadhri, T. G. Kolda, and A. Pinar. Community structure and scale-free collections of Erdös-Rényi graphs. *Phys. Rev. E*, 85:056109, May 2012.

[32] G. W. Stewart. A Krylov-Schur algorithm for large eigenproblems. *SIAM J. Matrix Anal. Appl.*, 23:601–614, 2000.

[33] B. Vastenhouw and R. H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review*, 47(1):67–95, 2005.

[34] A. Yoo, A. H. Baker, R. Pearce, and V. E. Henson. A scalable eigensolver for large scale-free graphs using 2d graph partitioning. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 63:1–63:11, New York, NY, USA, 2011. ACM.

[35] A. Yoo and K. Henderson. Parallel massive scale-free graph generators, 2010.