Streaming GPU singular value and dynamic mode decompositions

Seth D. Pendergrass  J. Nathan Kutz  Steven L. Brunton *
University of Washington  University of Washington  University of Washington
Computer Engineering  Applied Mathematics  Mechanical Engineering

July 29, 2017

Abstract

The dynamic mode decomposition (DMD) has recently emerged as a promising method for foreground/background modeling in real-time video applications. This work develops a parallelized algorithm to compute DMD on a graphics processing unit using the streaming method of snapshots singular value decomposition. This allows the algorithm to operate efficiently on streaming data by avoiding redundant inner-products as new data becomes available. In addition, it is possible to leverage the native compressed format of many data streams, such as HD video and computational physics codes that are represented sparsely in the Fourier domain, to massively reduce data transfer from CPU to GPU and to enable sparse matrix multiplications. Taken together, these algorithms facilitate real-time streaming DMD on high-dimensional data streams. We demonstrate the proposed method on numerous high-dimensional video datasets for background subtraction, and provide comparisons against standard algorithms for the SVD, DMD and background subtraction. However, this method may accelerate numerous other algorithms based on the SVD or DMD. The computational framework is developed as an open-source library written in C++ and CUDA to promote reproducible research.

Keywords: Singular Value Decomposition, Dynamic Mode Decomposition, GPU, Background Subtraction, Streaming

1 Introduction

Real-time video processing is a pressing challenge of the modern era, with significant technological and societal impact. Applications include security, surveillance, ecological monitoring, and autonomous vehicles, to name a few. In nearly all applications, there is a growing demand for higher-resolution imaging, exemplified by 4K video. These vast and increasing volumes of data pose a tremendous challenge for current computational algorithms, despite the growth of modern computational power. However, the complexity of the underlying scene is typically of much lower dimension than the ambient measurement dimension.

In video applications, foreground/background modeling is a primary task. The quality of the foreground separation impacts many downstream tasks, such as object classification, and the computational time required limits the resources available for downstream processing. However, foreground/background modeling is a computationally expensive task, which only becomes more challenging with increased resolution (Bouwmans, 2014; Bouwmans and Zahzah, 2014; Bouwmans et al., 2016). Many algorithms have been developed to accomplish this task at various levels of accuracy and speed (Bouwmans et al., 2016). Candès et al. (2011) framed the problem of background subtraction as a separation of the input matrix into its sparse foreground and low-rank background components, using robust principle component analysis (RPCA). Although RPCA is expensive, performing an iterative singular value decomposition (SVD) until convergence on a final result, the formulation of the foreground/background problem in terms of matrix decomposition has a number of advantages. Matrix decomposition methods, such as the SVD and RPCA benefit from parallelization and modern randomized methods that scale with the complexity of the signal rather than the ambient measurement dimension.

The dynamic mode decomposition (DMD) is a recent matrix decomposition that has shown promise as a potential method for real-time foreground/background separation in high-resolution video streams (Grosek and Kutz, 2014; Erichson et al., 2016). DMD is closely related to RPCA for video applications, and it only requires a single SVD computation, as opposed to an iterative SVD procedure. DMD was first introduced by Schmid in the fluids community (Schmid, 2010) as a data-driven method to decompose complex fluid systems into spatiotemporal coherent structures, where each mode is associated with a particular frequency and rate of growth or decay. DMD has since been rigorously connected to nonlinear dynamical systems via Koopman operator theory (Rowley et al., 2009; Kutz et al., 2014), which provides an alternative infinite-dimensional linear representation of nonlinear dynamical systems (Koopman, 1931; Mezić, 2005, 2013). DMD may also be thought of as an algo-

*Corresponding author: +1 (609)-921-6415 sbrunton@uw.edu
Despite the growing success of DMD, the underlying algorithm is based on an expensive singular value decomposition on high-dimensional data. Moreover, in many applications, such as video processing and high-performance computations of transient physical processes, a sliding window DMD computation must be performed repeatedly for streaming data. Many algorithms have been proposed to increase the speed of the SVD and DMD algorithms. Sayadi and Schmid (2016) proposed using a parallel QR factorization as the basis for a parallel SVD on tall-skinny matrices, as are common in scientific computing and video processing. Hemati et al. (2014) developed a batch-process and POD compressed version of the DMD, in order to accommodate large data streams. Brand (2002) created the incremental SVD, a method for updating an SVD to adjust for new data. Wang et al. (2016) propose a partitioned method of snapshots SVD that is readily parallelized. Kutz et al. (2016c); Erichson et al. (2016); Bistrian and Navon (2017) used compression or randomized techniques in order to reduce the size of the matrix DMD is performed on. In Kutz et al. (2014) it was shown that the computational bottleneck in the DMD, when computing the singular value decomposition using the method of snapshots (Sirovich, 1987), is the calculation of the inner product matrix on high-dimensional data. They further note that when computing DMD on a sequential timeseries, many redundant inner products may be avoided from one timestep to the next. The focus of this paper is to develop a new streaming DMD algorithm, designed to eliminate redundant computations when repeatedly performing the DMD on a sequence of data. Thus by copying these shared elements rather than recalculating them, a massive speed-up can be realized. The present work synthesizes and builds on many of these ideas, providing an accelerated DMD computation using a streaming method of snapshots SVD, parallelized on a GPU, and extendable to work directly on compressed data.

1.1 Contributions

In this paper, we develop a streaming DMD algorithm, shown in Fig. 1 designed to reuse computations when processing sequential inputs. The core of this algorithm is the streaming SVD based on the method of snapshots, shown in Fig. 2 which we compare to a standard SVD algorithm, demonstrating considerable speed up with negligible loss in accuracy. We also demonstrate a new, efficient way to calculate DMD mode amplitudes on POD coefficients, as opposed to the traditional high-dimensional least-squares fit. Additionally, we implement both CPU and GPU versions of streaming DMD and show that these algorithms are well suited to parallel processing. We compare the GPU implementation of the streaming DMD against a non-streaming CPU implementation, with negligible difference in outcome. Further, we design this architecture to work with the native compressed format of many data streams (e.g. in the Fourier domain) to reduce data transfer size and leverage sparse matrix multiplications. Many of the innovations developed for streaming, GPU, compressed DMD are also equally valid for the SVD, and may have significant impact on scientific computing. The C++ package for the streaming DMD and SVD algorithms is available under an open-source license on GitHub to promote reproducible research, at https://github.com/sethdp/libssvd.

This paper is organized as follows: First, we review background material, including the method of snapshots SVD and the DMD in section 2. We also discuss the motivation for graphics processing unit (GPU) acceleration for our algorithms. Next, in section 3 we explain our core innovations, including the streaming SVD and DMD, fast computation of DMD mode amplitudes, our implementations, and leveraging compressed data formats. In section 4 we show the significant performance improvements made by our streaming algorithms and the similar accuracy on video datasets. Lastly, in section 5 we sum-
2 Background

In order to develop our streaming DMD algorithm, we first provide an overview of the standard DMD, the method of snapshots SVD and general purpose GPU computing. The backbone of our streaming versions of the SVD and DMD is the method of snapshots SVD.

In all of the analysis that follows, we consider a matrix of data snapshots $X \in \mathbb{R}^{m \times n}$,

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$$

(1)

where $n$ is the number of measurements and $m$ is the number of temporal snapshots. For example, if the columns of $X$ represent image frames in a movie, then $n$ is the number of pixels per frame (i.e., $n = n_x n_y$, where the image has $n_x \times n_y$ pixels, and the image is reshaped into an $n \times 1$ vector) and $m$ is the number of frames in the movie.

Similarly, we may consider a time-series of an evolving spatial field from a numerical simulation of a partial differential equation.

Figure 3: Comparison between the method of snapshots SVD and the standard SVD of singular values from the standard and method of snapshots SVDs. This comparison was made on the first Yale face sequence (Belhumeur et al., 1996). See Figure 4 for the corresponding image reconstruction.

Figure 4: Comparison of the standard and method of snapshots SVDs for image reconstruction with one, half or all singular values and vectors. Created using the first Yale face sequence (Belhumeur et al., 1996). See Figure 3 for the corresponding comparison of singular values.

2.1 Method of Snapshots Singular Value Decomposition

The method of snapshots is an alternative way to calculate the singular value decomposition of a matrix $X$,

$$X = U \Sigma V^*,$$

(2)

developed for matrices where one dimension is much larger than the other. This method was originally developed for data from fluid dynamics, in which the target matrices are significantly taller than they are wide (Sirovich, 1987), i.e. $m \gg n$. In these applications, it is observed that the nonzero eigenvalues of $X^*X$ are the same as those of $XX^*$, although the first matrix is size $n \times n$ while the second matrix is size $m \times m$. It is computationally more efficient to compute the eigendecomposition of the smaller matrix $X^*X$ and then use this information to reconstruct the left and right singular vectors of $X$. This allows for significant reductions in computation time, although with a potential reduction in accuracy. The method of snapshots is summarized as follows:

1. Multiply $X$ by its transpose, in whichever order creates the smallest output. We assume $X$ is a tall-skinny matrix (i.e., $m \gg n$). Then find the eigendecomposition:

$$X^*XV = \Lambda$$

(3)

where $\Lambda$ are the eigenvalues and $V$ the eigenvectors of $X^*X$. The non-negative square roots of $\Lambda$ are the singular values $\Sigma$ of the original matrix $X$.

2. The left singular vectors $U$ are calculated as follows:

$$U = XV\Sigma^{-1}.$$
This creates an “economy” SVD, where $U \in \mathbb{R}^{m \times n}$ is the same dimension as $X$, and $\Sigma \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{n \times n}$ are both small square matrices. Figure 3 shows the singular values calculated with both the standard SVD and the method of snapshots, performed on the Yale faces dataset [Belhumeur et al., 1996]. The method of snapshots is a standard technique in the fluid dynamics community due to the high aspect ratio of the data matrix. Figure 4 compares the reconstruction of the Yale faces [Belhumeur et al., 1996] between the standard SVD and method of snapshots SVD, as well as the absolute difference between the two. This further demonstrates how close the method of snapshots is to the standard SVD, regardless of the number of eigenvalues and eigenvectors used to reconstruct the images. In turn, the speed-up provided by the method of snapshots SVD can be carried over to the DMD.

The Läuchli matrix [Läuchli, 1961] is one example where the Method of Snapshots SVD can lose significant accuracy, due to the limited precision of floating point representations. The Läuchli matrix $X \in \mathbb{R}^{(n+1) \times n}$ is as follows:

$$X = \begin{bmatrix}
1 & \ldots & 1 \\
\varepsilon & 0 \\
\vdots & & 0 \\
0 & \varepsilon
\end{bmatrix}$$

where $\varepsilon$ is much smaller than 1. When multiplied by its transpose, the Läuchli matrix will create a matrix with all off diagonals equal to 1 and the diagonal equal to $1 + \varepsilon^2$. Thus when $\varepsilon \leq \sqrt{\varepsilon}$, where $\varepsilon$ is the machine epsilon, $1 + \varepsilon^2$ will equal 1 in floating point arithmetic. However, we find that this is not often an issue in practice. Consider a matrix representing 8 bit per pixel normalized image data: the smallest non-zero value of $\varepsilon$ is $1/255 \approx 0.0039$. This will be greater than the machine epsilon for 32-bit floating point numbers, and thus will not lose the contributions from $\varepsilon$.

### 2.2 Dynamic Mode Decomposition

The DMD arose out of the fluid dynamics community to analyze the spatio-temporal coherent structures arising from fluids data [Schmid 2010]. It quickly gained popularity as strong connections were made between DMD and Koopman spectral analysis [Rowley et al., 2009; Kutz et al., 2014; Alekseev et al., 2016; Kutz et al., 2016a], which provides an infinite-dimensional linear representation of nonlinear dynamical systems [Koopman, 1931; Mezić, 2005; 2013].

DMD finds the dominant eigenvalues and eigenvectors of a best-fit linear dynamical system modeling the transition of a state $x_t$ to the next time-step $x_{t+1}$; nonlinear model reduction is also possible with similar data [Brunton et al., 2016b]. In particular, given a matrix $X$ and another matrix $X'$ consisting of the snapshots one-time step in the future:

$$X' = \begin{bmatrix}
x_2 \\
x_3 \\
\vdots \\
x_{n+1}
\end{bmatrix}$$

the DMD algorithm obtains the eigendecomposition of the best-fit linear operator $A$ given by

$$A = X'X'^\dagger = X'V\Sigma^{-1}U^*,$$

where $\dagger$ denotes the Moore-Penrose pseudo inverse [Kutz et al., 2014].

However, since the state dimension $n$ may be quite large (on the order of a million for HD video, tens of millions for 4K video, and even larger for scientific computing applications), the matrix $A$ is too large to directly analyze on simple computational architectures. Instead, it is possible to analyze a smaller matrix $\hat{A}$ obtained via projection onto the left singular vectors in $U$:

$$\hat{A} = U^*AU = U^*X'V\Sigma^{-1}.$$  

Much like the method of snapshots, the matrix $\hat{A}$ is size $n \times n$, and it has the same eigenvalues as the high-dimensional matrix $A$, as shown in [Kutz et al., 2014]. Taking the eigendecomposition

$$\hat{A}W = W\Lambda$$

it is then possible to obtain eigenvectors of the original high-dimensional matrix $A$ via

$$\Phi = X'V\Sigma^{-1}W.$$  

The columns of $\Phi$ are called dynamic modes of $X$ and they are spatio-temporal modes that have a single temporal signature given by the corresponding eigenvalue $\lambda$ in $A$.

The large number of independent inner-products performed in the process of calculating the SVD and DMD make it a perfect fit for being computed on a graphics processing unit (GPU), where their many cores can be leveraged.

#### 2.2.1 DMD for Video Background Subtraction

Grosek and Kutz [2014] show that the DMD can be effectively leveraged to compute decomposition of a video into the foreground and background components. This provides a similar decomposition as in the robust principle component analysis (RPCA) [Candes et al., 2011], but at a fraction of the cost, as RPCA involves an iterative procedure requiring dozens of SVD computations. In this framework, the video $X$ is decomposed into its constituent low-rank and sparse components, where the low-rank contains a low-dimensional representation of the system under observation and the sparse the outliers, noise and/or corruption measured by the input. This is represented as:

$$X = L + S,$$
where $\mathbf{L}$ is the low-rank component (background) and $\mathbf{S}$ is the sparse component (foreground).

Because each DMD mode has a corresponding frequency given by the DMD eigenvalue $\lambda$, the discrete-time eigenvalues that are nearly equal to 1 correspond to modes that do not change from frame to frame, i.e., the background modes. Thus, DMD can also be used to split the matrix $\mathbf{X}$ into two components, corresponding to slowly varying modes with eigenvalues $\lambda_p \approx 1$, and those that have faster dynamics:

$$
\mathbf{X} = \sum_p b_p \phi_p \lambda_p^{t-1} + \sum_{j \neq p} b_j \phi_j \lambda_j^{t-1}, \tag{12}
$$

where $t = [1 \ 2 \ \cdots \ n]$ is a vector of time indices. Refer to Erichson et al. (2016) for the state of the art DMD implementation of background modeling.

### 2.3 General Purpose GPU Computing

We will provide a brief overview of general purpose GPU (GPGPU) programming here, but refer the reader to Saunders and Kandrot (2010) for in-depth information on GPGPU programming with CUDA. General purpose GPU computing has proven effective for accelerating many linear algebra problems, as it is able to perform many operations in parallel. Creating an efficient algorithm for use on a modern GPU requires a very different approach than would be used on a central processing unit (CPU). This design allows a GPU to achieve a much higher throughput than a CPU (NVIDIA, 2015), if the algorithm is written with the GPU in mind. This Single-Instruction, Multiple Data (SIMD) style of code works best when there is a large amount of input data needing to be independently processed. NVIDIA (2015) notes that minimizing host (CPU) to device (GPU) memory transfers is key to maximizing performance. This lends itself naturally to streaming algorithms where only the updated data need be transferred on or off the device.

### 3 Streaming Algorithms

In many applications, data is continually acquired from sensors in a streaming fashion; new data is appended as columns to the right of the matrix $\mathbf{X}$, while old columns may be removed from the left of $\mathbf{X}$ if necessary. In streaming applications, such as online video processing or windowed DMD on transient simulations, the cost of repeated DMD and SVD calculations may be prohibitively expensive.

Here, we build a suite of complementary techniques to accelerate repeated SVD and DMD computations for streaming data. The core of the streaming DMD algorithm is the streaming method of snapshots SVD, whereby redundant inner product calculations in $\mathbf{X}'\mathbf{X}'$ are reused from one timestep to the next, reducing the SVD computational complexity from $\mathcal{O}(mn^2)$ to $\mathcal{O}(mn)$. The streaming SVD and DMD are discussed in subsection 3.1 and subsection 3.2 respectively. When it is necessary to compute the mode amplitudes in $\mathbf{b}$, we introduce an efficient computation in subsection 3.3. All of the above methods are readily parallelized, and we discuss GPGPU implementation in subsection 3.4. Once GPU parallelized algorithms have been implemented, data transfer from the CPU to GPU becomes the main computational bottleneck. However, in many applications it is possible to leverage the native sparse representation of the data (e.g., image sequences are stored in compressed Fourier or wavelet representations) to significantly reduce data transfer and promote sparse matrix operations, further reducing the computational burden. This is discussed in subsection 3.5.

#### 3.1 Streaming SVD

In the streaming context, let $\mathbf{X}$ be the current data matrix and $\mathbf{X}'$ be the next matrix in the sequence. Many of the inner products in $\mathbf{X}'\mathbf{X}'$, shown in blue, may be reused in $\mathbf{X}'\mathbf{X}'$:

$$
\mathbf{X}'\mathbf{X}' = 
\begin{pmatrix}
\langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \cdots & \langle x_1, x_{n-1} \rangle \\
\langle x_2, x_1 \rangle & \langle x_2, x_2 \rangle & \cdots & \langle x_2, x_{n-1} \rangle \\
\vdots & \vdots & \ddots & \vdots \\
\langle x_{n-1}, x_1 \rangle & \langle x_{n-1}, x_2 \rangle & \cdots & \langle x_{n-1}, x_{n-1} \rangle \\
\langle x_{n-1}, x_2 \rangle & \cdots & \langle x_2, x_{n-1} \rangle & \langle x_{n-1}, x_{n-1} \rangle \\
\langle x_{n}, x_2 \rangle & \cdots & \langle x_{n-1}, x_{n-1} \rangle & \langle x_{n-1}, x_{n} \rangle \\
\langle x_{n}, x_2 \rangle & \cdots & \langle x_{n-1}, x_{n} \rangle & \langle x_{n}, x_{n} \rangle \\
\end{pmatrix}
\tag{13a}
$$

Thus, as $\mathbf{X}'\mathbf{X}'$ is symmetric, only the last row or column will need to be recalculated (shown in green). Removing the redundant inner product calculations reduces the computational complexity from $\mathcal{O}(mn^2)$ to $\mathcal{O}(mn)$. As this is the most time-consuming part of the method of snapshots (Kutz et al., 2014), a large performance gain is realized. This streaming method of snapshots facilitates a streaming version of the DMD.

#### 3.2 Streaming DMD

The streaming DMD relies on the streaming SVD in order to process data in sequence, but is also able to realize speed-ups from reusing intermediate steps from the SVD, and by only returning the last column of the sparse matrix $\mathbf{X}s$ in the case of background subtraction. Figure 1 shows an outline of how the streaming DMD is set up in order to perform background subtraction.
3.3 Efficient Mode Amplitudes

Computing the vector $\mathbf{b}$ of DMD mode amplitudes has been investigated in the past (Jovanović et al., 2014; Kutz et al., 2016a). The simplest approach involves computing a best-fit $\mathbf{b}$ vector using the least-squares approximation:

$$\mathbf{b} = \Phi^\dagger x_1.$$  \hfill (14)

Instead, we use the following formulation directly on POD coefficients using Equation 8 and Equation 10:

$$x_1 = \Phi \mathbf{b}$$ \hfill (15a)

$$\Rightarrow U \alpha_1 = X' \Sigma^{-1} W \mathbf{b}$$ \hfill (15b)

$$\Rightarrow \alpha_1 = \tilde{A} W \mathbf{b}$$ \hfill (15c)

$$\Rightarrow \mathbf{b} = (W \Lambda)^{-1} \alpha_1,$$ \hfill (15e)

where $\alpha_1$ is the vector of POD coefficients for $x_1$. This is significantly more efficient than the high-dimensional least-squares algorithm. Additionally, only the row corresponding to the smallest absolute DMD eigenvalue need be calculated when streaming. The benefit of this faster calculation of the DMD mode amplitudes is even more pronounced on a GPU, requiring fewer synchronizations with the device, and reducing the amount of data transfer.

3.4 Implementation

$\mathbf{U}$ is not explicitly calculated so as to reduce space and computational complexity of the DMD. While it is possible for this to cause issues with numerical accuracy, we show that the results of these algorithms are similar in subsection 4.2. Additionally, we used single-precision to further reduce memory usage and increase performance. NVIDIA (2017) discusses how CUDA may diverge from typical IEEE 754 floating point. Floating point itself is not perfectly accurate (Goldberg, 1991), which will cause further rounding issues for large matrix operations. Our code relies on OpenBLAS (Wang et al., 2013) for LAPACK Anderson et al., 1999 and BLAS functions on the CPU, and MAGMA (Tomov et al., 2010a) for the GPU. We also found that writing algorithms in MAGMA improved performance over those written in OpenCL. This can be attributed to the amount of work put into tuning MAGMA (Tomov et al., 2010b; Dongarra et al., 2014) and the better performance of CUDA over OpenCL (Karimi et al., 2010; Fang et al., 2011; Du et al., 2012).

3.5 Sparse Data

After parallelization on the GPU, data transfers between the CPU and GPU become a bottleneck. We may naively transfer data in the ambient signal space, such as pixel space for images or a spatial domain for high performance computations. However, in both cases, these signals are typically stored or computed in a transformed basis, such as Fourier or wavelets. Moreover, these transform bases allow the data to be massively compressed, often by orders of magnitude, which would lead to a significant savings in data transfer. Recent work combining compressed sensing and DMD (Kutz et al., 2016c) showed that both the SVD and DMD are invariant to unitary transformation, such as the fast Fourier transform (FFT). Thus, it is possible to directly transfer FFT compressed data to the GPU, perform DMD on the Fourier representation, and transfer the compressed DMD from the GPU back for storage or further processing. There is an added benefit that many of the core steps in the DMD algorithm will be performed on sparse data matrices, enabling further efficiency gains. This procedure is shown schematically in Figure 5. This is not explicitly implemented in our code, but is included because of the potentially important role in reducing the size of memory transfers between CPU and GPU in practical implementations. Note that compressed and randomized (Halko et al., 2011) architectures have recently been used to great advantage in scientific computing applications, for example in (Schaeffer et al., 2013; Mackey et al., 2014).

4 Results

We now present the performance and accuracy comparisons of our streaming SVD, DMD and background subtraction algorithms. The algorithms are demonstrated on two high-resolution video datasets; however, the streaming SVD and DMD algorithms are general to any high-dimensional data inputs.

4.1 Performance

We benchmarked all algorithms on the PEViD “Walking Day Indoor 4” video (Korshunov and Ebrahimi, 2014), converted to greyscale and resized to common 16:9 resolu-
Examining (c) and (d) show that the results are nearly expected, since that the cost to update \( \lambda \) to 1 is used. Timings are a best of 5 mean with memory transfers to and from the GPU excluded. Tests were run on PEViD “Walking Day Indoor 4” [Korshunov and Ebrahimi, 2014].

Figure 6 shows comparisons of the CPU, GPU, streaming CPU (SCPU), GPU and streaming GPU (SGPU) versions of the SVD, DMD and DMD background subtraction. Times represent a one-frame update from steady state with all singular values in use. In the case of background subtraction, only the closest single or complex conjugate pair of \( \lambda \) to 1 is used. Timings are a best of 5 mean with memory transfers to and from the GPU excluded. Tests were run on PEViD “Walking Day Indoor 4” [Korshunov and Ebrahimi, 2014].

ootnote{Our tests were performed on an Intel Xeon E5-2620v3 with 32GB of RAM and an NVIDIA Tesla K40, running Ubuntu 16.04.2 LTS. Our code was compiled with gcc 5.4.0 and depends on OpenBLAS 0.2.18 [Wang et al., 2013] and MAGMA 2.2. [Tomov et al., 2010a].}


4.2 Accuracy

It is important to verify that the significant speed-up of the streaming and GPU implementations do not come with unacceptable losses in accuracy. Figure 7 shows a subjective comparison between our CPU and streaming GPU implementations of DMD background subtraction. Examining (c) and (d) show that the results are nearly
indistinguishable.

Table 1 shows comparisons made between our streaming SVD and DMD output against Python implementations of the standard algorithms. In both cases, the relative error is quite small, even for the largest input sizes. The DMD comparison was made on the product of column of $\Phi$ corresponding to the smallest absolute value in $\Lambda$. However, in many applications of DMD, such as video background modeling, this constitutes an acceptable error for the considerable speed-up, as downstream processing algorithms do not require machine precision.

Table 2 lists the standard foreground mask evaluation metrics generated by the Background Models Challenge Wizard. These evaluations correspond to the same “MOG2”, “KNN” and DMD setups as in Figure 8. In general, the DMD performance is close to that of the “MOG2” and “KNN” methods. This is promising, as DMD is a relatively new method and may be further developed and improved with recent innovations [Kutz et al. 2016c,b; Dawson et al. 2016]. Moreover, the significant computational savings associated with the new streaming GPU DMD enables real-time video processing with computations left over for downstream analysis and processing.

5 Discussion and Conclusion

Matrix decompositions, such as the singular value decomposition (SVD) and dynamic mode decomposition (DMD), are cornerstones of numerical linear algebra.
Further innovations to DMD may improve performance. which have been developed and optimized over decades. promising, as it is competitive with standard algorithms, and peak signal to noise ratio (PSNR). DMD accuracy is performance is quantified by recall, precision, F-measure, and peak signal to noise ratio (PSNR). DMD accuracy is promising, as it is competitive with standard algorithms, which have been developed and optimized over decades. Further innovations to DMD may improve performance. However, these methods typically become computationally intractable for high-dimensional data, and this cost is compounded in streaming applications, where a new matrix decomposition is required for each new measurement in time. These computational issues hinder efforts for real-time processing of high-dimensional data, such as HD video, which will only get worse with growing big data volumes. In this work, we have exploited the fact that these streaming architectures have many redundant computations and may be readily parallelized on a graphics processing unit (GPU), providing significant acceleration of the algorithm. We have developed and analyzed streaming singular value and dynamic mode decomposition algorithms and their GPU implementations. In addition, we show performance benefits for streaming video background subtraction. In all cases, a large number of calculations are able to be carried forward from frame to frame by exploiting the structure of the method of snapshots SVD. This allows both the SVD and DMD to process large data streams in real-time, whether for video or otherwise. We have evaluated the proposed algorithms on multiple datasets, demonstrating the significantly improved computational performance for stream processing with negligible loss in accuracy. Our C++ and CUDA implementation of the SVD and DMD are available under an open-source license on GitHub.

The results of our performance comparison suggest that streaming algorithms are favorable, regardless of whether a GPU is available on a target platform. Additionally, significant speed-ups are possible at smaller data sizes once faster transfers are available to and from a GPU. While not suitable for extreme-precision applications, we believe our streaming SVD and DMD algorithms provide a valuable improvement for many applications due to their improved computational performance. The small loss in accuracy was shown to be negligible for video background modeling applications.

There are a number of interesting future directions that may arise from this work. One could modify the streaming algorithms shown here to support dynamic updating with more than one column at a time; when data inputs slow down, the number of new columns processed may be increased to catch up, and vice versa. This dynamic streaming update could help to recover from a build-up of columns waiting to be processed for a long-running instance of the streaming SVD or DMD. A streaming input build-up could also be used instead of waiting for enough initial inputs for the first SVD or DMD. This would instead pre-allocate the maximum matrix size, but start the algorithm with only 2 columns. Until the matrix is filled, the new columns would be appended without erasing the oldest. Dynamically changing the number of singular values used in the economy SVD is also possible. In using the streaming DMD for background subtraction, the algorithm could be modified to use some small subset of background DMD modes rather than just the single slowest changing mode, as suggested in Ericson et al.

### Table 2: Comparison of DMD, Gaussian mixture model (MOG2), and k-nearest neighbors (KNN) on Background Models Challenge data sets [Vacavant et al. 2013].

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Alg</th>
<th>Recall</th>
<th>Precision</th>
<th>F-Measure</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video 1</td>
<td>DMD</td>
<td>0.588224</td>
<td>0.585362</td>
<td>0.584296</td>
<td>38.4607</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.618224</td>
<td>0.563368</td>
<td>0.585923</td>
<td>34.3749</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.608381</td>
<td>0.559737</td>
<td>0.583046</td>
<td>34.3468</td>
</tr>
<tr>
<td>Video 2</td>
<td>DMD</td>
<td>0.574118</td>
<td>0.524646</td>
<td>0.661912</td>
<td>26.0115</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.72366</td>
<td>0.686597</td>
<td>0.704641</td>
<td>22.42</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.792748</td>
<td>0.669372</td>
<td>0.725855</td>
<td>20.6406</td>
</tr>
<tr>
<td>Video 3</td>
<td>DMD</td>
<td>0.758537</td>
<td>0.696991</td>
<td>0.725974</td>
<td>36.2454</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.850997</td>
<td>0.687585</td>
<td>0.753383</td>
<td>33.246</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.841266</td>
<td>0.758035</td>
<td>0.751985</td>
<td>38.0807</td>
</tr>
<tr>
<td>Video 4</td>
<td>DMD</td>
<td>0.70198</td>
<td>0.700967</td>
<td>0.701473</td>
<td>34.3557</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.799231</td>
<td>0.795393</td>
<td>0.791882</td>
<td>31.8003</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.798056</td>
<td>0.658809</td>
<td>0.703935</td>
<td>36.9051</td>
</tr>
<tr>
<td>Video 5</td>
<td>DMD</td>
<td>0.626271</td>
<td>0.551574</td>
<td>0.586554</td>
<td>39.1576</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.654289</td>
<td>0.507281</td>
<td>0.571482</td>
<td>17.4064</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.651026</td>
<td>0.510855</td>
<td>0.572486</td>
<td>22.2382</td>
</tr>
<tr>
<td>Video 6</td>
<td>DMD</td>
<td>0.699749</td>
<td>0.729335</td>
<td>0.714092</td>
<td>32.3418</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.82021</td>
<td>0.659463</td>
<td>0.731105</td>
<td>26.2544</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.825746</td>
<td>0.665474</td>
<td>0.706445</td>
<td>26.5406</td>
</tr>
<tr>
<td>Video 7</td>
<td>DMD</td>
<td>0.500475</td>
<td>0.594811</td>
<td>0.600838</td>
<td>22.5322</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.750474</td>
<td>0.667586</td>
<td>0.706608</td>
<td>22.9777</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.808194</td>
<td>0.67034</td>
<td>0.732841</td>
<td>22.1621</td>
</tr>
<tr>
<td>Video 8</td>
<td>DMD</td>
<td>0.563182</td>
<td>0.595393</td>
<td>0.593293</td>
<td>22.9505</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.579596</td>
<td>0.510637</td>
<td>0.545972</td>
<td>19.8442</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.55195</td>
<td>0.507792</td>
<td>0.528951</td>
<td>16.5084</td>
</tr>
<tr>
<td>Video 9</td>
<td>DMD</td>
<td>0.505496</td>
<td>0.551778</td>
<td>0.527624</td>
<td>35.3733</td>
</tr>
<tr>
<td></td>
<td>MOG2</td>
<td>0.616528</td>
<td>0.570914</td>
<td>0.592845</td>
<td>45.5653</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.584392</td>
<td>0.575333</td>
<td>0.580303</td>
<td>45.6861</td>
</tr>
</tbody>
</table>
This would improve results at the cost of performance. Our algorithm should also be able to be used to accelerate the segmentation of Tirunagari et al.\textsuperscript{2016}’s color DMD for background subtraction. Lastly, our method could be joined with other modified DMD algorithms, such as compressed DMD (Kutz et al.\textsuperscript{2016}; Ericsson et al.\textsuperscript{2016}), multi-resolution DMD (Kutz et al.\textsuperscript{2016}), or de-noised DMD (Dawson et al.\textsuperscript{2016}) in order to improve performance.

The emergence of the big data era across the physical, biological, social and engineering sciences has severely challenged our ability to extract meaningful features from data in a real-time manner. Critical technologies such as LIDAR, 4K video streams, computer vision, high-fidelity numerical simulations, sensor networks, brain-machine interfaces, internet of things, and augmented reality will all depend on scalable algorithms that can produce meaningful decompositions of data in real time. Failure to compute data streams in real time results in a data mort-gage (Polagye et al.\textsuperscript{2014}) whereby the cost of collection and storage limits the available resources to analyze and extract features. We are already seeing this across the sciences where massive data-sets are collected and stored, yet remained un-mined for informative features and/or critical information for automated decision making processes. The streaming technique presented here provides a mathematical architecture for real-time processing of data and extraction of features. The method is adaptive, efficient, parallelizable and scalable, potentially enabling a host of applications currently beyond the capabilities of standard techniques.

Acknowledgements

We gratefully acknowledge valuable discussions with Zhe Bai and Ben Erichson. SLB and SDP acknowledge support from the Department of Energy: The information, data, or work presented herein was funded in part by the Office of Energy Efficiency and Renewable Energy (EERE), U.S. Department of Energy, under Award Number DE-EE006785. We also thank NVIDIA for providing a Tesla K40 GPU for this research. SLB also acknowledges support from the Boeing corporation [contract SSOW-BRT-W0714-0004], and SDP acknowledges support from the Mary Gates Research Scholarship.

Disclaimer

The information, data, or work presented herein was funded in part by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References


