Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 180214, 12 pages
http://dx.doi.org/10.1155/2015/180214

Research Article

Parallel Framework for Dimensionality Reduction of Large-Scale Datasets

Sai Kiranmayee Samudrala,[1] Jaroslaw Zola,[2,3] Srinivas Aluru,[4] and Baskar Ganapathysubramanian[5]

[1]*Department of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA 30080, USA*
[2]*Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY 14620, USA*
[3]*Department of Biomedical Informatics, University at Buffalo, Buffalo, NY 14620, USA*
[4]*School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA*
[5]*Department of Mechanical Engineering, Iowa State University, Ames, IA 50011, USA*

Correspondence should be addressed to Baskar Ganapathysubramanian; baskarg@iastate.edu

Received 9 March 2013; Accepted 1 August 2013

Academic Editor: Boleslaw Szymanski

Dimensionality reduction refers to a set of mathematical techniques used to reduce complexity of the original high-dimensional data, while preserving its selected properties. Improvements in simulation strategies and experimental data collection methods are resulting in a deluge of heterogeneous and high-dimensional data, which often makes dimensionality reduction the only viable way to gain qualitative and quantitative understanding of the data. However, existing dimensionality reduction software often does not scale to datasets arising in real-life applications, which may consist of thousands of points with millions of dimensions. In this paper, we propose a parallel framework for dimensionality reduction of large-scale data. We identify key components underlying the spectral dimensionality reduction techniques, and propose their efficient parallel implementation. We show that the resulting framework can be used to process datasets consisting of millions of points when executed on a 16,000-core cluster, which is beyond the reach of currently available methods. To further demonstrate applicability of our framework we perform dimensionality reduction of 75,000 images representing morphology evolution during manufacturing of organic solar cells in order to identify how processing parameters affect morphology evolution.

1. Introduction

Computational analysis of high-dimensional data continues to be a challenging problem, spurring the development of numerous computational techniques. An important and emerging class of methods for dealing with such data is dimensionality reduction. In many applications, features of interest can be preserved while mapping the high dimensionality data to a small number of dimensions. These mappings include popular techniques such as principle component analysis (PCA) [1] and complex nonlinear maps such as Isomap [2] and kernel PCA [3].

Linear manifold learning techniques, for example, PCA or multidimensional scaling [4–7], existed as orthogonalization methods for several decades. Nonlinear methods like Isomap, LLE (locally linear embedding) [8], and Hessian LLE [9] were discovered recently. Another class of methods that emerged in the past few years are the unsupervised learning techniques, including artificial neural networks for Sammon's nonlinear mapping [10], Kohenen's or self organizing maps (SOM) [11], and curvilinear component analysis [12]. Modifications to the existing algorithms of manifold learning, to improve either their efficiency or performance, were another area where efforts were focused [13–16]. For example, Landmark Isomap [17] is a modification to the original Isomap method to extend its usage to larger datasets by picking a few representative points and applying Isomap technique to them. Along with the emergence of new manifold learning techniques, different sequential implementations of these

techniques, targeting various hardware platforms and various programming languages, have been developed [18, 19].

Dimensionality reduction techniques are often compute intensive and do not easily scale to large datasets. Recent advances in high-throughput measurements using physical entities, such as sensors, or results of complex numerical simulations are generating data of extremely high dimensionality. It is becoming increasingly difficult to process such data sequentially.

In this paper, we propose a parallel framework for dimensionality reduction. Rather than focusing on a particular method, we consider the class of spectral dimensionality reduction methods. Till date, few efforts have been made in developing parallel implementations of these methods, other than development of a parallel version of PCA [20, 21], parallelization of multidimensional scaling (MDS) for genomic data [22], and algorithm development for GPU platforms [23, 24].

We perform a systematic analysis of spectral dimensionality reduction techniques and provide their unified view that can be exploited by dimensionality reduction algorithm designers. We identify common computational building blocks required for implementing spectral dimensionality reduction methods and use these abstractions to derive a common parallel framework. We implement such a framework and show that it can handle large datasets and it scales to thousands of processors. We demonstrate advantages of our software by analyzing 75,000 images of morphology evolution during manufacturing of organic solar cells, which enables us to visually inspect and correlate fabrication parameters with morphology.

The remainder of this paper is organized as follows. In Section 2 we introduce the dimensionality reduction problem and describe basic spectral dimensionality reduction techniques, highlighting their computational kernels. In Section 3 we provide a detailed description of our parallel framework including algorithmic solutions. Finally, in Section 4 we present experimental results, and we conclude the paper in Section 5.

2. Definitions and Methods Overview

The problem of dimensionality reduction can be formulated as follows: Consider a set $X = \{x_0, x_1, \ldots, x_{n-1}\}$ of n points, where $x_i \in \mathbb{R}^D$, and $D \gg 1$. We are interested in finding a set $Y = \{y_0, y_1, \ldots, y_{n-1}\}$, such that $y_i \in \mathbb{R}^d$, $d \ll D$, and $\forall_{i,j}\ |x_i - x_j|_h = |y_i - y_j|_h$. Here, $|a-b|_h$ denotes a specific norm that captures properties we want to preserve during dimensionality reduction [25]. For instance, by defining h as Euclidean norm we preserve Euclidean distance, thus obtaining a reduction equivalent to the standard technique of principal component analysis (PCA) [1]. Similarly, defining h to be the angular distance (or conformal distance [26]) results in locally linear embedding (LLE) [8] that preserves local angles between points. In a typical application [27, 28], x_i represents a state of the analyzed system, for example, temperature field and concentration distribution. Such state description can be derived from experimental sensor data or can be the result of a numerical simulation. However, irrespective of the source, it is characterized by high dimensionality, that is, D is typically of the order of 10^6 [29]. While x_i represents just a single state of the system, common data acquisition setups deliver large collections of such observations, which correspond to the temporal or parametric evolution of the system [27]. Thus, the cardinality n of the resulting set X is usually large ($n \sim 10^4$-10^5). Intuitively, information obfuscation increases with data dimensionality. Therefore, in the process of dimensionality reduction (DR) we seek as small a dimension d as possible, given constraints induced by the norm $|a-b|_h$ [25]. Routinely, $d < 4$ as it permits, for instance, visualization of the set Y.

DR techniques have been extensively researched over the last decade [25]. In particular, methods based on spectral data decomposition have been very successful [1, 2, 9] and have been widely adopted. Early approaches in this category exploited simple linear structure of the data, for example, PCA or multidimensional scaling (MDS) [30]. More recently, techniques that can unravel complex nonlinear structures in the data, for example, Isomap [2], LLE, and kernel PCA [3], have been developed. While all these methods have been proposed taking into account specific applications [19, 25], their underlying formulations share similar algorithmic mechanisms. In what follows we provide a more detailed overview of spectral DR techniques and we identify their common computational kernels that form the basis for our parallel framework.

2.1. Spectral Dimensionality Reduction. The goal of DR is to identify a low-dimensional representation Y of the original dataset X, that preserves certain predefined properties. The key idea underpinning spectral DR can be explained as follows. We encode desired information about X, that is, topology or distance, in its entirety by considering all pairs of points in X. This encoding is represented as a matrix $A_{n\times n}$. Next, we subject matrix A to unitary transformation V, that is, transformation that preserves norm of A, to obtain its sparsest form Λ, where $A = V\Lambda V^T$. Here, $\Lambda_{n\times n}$ is a diagonal matrix with rapidly diminishing entries. As a result, it is sufficient to consider only d entries of Λ to capture all the information encoded in A. These d entries constitute the set Y. The above procedure hinges on the fact that unitary transformations preserve original properties of A [31]. Note also that it requires a method to construct matrix A in the first place. Indeed, what differentiates various spectral methods is the way information is encoded in A.

We summarize the general idea of spectral DR in Algorithm 1. In the first four steps we construct the matrix A. As indicated, this matrix encodes information about the property that we wish to preserve in the process of DR. To obtain A we first identify the k nearest neighbors (KNN) of each point $x_i \in X$. Note that currently all studied methods use KNN defined in Euclidean space. This enables us to define a weighted graph G that encapsulates, both distance and topological, properties of the set X. Given graph G, we can construct a function $F_G : X \times X \to \mathbb{R}$ to isolate the desired property. For instance, consider the Isomap

Input: Set $X = \{x_0, x_1, \ldots, x_{n-1}\}$, $x_i \in \mathbb{R}^D$, and the target dimension d.
Output: Set $Y = \{y_0, y_1, \ldots, y_{n-1}\}$, $y_i \in \mathbb{R}^d$.
(1) For each $x_i \in X$ find its k nearest neighbors.
(2) Define directed weighted graph $G = (X, E, \omega)$,
where $(x_i, x_j) \in E$ iff x_j is a neighbor of x_i,
and $\omega(x_i, x_j)$ is a distance measure,
usually $\omega(x_i, x_j) = |x_i - x_j|_2$.
(3) Let $W_{ij} = F_G(x_i, x_j)$, where F_G extracts specific property
from graph G.
(4) Normalize W to obtain matrix A.
(5) Find eigenvectors of A, $A = V \Lambda V^T$.
(6) Identify latent dimensionality d.
(7) Y is represented by the first d rows of V.

Algorithm 1: Spectral dimensionality reduction.

algorithm in which the geodesic distance is maintained. In this case, F_G returns the length of the shortest path between x_i and x_j in G. Note that for some methods F_G is very simple; for example, for PCA it is equivalent to a distance measure ω, $F_G(x_i, x_j) = \omega(x_i, x_j)$, while for other methods F_G can be more involved. Differences between various DR methods and their corresponding function F_G are outlined in Table 1. The property extracted by function F_G is stored in an auxiliary matrix W, which is next normalized to obtain matrix A. This process of normalization is a simple algebraic transformation, which ensures that A is centered and hence that the final low-dimensional set of points Y contains the origin and is not an affine translation [31]. Subsequently, A is spectrally decomposed into its eigenvalues that constitute the sparsest representation of A. Resulting eigenvectors and eigenvalues are then postprocessed to extract the set Y of low-dimensional points.

The abstract representation of spectral DR methods in Algorithm 1 is based on a thorough analysis of existing techniques [1, 2, 8]. While this representation is compact, it offers sufficient flexibility to, for instance, design new dimensionality reduction procedures. At the same time it provides clear separation of individual computational steps and explicates data flow in any DR process. We exploit both these facts when designing our parallel framework.

2.2. Performance Analysis of Dimensionality Reduction Methods. We used the above presentation of DR methods to identify their basic computational kernels. To better understand how these kernels contribute to the overall performance of different DR methods, we performed a set of experiments using domain specific implementation in Matlab. Experiments were carried out for varying n and a fixed $D = 1000$ on a workstation with 8 GB of RAM and an Intel 3.2 GHz processor. Obtained results are presented in Table 2.

As can be seen, the run time of analyzed methods is dominated by two steps, namely, KNN and construction of the auxiliary matrix W. Together they account for 99.8% of the total execution time for $n = 4000$. In our implementation the KNN procedure depends on all-versus-all distance calculations. This is justified taking into account that D is very large and thus efficient algorithmic strategies for KNN, for example, based on hierarchical space decomposition [32], are infeasible. Consequently, complexity of this step is $O(Dn^2)$. The cost of computing matrix W depends explicitly on the definition of function F_G. Among existing DR techniques this function is the most complex for the Isomap method. Recall that in the process of DR we are interested in preserving either distance or local topology characteristics. Local topology properties can be directly obtained from KNN [8, 9, 33], inducing computationally efficient definition of F_G. Conversely, distance characteristics must conform to global constraints and therefore have higher computational complexity [34]. In case of Isomap, pairwise geodesic distances can be efficiently derived from all-pairs shortest path distances using, for example, Floyd-Warshall algorithm, with $O(n^3)$ worst-case complexity.

Another significant DR component is normalization. Although implementation of this step varies between different methods it is invariably dominated by matrix-matrix multiplication. Therefore, we assume overall normalization complexity to be $O(n^3)$. The last important component is the eigenvalue solver. In general, complexity of this kernel varies depending on the particular solver used. Commonly employed algorithms include Lanczos method [35], Krylov subspace methods [36], or deflation-based power methods [37, 38]. The choice of method is driven by the structure of the matrix and the number of required eigenvalues. Standard distance preserving DR methods operate on dense symmetric matrices, while topology preserving methods involve sparse symmetric matrices. Accordingly, complexity of these techniques is usually $O(dn^2)$, where d is the number of desired eigenvalues.

A final key factor we have to consider is memory complexity of the described kernels. Here, the main contributing structures are matrices W and A. These matrices are most often dense and in the majority of cases require $O(n^2)$ storage. Because KNN directly depends on distances between all pairs, it utilizes an $n \times n$ matrix as well. Finally, input dataset X requires $O(Dn)$ memory.

Table 1: Comparison of selected spectral dimensionality reduction methods.

	PCA	Isomap	LLE
Parameter k in KNN	n	$\sim d$	$\sim d$
Function F_G	$\omega(x_i, x_j)$	Length of the shortest path between x_i and x_j in G	α_{ij} if $(x_i, x_j) \in E$, 0 otherwise, where $x_i = \sum_{x_l:(x_i,x_l)\in E} \alpha_{il} x_l$
Normalization	$W^*_{ij} = W^2_{ij}$, $A = H^T W^* H$	$W^*_{ij} = W^2_{ij}$, $A = H^T W^* H$	$A = (I - W)^{-1}(I - W)^{-T}$

Note: I is the identity matrix, and $H = I - (1/n)\mathbf{1}_{n\times n}$.

Table 2: Run time (in seconds) of different DR components ($d = 3$).

n	100	1000	2000	4000
KNN	0.08640	1.34998	5.66768	27.91930
W in PCA	—	—	—	—
W in Isomap	0.06470	14.9030	130.130	1153.30
W in LLE	0.08960	0.12601	0.24609	0.49253
Normalize	0.00195	0.11875	0.74934	5.56630
Eigensolve	0.02916	0.05536	0.23267	0.85211
Extract Y	0.00020	0.00014	0.00016	0.00022

One important caveat that affects the above analysis is the relationship between D and n. In many applications D is significantly greater than n. This is not surprising taking into account that acquiring high resolution data (hence high-dimensional) is resource intensive. Therefore one may expect that with increasing D there is usually decrease of n. In our applications [29, 39] it is not uncommon that $D = O(n^2)$ or even $D = O(n^4)$. Consequently, the KNN step in Algorithm 1 becomes the most compute intensive while memory requirement is dominated by the input data. Observe that this trend is reflected in our experimental data.

3. Parallel Framework for Dimensionality Reduction

Dimensionality reduction very quickly becomes both memory and compute intensive, irrespective of the particular method. Memory consumption arises from the size of input data and the auxiliary matrices created in the process. The computational cost is dominated by pairwise computations and weight matrix construction. The goal of our framework is to scale DR methods to very large datasets that could be analyzed on large parallel machines.

We designed our parallel DR package following the general outline presented in Algorithm 1. Taking into account significant memory and computational complexity, we focused on distributed memory machines with MPI. To ensure modularity of the framework without sacrificing performance and scalability, we decided to employ a scheme in which processors are organized into a logical 2D mesh. In what follows, we assume a simple point-to-point communication model with latency t_s and bandwidth $1/t_m$.

3.1. Constructing Graph G. The graph construction procedure is based on identifying k nearest neighbors of each input point. Because of the high dimensionality of the input data it is advantageous to implement KNN in two steps, where we first compute all pairwise distances and then we identify neighbors in a simple scan. Note that these pairwise distances actually represent the distance measure ω (see Algorithm 1). Therefore we will consider ω to be an $n \times n$ distance matrix. Parallel pairwise computation is a well studied problem [40]. Here, we benefit from our earlier experience with accelerating pairwise computations on heterogeneous parallel processors [41].

Let $p = q^2$ denote the number of processors conceptualized as organized into a $q \times q$ virtual mesh. We decompose ω into blocks of $(n/q) \times (n/q)$ elements. Processor with coordinates (i, j) is responsible for computing elements of ω within block (i, j). This scheme requires that each processor stores two blocks of n/q points of the input dataset X that correspond to row-vectors and column-vectors used to compute respective part of the matrix ω. In our implementation, the distribution of the input dataset is performed by parallel I/O with initially preprocessed X. Note that to obtain a single element of ω we perform $|a - b|_2$ norm computations, which are particularly well suited for vectorization. Therefore, we hand-tuned our code to benefit from SIMD extensions of modern processors.

Given pairwise distances, the second step is to identify neighbors of individual points (i.e., vertices of G). This step is executed only for methods where $k < n$, which virtually involves all methods other than PCA (see Table 1). As in the case of pairwise computations it can be efficiently parallelized using the following scheme. Initially, each processor creates a set of k candidate neighbors with respect to the block of matrix ω it stores. Specifically, processor with coordinates (i, j) searches for neighbors of the set of points $[x_{in/q}, \ldots, x_{(i+1)n/q})$ by analyzing rows of its local block of ω. Because k is very small this operation can be performed using a simple linear scan. Next, all processors within the same row perform all-to-all communication to aggregate candidate neighbors. Here, the candidate neighborhood of point x_l is assembled on the processor with coordinates $(\lfloor lq/n \rfloor, \lfloor lp/n \rfloor \bmod q)$. This processor merges candidate neighborhood lists into the final set of k nearest neighbors. Observe that this completes the graph construction phase—graph G is now stored in the form of adjacency list distributed over p processors.

The computational complexity of the entire procedure can be decomposed into $O(Dn^2/p)$ cost of computing distance matrix, $O((n/\sqrt{p}) \log k)$ cost of performing local KNN search, and $O((t_s + t_m(nk/p))\sqrt{p})$ cost of distributing local

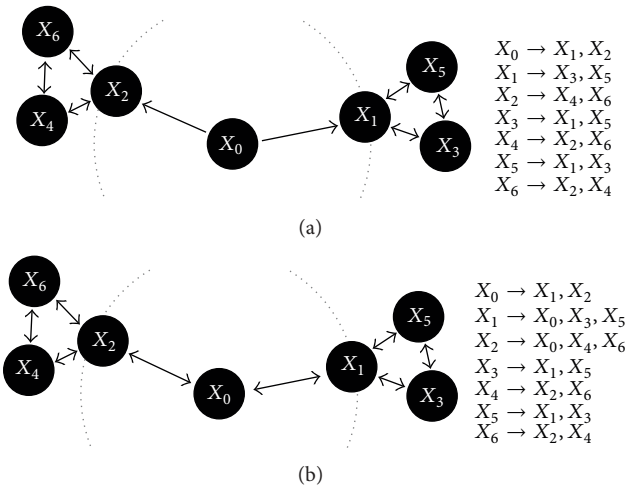

FIGURE 1: Graph G before (a) and after (b) symmetrization for an example set of seven points and $k = 2$.

KNN information to all processors sharing a row. Hence, the total complexity is bounded by $O(Dn^2/p+(t_s+t_m(nk/p))\sqrt{p})$, which is optimal for $p < n^2$.

3.2. Building Auxiliary Matrix W. Given graph G we proceed to the next step, which involves constructing the auxiliary matrix W from the information encapsulated in G. As is the case of ω we distribute W over $q \times q$ mesh of processors.

Recall that the formulation of DR methods proposed in Algorithm 1 ensures that the only step that is method dependent is the construction of matrix W. Consequently, any parallel implementation of this step will vary but it will reflect limitations inherent to the sequential counterpart. Specifically, topology preserving methods, such as, LLE, will involve only local data and hence will be amenable to embarrassing parallelism with limited or no communication. Conversely, distance preserving methods will inevitably require a global data view and thus potentially more sophisticated parallelization strategies. Following our previous claim regarding complexity of Isomap we focus our presentation on the parallel implementation of this particular method.

The function F_G used in Isomap is based on the geodesic distance, which has been mathematically shown to be asymptotically equivalent to a graph distance in G (i.e., shortest path distance) [42]. However, the geodesic distance is a metric, while all-pairs shortest paths in directed graph G do not have to satisfy the symmetry condition. Therefore, to obtain W, special attention must be paid to how shortest path distances are used. More precisely, graph G must be transformed to ensure that it is symmetric. Note that after such transformation the graph is no longer regular; that is, certain nodes may have more than k neighbors (see Figure 1).

Taking into account the above requirements we obtain the following procedure of constructing W in parallel. First, all processors within the same row perform all-to-all communication to replicate graph G. As a result, each column of processors stores a copy of the entire graph G, that is, row-wise distributed between q processors in that column. Thanks to this, each processor can initialize its local part of W without further communication. After initialization W represents the distributed adjacency matrix of G, where $W_{ij} = \omega(x_i, x_j)$ if x_j is a neighbor of x_i, and $+\infty$ otherwise. In the next step symmetrization procedure is executed. Processors with coordinates (i, j) and (j, i), where $i \neq j$, exchange respective blocks of W and select element-wise minimum value between blocks. Similar operation is performed locally by processors on the diagonal, that is, processors for which $i = j$. At this stage W can be used to identify all-pairs shortest paths. Several parallel algorithms have been proposed to address this problem, targeting the PRAM model [43–45], shared memory architectures [46], and multi/many cores [47–49], as well as distributed memory machines [45, 50]. Among the existing parallel strategies we decided to adopt the checkerboard version of the parallel Floyd algorithm [46]. Briefly, the method proceeds in n iterations, where in each iteration every processor performs $O(n^2/p)$ operations to update its local block of W. All processors are synchronized in each iteration, owing to the fact that in iteration l, lth row and lth column of W have to be broadcasted. The broadcast is performed between processors that share the same row/column. After n iterations, matrix W stores all-pairs shortest path, which concludes the entire procedure.

Complexity of this phase is $O(n^3/p + n(t_s + t_m(n/\sqrt{p}))\log(\sqrt{p}))$ and is dominated by the parallel Floyd's algorithm. While replication and symmetrization of G can be executed efficiently in $O(n^2/p + t_s + t_m(n^2/p))$ time, all-pairs path searching involves extensive communication that slightly hinders scalability. Nevertheless, the algorithm remains scalable as long as $p < n^2/\log^2(n)$, which is true in the majority of real-life cases.

3.3. Matrix Normalization. The goal of normalization is to transform matrix W such that the resulting matrix A is both row and column centered; that is, $\sum_i A_{ij} = 0$ and $\sum_j A_{ij} = 0$. The normalization stage in all cases consists of matrix-matrix multiplication (see Table 1). However, in certain situations, especially in distance-preserving methods, explicit matrix multiplication can be avoided by taking advantage of structural properties of one of the matrices (e.g., the matrix H in Table 1). This is the case for, for example, PCA and Isomap, where we exploit the fact that matrices H and H^T are given analytically and thus can be generated in-place on each processor that requires them to perform multiplication. Consequently, the communication pattern inherent to the standard parallel matrix-matrix multiplication algorithms is simplified to one parallel reduction in the final dot-product operation. The complexity of this approach is $O(n^3/p + (t_s + t_m(n^2/p))\sqrt{p})$.

3.4. Finding Eigenvalues. Computing eigenvalues is the final step in the dimensionality reduction process. Although parallel eigensolvers are readily available, they are usually designed for shared memory and multi/many core architectures [51–54]. This unfortunately makes them impractical for our purposes. At the same time, existing distributed memory

solutions are not scalable and cannot handle large and dense data. For instance, one of the more popular packages, SLEPc [55], uses a simple 1D decomposition and in our tests did not scale to more than 4096 processors. A more recent library, elemental [56], which is still under development, offers 2D block-cyclic decomposition, but relies on a fixed block size (private communication). Finally, ELPA [57], which is the most promising library in this category, is also under development.

For these reasons we decided to implement a custom eigenvalue solver that exploits special properties of matrix A (symmetric, positive semidefinite) and computes only the first d eigenvalues. Our solver is based on the power method [31] and matrix deflation and is outlined in Algorithm 2. Note that power methods are considered easy but not efficient to parallelize. At the same time, however, they are at heart of several important real-life systems, for instance, Google's PageRank [58].

In general, our approach follows the standard scheme of the power method (lines 3–18), repeated d times to identify first d largest eigenvalues. After identification of an eigenvalue and its associated eigenvector, the matrix A is deflated—the contribution of the vector is removed from A (line 19). Observe that power method involves nested matrix-vector product (lines 6–11) that under normal circumstances would require parallel vector transposition. However, our parallel implementation benefits from the fact that A is symmetric, hence eliminating need for vector transposition. Indeed, the entire procedure consists of local matrix-vector product followed by all-reduce operation. Here, the reduction operation alternates between columns and rows as required to ensure that vector $\mathbf{x}$ is stored properly. Note that the power method is bounded by convergence criteria (line 5). In our case we use one of several popular conditions, which involves checking relative error between the current and previous estimate of the eigenvalue that can be performed every several iterations. We also note that convergence is significantly improved by using a matrix shifting strategy in the form $A = A - \delta I$, where δ is a positive number [59].

Extracting eigenvalue and eigenvector in iteration i (lines 11–18) depends on vectors u and z, while deflation step involves λ_i. Therefore, it is advantageous to replicate both u and z in their entirety on each processor. We achieve this with all-to-all communication executed by processors within the same row. This allows us to execute the deflation step in parallel, with each processor updating its local block of matrix A. Thus, the complexity of a single iteration of the power method is $O(n^3/p + (t_s + t_m(n/\sqrt{p}))\log(p))$, while the deflation step is $O(n^2/p + t_s\sqrt{p} + t_m n)$.

To conclude this section we would like to emphasize that our solver operates under the same assumptions as any power method. It requires that the first d eigenvectors of A are linearly independent, the initial vector $\mathbf{x}$ generated in ith iteration is not orthogonal to the eigenvector v_i, and finally, the first d eigenvalues are nondegenerate [31]. Note that these conditions are not restrictive and are easily satisfied in the context of dimensionality reduction.

Table 3: Run time in seconds for different p and varying problem sizes n. Due to memory limitations problem instance with $n = 32768$ could not be solved on less than $p = 256$ processors.

p	n			
	4096	8192	16384	32768
16	404.63	3492.28	45288.93	—
64	101.72	761.75	6906.64	—
256	33.99	263.24	1655.39	14613.33
1024	39.06	124.19	682.91	3964.65

4. Experimental Results

To assess scalability of our framework and test its performance in real-life applications, we performed a set of experiments using the TACC Ranger cluster [60]. A single node of this machine is based on AMD processors working at 2.3 GHz and provides 16 cores with 32 GB of DDR2 RAM and 512 KB of L2 cache per core. Nodes are connected by a multistage Infiniband network that offers 1 GB/s bandwidth. To compile all test programs and the framework, we used the Intel C++ 10.1 compiler with standard optimization flags and MVAPICH 1.0.1 MPI implementation. In every test we ran one MPI process per CPU core, which we refer to as processor.

4.1. Scalability Tests. In the first set of experiments we measured how problem size influences performance of our solution. We created a collection of synthetic datasets consisting of $n = \{4096, 8192, 16384, 32768\}$ points with $D = 10000$. Next, we performed Isomap dimensionality reduction using different numbers of processors. Obtained results are summarized in Table 3 and Figure 2.

The results show that our framework provides very good scalability for large problem sizes in the entire range of tested processor configurations. The super-linear speedup observed for $n = 16384$ is naturally explained by cache performance. Observe that the dominating computational factors in our framework are operations like matrix-matrix and matrix-vector products, which are well suited to exploit memory hierarchy. In our current implementation, we use direct SIMD-based implementation of these routines. A slightly weaker performance for small problem sizes and large number of processors can be attributed to network latency that offsets computational gains.

To further understand how different components of the framework perform, we measured their run time obtained for changing problem sizes. Table 4 shows that all modules scale as we would expect based on their theoretical complexity. The most time consuming stages are construction of the auxiliary matrix W for Isomap and normalization. This is not surprising taking into account that both components scale as $O(n^3)$ and the parallel Floyd's algorithm involves n rounds of communication. The abrupt performance decrease in the normalization stage, which can be observed for $n = 16384$, can be attributed to cache performance. Recall that normalization depends on matrix-matrix multiplication and hence is inherently sensitive to data locality. The final remark

Input: Matrix $A_{n\times n}$ and the required numbed of eigenvalues d.
2D mesh of $p = q \times q$ processors.
Output: Set of eigenvalues and eigenvectors of A,
$\Lambda_{0,\ldots,d-1} = \{\lambda_0, \lambda_1, \ldots, \lambda_{d-1}\}$ and $V_{0,\ldots,d-1} = \{v_0, v_1, \ldots, v_{d-1}\}$.
(1) Let $\mathbf{x}$ be a column-wise distributed vector in $\mathbb{R}^n$.
(2) **for** $i \leftarrow 0 : d - 1$ **do**
(3) Initialize $\mathbf{x}$ randomly. Processors within the same column use the same seed.
(4) $column \leftarrow true$
(5) **while** *not converged* **do**
(6) Compute $z = A\mathbf{x}$ locally.
(7) **if** $column = true$ **then**
(8) Perform column-wise all-reduce to obtain z.
(9) **else**
(10) Perform row-wise all-reduce to obtain z.
(11) **end if**
(12) $column \leftarrow \neg column$
(13) $\mathbf{x} \leftarrow z$
(14) **end while**
(15) Compute $u = Az$ as in steps (6)–(11).
(16) Replicate entire vector u and z on each processor.
(17) $\lambda_i \leftarrow \dfrac{z \cdot u}{z \cdot z}$
(18) $v_i \leftarrow \dfrac{z}{|z|_2}$
(19) Deflate local block of A: $A \leftarrow A - \lambda_i v_i v_i^T$.
(20) **end for**

Algorithm 2: 2D-Block Parallel Power Method.

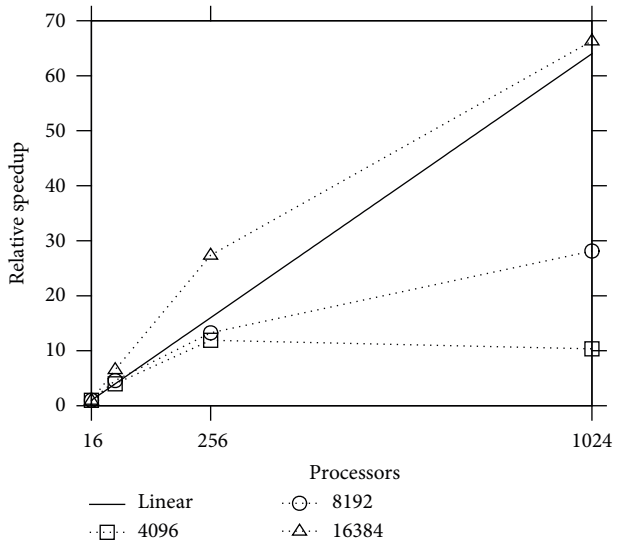

Figure 2: Relative speedup for different problem sizes n.

Table 4: Component-wise run time in seconds for varying problem sizes, and $p = 1024$ and $D = 10000$.

n	2048	4096	8192	16384	32768
KNN	0.623	1.389	5.721	22.254	86.706
W in Isomap	9.132	56.517	128.225	457.306	1697.124
Normalize	0.160	0.905	6.526	223.240	2546.11
Eigensolve	0.050	0.155	0.188	0.699	2.838

Table 5: Run time in seconds of KNN component for $n = 1024$ and different number of processors and varying D.

p	D			
	100	1000	10000	100000
16	0.053	0.530	5.466	92.014
64	0.015	0.115	1.373	22.984
256	0.005	0.027	0.349	5.860
1024	0.002	0.007	0.682	1.875

concerns k nearest neighbors module and eigenvalue solver. The KNN scales linearly with the data dimension D (see Table 5) and both modules can be used as standalone replacements whenever KNN or d largest eigenvalues problem has to be solved.

In the final test we compared our parallel eigensolver with SLEPc [55], one of the most popular and widely used libraries providing eigensolvers. SLEPc is an efficient and portable framework that offers an intuitive user interface. In many cases it is the first choice for solving large-scale sparse eigenvalue problems.

Table 6 shows that our implementation systematically outperforms SLEPc. This can be explained by two main factors: first, unlike SLEPc our implementation follows 2D data decomposition scheme, which offers better scalability, and second, we are seeking only the d largest eigenvalues.

4.2. Using Dimensionality Reduction to Explore Manufacturing Pathways. Solar cells, or plastic solar cells, manufactured from organic blends, that is, a blend of two polymers,

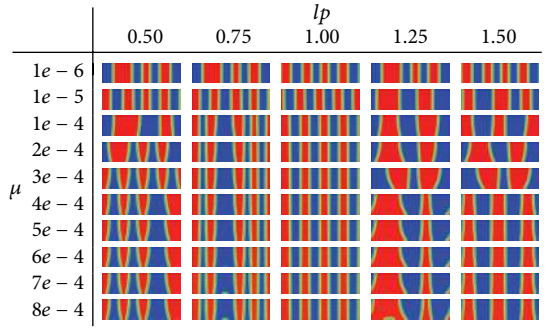

Figure 3: Snapshots of microstructures representing final morphologies from 50 different manufacturing processes.

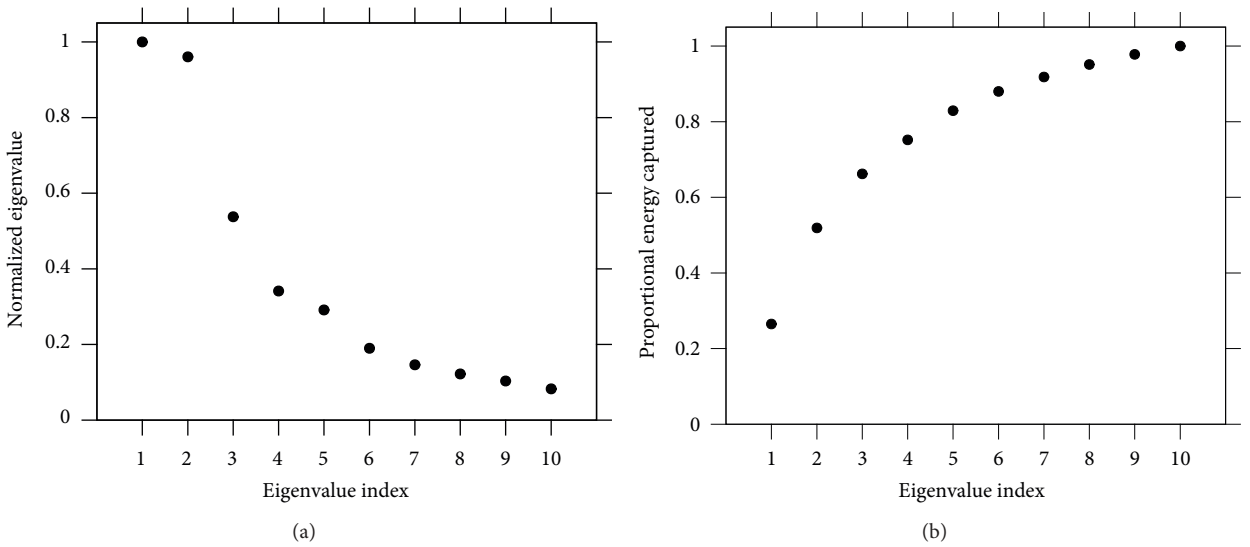

Figure 4: Scree plot of the ten largest eigenvalues (a) and their proportional energy covered (b).

Table 6: Comparison of our eigensolver with SLEPc. For $p = 1024$ SLEPc failed to execute.

p	$n = 1024$		$n = 4096$	
	Our solver	SLEPc	Our solver	SLEPc
16	0.0444	4.8159	2.5315	0.7049
64	0.0088	2.1666	0.6056	0.8134
256	0.0705	8.5538	0.1251	2.4143
1024	0.0742	*	0.1320	*
4096	0.0411	N/A	0.2024	10.9992

represent a promising low-cost, rapidly deployable strategy for harnessing solar energy. While highly cost-effective and flexible, their low power conversion efficiency makes them less competitive on a commercial scale in comparison with conventional inorganic solar cells. A key aspect determining the power conversion efficiency of organic solar cells is the morphological distribution of the two polymer regions in the device. Recent studies reveal that significant improvement in power conversion efficiency is possible through better morphology control of the organic thin film layer during the manufacturing process [29, 61–66]. High-throughput exploration of the various manufacturing parameters, for example, evaporation rate, blend ratio, substrate patterning frequency, substrate patterning intensity, and solvent, can potentially unravel process-morphology relationships that can help tailor processing pathways to obtain enhanced morphologies. Note that such high-throughput analysis results in datasets that are too large to visually inspect for trends and relationships. A promising approach towards unraveling process-morphology relationships in this high-throughput data is via dimensionality reduction. Here, we showcase our parallel framework on this pressing scientific problem. In particular, we focus on using dimensionality reduction to understand the effects of substrate patterning [67, 68], described by patterning frequency and intensity, on morphology evolution. We note that nanotip photolithography patterning of the substrate has shown significant potential to guide morphology evolution [69].

The input dataset consists of $n = 75150$ morphologies. Each morphology is a 2-dimensional snapshot which is vectorized to have dimensionality $D = 8326$. Figure 3 shows several representative final morphologies obtained by varying the patterning frequency, lp, from 0.5 to 1.50, and the intensity of the attraction/repulsion, μ, from $1 + 1e - 6$ to $1 + 8e - 4$ (in the remaining we omit leading 1 for clarity). In

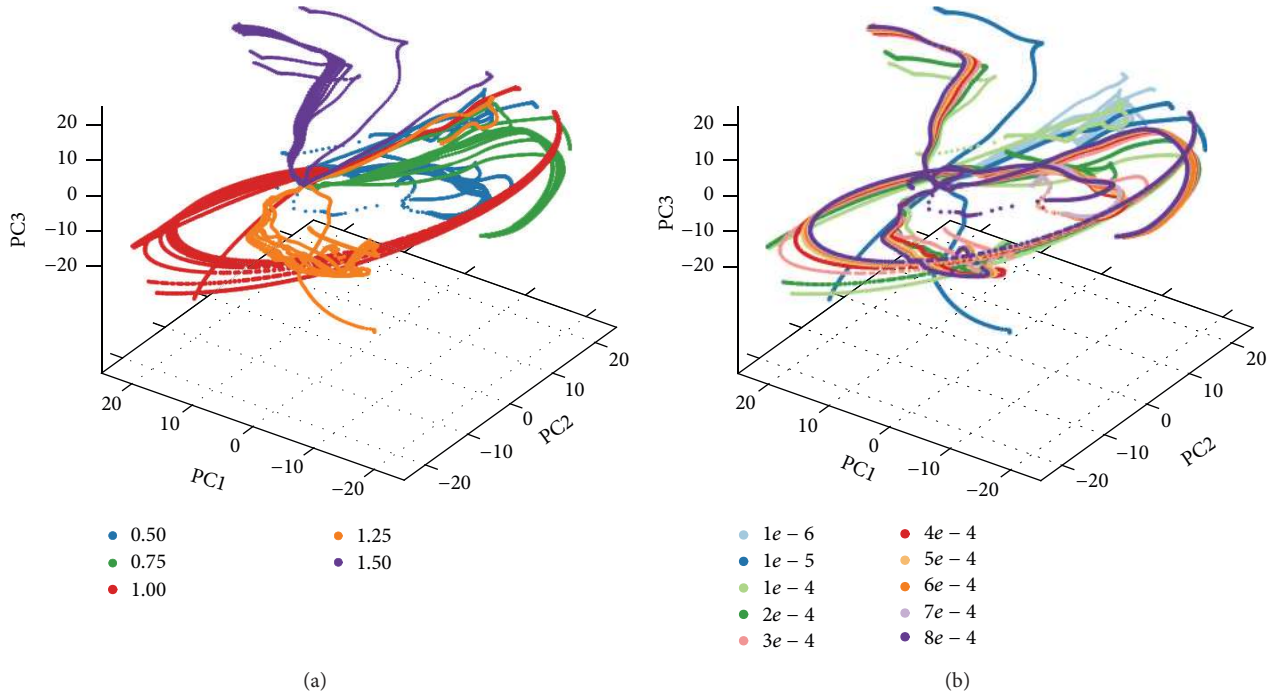

FIGURE 5: Morphology evolution as captured by the first three principal components of the original data. Different colors represent different patterning frequency lp (left) and different patterning intensity μ (right).

all these cases, the lower surface of the domain is patterned to attract and/or repel specific classes of polymers, thus affecting the morphology. We performed dimensionality reduction on this dataset using $p = 16384$ processors on TACC Ranger. The total run time was 1058 seconds.

Figure 4 plots the first 10 eigenvalues of the data. Note that the first three eigenvalues, and hence the first three principle components of the data, represent ~70% of the information content of the entire data. We therefore characterize each morphology in terms of this three-dimensional representation.

Figure 5 represents *all the morphologies* in such reduced space. In the plot to the left, the points are color coded according to the patterning frequency used, while in the plot to the right, the points are color coded according to the patterning intensity. This plot provides significant visual insight into the effects of patterning frequency and intensity. There exists a central plane of patterning frequency, where the morphology evolution is highly regulated irrespective of the patterning intensity ($lp \leq 1$). This is particularly valuable information as the patterning frequency is much easier to control than patterning intensity from a manufacturing perspective. For patterning frequencies above $lp = 1$, the morphologies are highly sensitive to slight variations in both frequency and intensity. This is also clearly seen in Figure 6, where slight variations in the intensity give dramatically different final morphologies. Notice also that higher intensity does not necessarily give different morphologies. This is a very important insight that allows us to preclude further, potentially expensive, exploration of the phase space of increasing patterning intensity.

Finally, the low-dimensional plots illustrate the ability to achieve the same morphology using different processing conditions. For instance, in Figure 7, we isolate the morphology evolution under two processing conditions that result in identical morphologies. Such correlations, most sensitive regions versus least sensitive regions (Figure 6) and configurations resulting in identical morphologies, are enormously useful as we tailor processing pathways to achieve designer morphologies. This analysis illustrates the power of parallel dimensionality reduction methods to achieve this goal. We defer a comprehensive physics-based analysis of this dataset to a subsequent publication.

5. Conclusions

In this work we illustrate a systematic analysis of dimensionality reduction techniques and recast them into a unified view that can be exploited by dimensionality reduction algorithm designers. We subsequently identify the common computational building blocks required to implement a spectral dimensionality reduction method. We use this insight to design and implement a parallel framework for dimensionality reduction that can handle large datasets and scales to thousands of processors. We demonstrate the capability and scalability of this framework on several test datasets. We finally showcase the applicability and potential of the framework towards unraveling complex process-morphology relationships in the manufacturing of plastic solar cells.

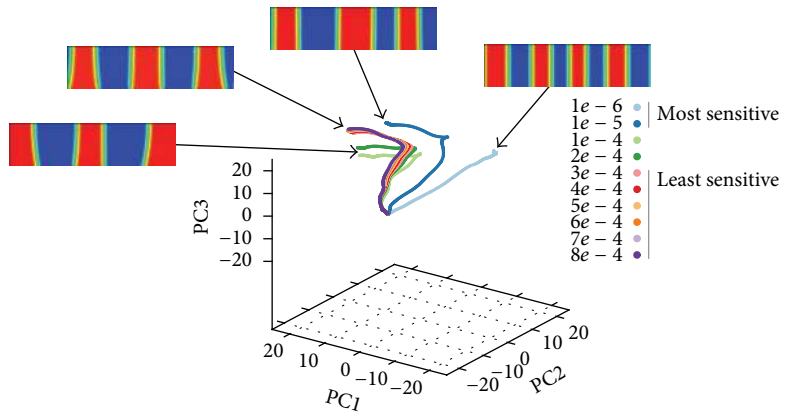

Figure 6: Morphology evolution for $lp = 1.50$, and categorization of parametric space.

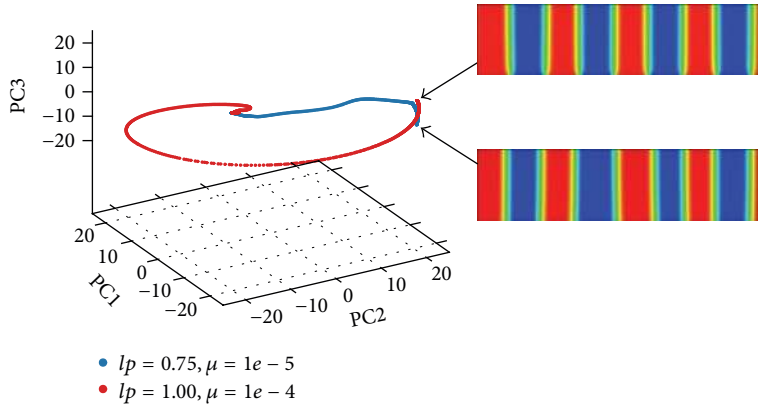

Figure 7: Multiple pathways of morphology evolution.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research is supported in part by the National Science Foundation through XSEDE resources provided by TACC under Grant no. TG-CTS110007 and supported in part by NSF PHY-0941576 and NSF-1149365.

References

[1] J. L. Lumley, "Atmospheric turbulence and radio wave propagation," in *The Structure of Inhomogeneous Turbulent Flows*, 1967.

[2] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[3] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.

[4] M. L. Davison, *Multidimensional Scaling*, John Wiley & Sons, New York, NY, USA, 1983.

[5] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[6] R. N. Shepard, "The analysis of proximities: multidimensional scaling with an unknown distance function. I," *Psychometrika*, vol. 27, no. 2, pp. 125–140, 1962.

[7] W. S. Torgerson, "Multidimensional scaling. I. Theory and method," *Psychometrika*, vol. 17, no. 4, pp. 401–419, 1952.

[8] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[9] D. L. Donoho and C. Grimes, "Hessian eigenmaps: new locally linear embedding techniques for high-dimensional data," *Proceedings of the National Academy of Sciences*, vol. 100, pp. 5591–5596, 2003.

[10] J. W. Sammon Jr., "A nonlinear mapping for data structure analysis," *IEEE Transactions on Computers*, vol. C-18, no. 5, pp. 401–409, 1969.

[11] T. Kohonen, *Self-Organizing Maps*, Springer, 2001.

[12] P. Demartines and J. Hérault, "Curvilinear component analysis: a self-organizing neural network for nonlinear mapping of data sets," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 148–154, 1997.

[13] V. de Silva and J. B. Tenenbaum, "Sparse multidimensional scaling using landmark points," Tech. Rep., Stanford University, 2004.

[14] A. Talwalkar, S. Kumar, and H. Rowley, "Large-scale manifold learning," in *Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition (CVPR '08)*, pp. 1–8, 8, June 2008.

[15] M. H. Yang, "Face recognition using extended Isomap," in *Proceedings of the International Conference on Image Processing*, vol. 2, pp. 117–120, 2002.

[16] J. Zhang, S. Z. Li, and J. Wang, "Nearest manifold approach for face recognition," in *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition (FGR '04)*, pp. 223–228, May 2004.

[17] V. Silva and J. B. Tenenbaum, "Global versus local methods in nonlinear dimensionality reduction," in *Advances in Neural Information Processing Systems*, vol. 15, pp. 705–712, 2002.

[18] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2625, 2008.

[19] L. J. P. van der Maaten, E. O. Postma, and H. J. van den Herik, "Dimensionality reduction: a comparative review," Tech. Rep., Maastricht University, 2009.

[20] M. Andrecut, "Parallel GPU implementation of iterative PCA algorithms," *Journal of Computational Biology*, vol. 16, no. 11, pp. 1593–1599, 2009.

[21] F. Zhong, D. W. Capson, and D. C. Schuurman, "Parallel architecture for PCA image feature detection using FPGA," in *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE '08)*, pp. 1341–1344, May 2008.

[22] S.-H. Bae, J. Y. Choi, J. Qiu, and G. C. Fox, "Dimension reduction and visualization of large high-dimensional data via interpolation," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, pp. 203–214, June 2010.

[23] S. Ingram, T. Munzner, and M. Olano, "Glimmer: multilevel MDS on the GPU," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 2, pp. 249–261, 2009.

[24] T. T. Yeh, T.-Y. Chen, Y.-C. Chen, and W.-K. Shih, "Efficient parallel algorithm for nonlinear dimensionality reduction on GPU," in *Proceedings of the IEEE International Conference on Granular Computing*, pp. 592–597, August 2010.

[25] J. A. Lee and M. Verleysen, *Nonlinear Dimensionality Reduction*, Information Science and Statistics, Springer, New York, NY, USA, 2007.

[26] S. Bergman, *The Kernel Function and Conformal Mapping*, American Mathematical Society, New York, NY, USA, 1950.

[27] A. Fontanini, M. G. Olsen, and B. Ganapathysubramanian, "Thermal comparison between ceiling diffusers and fabric ductwork diffusers for green buildings," *Energy and Buildings*, vol. 43, no. 11, pp. 2973–2987, 2011.

[28] Y. Xie, H. Hu, and B. Ganapathysubramanian, "Phase transitions in vortex shedding in the wake of a heated circular cylinder at low Reynolds number," in *Proceedings of the 3rd Annual Graduate Research Symposium*, 2011.

[29] O. Wodo, S. Tirthapura, S. Chaudhary, and B. Ganapathysubramanian, "A graph-based formulation for computational characterization of bulk heterojunction morphology," *Organic Electronics*, vol. 13, no. 6, pp. 1105–1113, 2012.

[30] J. B. Kruskal and M. Wish, *Multidimensional Scaling*, Sage Publications, Beverly Hills, Calif, USA, 1978.

[31] G. H. Golub and C. F. van Loan, *Matrix Computations*, The John Hopkins University Press, 1996.

[32] B. Hariharan and S. Aluru, "Efficient parallel algorithms and software for compressed octrees with applications to hierarchical methods," *Parallel Computing*, vol. 31, no. 3-4, pp. 311–331, 2005.

[33] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[34] L. K. Saul and S. T. Roweis, "Think globally, fit locally: unsupervised learning of low dimensional manifolds," *Journal of Machine Learning Research*, vol. 4, no. 2, pp. 119–155, 2004.

[35] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," *Journal of Research of the National Bureau of Standards*, vol. 45, pp. 255–282, 1950.

[36] Y. Saad, "Krylov subspace methods for solving large unsymmetric linear systems," *Mathematics of Computation*, vol. 37, no. 155, pp. 105–126, 1981.

[37] B. N. Parlett and W. G. Poole Jr., "A geometric theory for the QR, LU and power iterations," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 389–412, 1973.

[38] J. H. Wilkinson, *Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, UK, 1965.

[39] Q. Guo, D. Rajewski, E. Takle, and B. Ganapathysubramanian, "Constructing lowdimensional stochastic wind models from meteorology data," *Renewable Energy*. Submitted.

[40] B. Hendrickson and S. Plimpton, "Parallel many-body simulations without all-to-all communication," *Journal of Parallel and Distributed Computing*, vol. 27, no. 1, pp. 15–25, 1995.

[41] A. Sarje, J. Zola, and S. Aluru, "Accelerating pairwise computations on cell processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 1, pp. 69–77, 2011.

[42] M. Bernstein, V. de Silva, J. C. Langford, and J. B. Tenenbaum, "Graph approximations to geodesics on embedded manifolds," Tech. Rep., Stanford University, 2000.

[43] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders, "A parallelization of Dijkstra's shortest path algorithm," in *Mathematical Foundations of Computer Science*, vol. 1450 of *Lecture Notes in Computer Science*, pp. 722–731, Springer, Berlin, Germany, 1998.

[44] K. Madduri, D. A. Bader, J. W. Berry, and J. R. Crobak, "An experimental study of a parallel shortest path algorithm for solving large-scale graph instances," in *Proceedings of the Workshop on Algorithm Engineering and Experiments*, pp. 23–35, 2007.

[45] U. Meyer and P. Sanders, "δ-stepping: a parallel single source shortest path algorithm," in *Proceedings of the 6th Annual European Symposium on Algorithms*, pp. 393–404, Venice, Italy, August 1998.

[46] J. F. Jenq and S. Sahni, "All pairs shortest paths on a hypercube multiprocessor," in *Proceedings of the International Conference on Parallel Processing*, pp. 713–716, August 1987.

[47] A. Buluç, J. R. Gilbert, and C. Budak, "Solving path problems on the GPU," *Parallel Computing*, vol. 36, no. 5-6, pp. 241–253, 2010.

[48] D. Delling, A. V. Goldberg, A. Nowatzyk, and R. F. Werneck, "PHAST: hardware-accelerated shortest path trees," *Journal of Parallel and Distributed Computing*, vol. 73, no. 7, pp. 940–952, 2013.

[49] G. J. Katz and J. T. Kider Jr., "All-pairs shortest-paths for large graphs on the GPU," in *Proceedings of the 23rd SIGGRAPH/Eurographics Conference on Graphics Hardware*, pp. 47–55, June 2008.

[50] P. A. Humblet, "Another adaptive distributed shortest path algorithm," *IEEE Transactions on Communications*, vol. 39, no. 6, pp. 995–1003, 1991.

[51] S. Balay, J. Brown, K. Buschelman et al., *PETSc Users Manual Revision 3.5*, 2010.

[52] E. Breitmoser and A. Sunderland, "An overview of eigensolvers for HPCx," Tech. Rep., HPCx Consortium, 2003.

[53] J. Choi, J. Demmel, I. Dhillon et al., "ScaLAPACK: a portable linear algebra library for distributed memory computers—design issues and performance," *Computer Physics Communications*, vol. 97, no. 1-2, pp. 1–15, 1996.

[54] I. S. Dhillon, B. N. Parlett, and C. Vömel, "The design and implementation of the MRRR algorithm," *ACM Transactions on Mathematical Software*, vol. 32, no. 4, pp. 533–560, 2006.

[55] V. Hernandez, J. E. Roman, and V. Vidal, "SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Transactions on Mathematical Software*, vol. 31, no. 3, pp. 351–362, 2005.

[56] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, "Elemental: a new framework for distributed memory dense matrix computations," *ACM Transactions on Mathematical Software*, vol. 39, no. 2, pp. 1–24, 2013.

[57] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, and P. Willems, "Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem," *Journal of Computational Science*, vol. 2, no. 3, pp. 272–278, 2011.

[58] K. Bryan and T. Leise, "The #5,000,000,000 eigenvector: the linear algebra behind Google," *SIAM Review*, vol. 48, no. 3, pp. 569–581, 2006.

[59] K. Meerbergen, A. Spence, and D. Roose, "Shift-invert and Cayley transforms for detection of rightmost eigenvalues of nonsymmetric matrices," *BIT Numerical Mathematics*, vol. 34, no. 3, pp. 409–423, 1994.

[60] D. S. Katz, S. Callaghan, R. Harkness et al., "Science on the TeraGrid," *Computational Methods in Science and Technology*, vol. 1, pp. 81–97, 2010.

[61] L.-M. Chen, Z. Hong, G. Li, and Y. Yang, "Recent progress in polymer solar cells: Manipulation of polymer: fullerene morphology and the formation of efficient inverted polymer solar cells," *Advanced Materials*, vol. 21, no. 14-15, pp. 1434–1449, 2009.

[62] C. Deibel, V. Dyakonov, and C. J. Brabec, "Organic bulk-heterojunction solar cells," *IEEE Journal on Selected Topics in Quantum Electronics*, vol. 16, no. 6, pp. 1517–1527, 2010.

[63] H. Hoppe and N. S. Sariciftci, "Morphology of polymer/fullerene bulk heterojunction solar cells," *Journal of Materials Chemistry*, vol. 16, no. 1, pp. 45–61, 2006.

[64] S. H. Park, A. Roy, S. Beaupré et al., "Bulk heterojunction solar cells with internal quantum efficiency approaching 100%," *Nature Photonics*, vol. 3, no. 5, pp. 297–303, 2009.

[65] J. Peet, A. J. Heeger, and G. C. Bazan, "'Plastic' solar cells: self-assembly of bulk heterojunction nanomaterials by spontaneous phase separation," *Accounts of Chemical Research*, vol. 42, no. 11, pp. 1700–1708, 2009.

[66] S. E. Shaheen, C. J. Brabec, N. S. Sariciftci, F. Padinger, T. Fromherz, and J. C. Hummelen, "2.5% efficient organic plastic solar cells," *Applied Physics Letters*, vol. 78, no. 6, pp. 841–843, 2001.

[67] O. Wodo and B. Ganapathysubramanian, "Computationally efficient solution to the Cahn-Hilliard equation: adaptive implicit time schemes, mesh sensitivity analysis and the 3D isoperimetric problem," *Journal of Computational Physics*, vol. 230, no. 15, pp. 6037–6060, 2011.

[68] O. Wodo and B. Ganapathysubramanian, "Modeling morphology evolution during solvent-based fabrication of organic solar cells," *Computational Materials Science*, vol. 55, pp. 113–126, 2012.

[69] D. C. Coffey and D. S. Ginger, "Patterning phase separation in polymer films with dip-pen nanolithography," *Journal of the American Chemical Society*, vol. 127, no. 13, pp. 4564–4565, 2005.