GPU Accelerated Probabilistic Latent Sequential Motifs for Activity Analysis

Khaja Wasif Mohiuddin[1], Jagannadan Varadarajan[1], Rémi Emonet[3], Jean-Marc Odobez[4] and Pierre Moulin[1,2]

[1]*Advanced Digital Sciences Center, Singapore*

[2]*Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, IL, U.S.A.*

[3]*Jean Monnet University, Saint Étienne, France*

[4]*Idiap Research Institute, Martigny, Switzerland*

Keywords: PLSA, PLSM, Activity Analysis, Topic Models, GPU, CUDA, Motifs.

Abstract: In this paper, we present an optimized GPU based implementation of Probabilistic Latent Sequential motifs (PLSM) that was proposed for sequential pattern mining from video sequences. PLSM mines for recurrent sequential patterns from documents given as word-time occurrences, and outputs a set of sequential activity motifs and their starting occurrences. PLSM's uniqueness comes from modeling the co-occurrence and temporal order in which the words occur within a temporal window while also dealing with activities which occur concurrently in the video. However, the expectation-maximization algorithm used in PLSM has a very high time complexity due to complex nested loops, requiring several dimensionality reduction steps before invoking PLSM. In order to truly realize the benefits of the model, we propose two GPU based implementations of PLSM called GPU-pLSM (sparse and dense). The two implementations differ based on whether the entire word-count matrix (dense) or only the non-zero entries (sparse) are considered in inferring the latent motifs respectively. Our implementation achieves an impressive 265X and 366X times speed up for dense and sparse approaches respectively on NVIDIA GeForce GTX Titan. This speed up enables us to remove several pre-processing and dimension reduction steps used to generate the input temporal documents and thus apply PLSM directly on the input documents. We validate our results through qualitative comparisons of the inferred motifs on two different publicly available datasets. Quantitative comparison on document reconstruction based abnormality measure show that both GPU-PLSM and PLSA+PLSM are strongly correlated.

1 INTRODUCTION

We are entering an era of pervasive computing. More and more private and public settings are equipped with sensors such as proximity infrared sensors, RFIDs, and CCTV cameras, generating tones of data everyday. It is therefore, vital to create intelligent machines that can mimic human abilities; machines that can observe colossal amounts of data and churn out information with semantic significance and human interpretability. Such information is useful in applications such as surveillance, health care, infrastructure-planning and human behaviour analysis. However, the enormity of the generated data make even simple learning algorithms several hours or even days to run.

Recently, the general purpose graphic processing units (GPU) have become a powerful parallel computing platform, not only because of GPU's multi-core structure and high memory bandwidth, but also because of the popularity of parallel programming frameworks such as CUDA that enable developers to easily manipulate GPU's computing power. This motivates us to revisit and improvise conventional machine learning algorithms so that they can be used on large-scale datasets.

Specifically, we consider the task of mining recurrent sequential patterns (called "motifs") from large scale videos collected from public spaces such as airports, metro stations and shopping malls. Mining for such patterns can be useful both in offline tasks such as video summarization and understanding as well as online tasks such as anomaly detection, where delays in detection can cost dearly.

In this paper, we present accelerated implementations of **P**robabilstic **L**atent **S**equential **M**otifs (PLSM) (Varadarajan et al., 2010), a popular ap-

Mohiuddin, K., Varadarajan, J., Emonet, R., Odobez, J-M. and Moulin, P.
GPU Accelerated Probabilistic Latent Sequential Motifs for Activity Analysis.
DOI: 10.5220/0006537904090418
In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2018) - Volume 5: VISAPP*, pages 409-418
ISBN: 978-989-758-290-5

proach to discover sequential patterns from spatio-temporal data. PLSM is topic model based approach to activity mining in videos similar to probabilistic latent semantic analysis (PLSA) (Hofmann, 2001) and Latent Dirichlet Allocation (LDA) (Blei et al., 2003). However, PLSM addresses the disadvantages of the bag-of-words assumption in PLSA and performs temporal modeling at multiple levels: a) within motifs to identify when words occur, i.e., at which relative time with respect to the motif beginning; b) within video segments (temporal documents), to identify when a motif actually starts in the document (more details in sec 3.1). There are several advantages of temporal modeling in PLSM: a) PLSM helps in understanding how an activity unfolds over time enabling a time sensitive visualization of the discovered activity patterns; and b) it enables to precisely identify when an activity begins in a video, which could be used for tasks including event counting. Furthermore, PLSM relies on elegant generative model approach combined with well established inference techniques to uncover the latent variables. This allows an intuitive semantic interpretation of the observed and latent variables, making it an easy choice despite a few recent deep learning based approaches towards activity analysis presented in (Xu et al., 2015; Hasan et al., 2016).

Earlier PLSM implementations (Varadarajan et al., 2010) make use of complex dimensionality reduction steps using LDA (Blei et al., 2003), PLSA (Hofmann, 2001) to bring down the vocabulary size and thereby the running time of PLSM, but this is also cumbersome and time consuming. For instance, it takes nearly 4.5 hours to apply PLSA on a 90 minute long video. While this reduces the running time of PLSM, it is still inefficient due to the time spent in other pre-processing steps. Furthermore, the additional pre-processing layers also introduce difficulties in motif visualization and in higher level tasks such as abnormal event detection. Using multiple pre-processing steps makes it difficult to reason out which low-level feature caused an anomalous event. On the other had, applying PLSM directly on videos is complex and time taking due to high dimensional nature of videos combined with complex nested loops in PLSM EM procedure. However, thanks to the cheap availability of GPUs these days, it is easier to realize PLSM directly on the low-level visual features, while still achieving superior running time performance.

In this paper, we propose two different GPU based implementations of PLSM i) Dense GPU-PLSM, ii) Sparse GPU-PLSM. We perform the entire evaluation on GPU in an efficient manner minimizing the data transfers and providing good performance with high scalability. In order to ensure that our implementation is scalable, we ran exhaustive set of experiments using different generations of GPUs with increasing number of cores and memory, while varying the input dimensionality. We achieve peak performance of nearly 265X using dense approach and 366X using sparse approach.

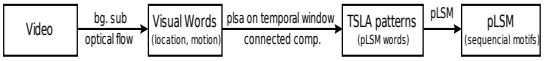


Figure 1: Flowchart for discovering sequential activity motifs from videos using PLSM, as presented in (Varadarajan et al., 2010).

2 RELATED WORK

Motion and appearance features have been used for video based activity analysis for several years. For instance, several methods have been proposed to (Xiang and Gong, 2008; Li et al., 2008; Wang et al., 2009) to fetch semantic activity patterns using low level features.

Recently, topic models like pLSA (Hofmann, 2001) LDA (Blei et al., 2003) originally proposed for text processing have been successfully used with simple image features to discover scene level activity patterns and detect abnormal events (Varadarajan and Odobez, 2009; Li et al., 2008; Wang et al., 2009). These Bag of Words methods assume that words are exchangeable and their co-occurrence is sufficient to capture latent patterns in the data. Using topic models like pLSA allows the use of different abnormality measures based on the interpretation of the model (Varadarajan and Odobez, 2009; Emonet et al., 2011). Generative topic models for large set of documents with large vocabulary size tend to consume too much computation time. There have been efforts to speed up probabilistic models like PLSA. For instance, Hong et.al (Hong et al., 2008) proposed a CPU-based parallel algorithm for PLSA and made 6x speedup on 8-core CPU machines. Yu et. al. applied GPU in Gibbs sampling for motif finding and achieved 10x speedup (Yu and Xu, 2009). Yan et. al. proposed a parallel inference method for Latent Dirichlet Allocation (LDA) on GPU and achieved 20x speedup (Yan et al., 2009). However, there has been no such efficient implementations for topic models that are popular for video based activity analysis. Therefore, in this paper, we consider the PLSM model that can be applied on video data and propose two different GPU implementations.

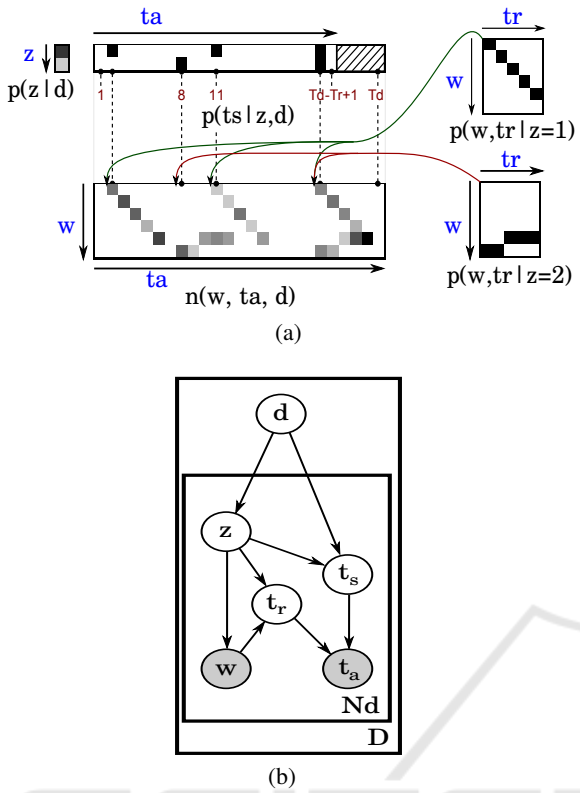


Figure 2: Generative process as presented in (Varadarajan et al., 2010) (a) Illustration of the document $n(w,t_a,d)$ generation. Words $(w,t_a = t_s + t_r)$ are obtained by first sampling the topics and their starting times from the $P(z|d)$ and $P(t_s|z,d)$ distributions and then sampling the word and its temporal occurrence within the topic from $P(w,t_r|z)$. (b) Graphical model.

3 PLSM - PROBABILISTIC LATENT SEQUENTIAL MOTIF MODEL

In this section, we first introduce the notations and provide an overview of the model, and then describe with more details the generative process and the EM steps derived to infer the parameters of the model. PLSM describes the starting times of motifs within a document as well as the temporal order in which words occur within a motif.

Figure 2a showcases generation of documents. Let D be the number of documents[1] d in the corpus, each spanning T_d discrete time steps. Let V = $\{W_i\}_{i=1}^{N_w}$ be the vocabulary of words that can occur at any given instant t_a = 1,..T_d. A document is then described by its count matrix n(w, t_a,d) indicating the number of times a word w occurs at the absolute time ta within the document. These documents are generated from a set of N_z topics $\{Z_i\}_{i=1}^{N_z}$ assumed to be temporal patterns P(w,t_r |z) with a fixed maximal duration of T_z time steps (i.e. $0 \leq$ tr$<$ T_z), where t_r denotes the relative time at which a word occurs within a topic, and that can start at any time instant t_s within the document. In other words, qualitatively, documents are generated in a probabilistic way by taking the topic patterns and reproducing them at their starting positions within the document, as illustrated in Figure 2a.

Figure 2a illustrates how documents are generated in our approach. Let D be the number of documents d in the corpus, each having N_d words and spanning T_d discrete time steps . Let $V = \{w_i\}_{i=1}^{N_w}$ be the vocabulary of words that can occur at any given instant $t_a = 1,..T_d$. A document is then described by its count matrix $n(w,t_a,d)$ indicating the number of times a word w occurs at the absolute time t_a within the document. These documents are generated from a set of N_z topics $\{z_i\}_{i=1}^{N_z}$ assumed to be temporal patterns $P(w,t_r|z)$ with a fixed maximal duration of T_z time steps (i.e. $0 \leq t_r < T_z$), where t_r denotes the relative time at which a word occurs within a topic, and that can start at any time instant t_s within the document.

In other words, qualitatively, documents are generated in a probabilistic way by taking the topic patterns and reproducing them at their starting positions within the document, as illustrated in Fig.2a.

3.1 Generative Process

The actual process to generate all triplets (w,t_a,d) which are counted in the frequency matrix $n(w,t_a,d)$ is given by the graphical model depicted in Figure 2b and works as follows:

- draw a document d with probability $P(d)$;
- draw a latent topic $z \sim P(z|d)$, where $P(z|d)$ denotes the probability that a word in document d originates from topic z;
- draw the starting time $t_s \sim P(t_s|z,d)$, where $P(t_s|z,d)$ denotes the probability that the topic z starts at time t_s within the document d;
- draw a word $w \sim P(w|z)$, where $P(w|z)$ denotes the probability that a particular word w occurs within the topic z;
- draw the relative time $t_r \sim P(t_r|w,z)$, where $P(t_r|w,z)$ denotes the probability that the word w within the topic z occurs at time t_r;
- set $t_a = t_s + t_r$, which assumes that $P(t_a|t_s,t_r) = \delta(t_a - (t_s + t_r))$, that is, the probability density function $P(t_a|t_s,t_r)$ is a Dirac

[1] We use the terms topic and motifs interchangeably. Similarly, we use the term document to refer to a video clip.

function. Alternatively, we could have modeled $P(t_a|t_s,t_r)$ as a noise process specifying uncertainty on the time occurrence of the word.

The joint distribution of all variables can be derived from the graphical model. However, given the deterministic relation between the three time variables ($t_a = t_s + t_r$), only two of them are actually needed to specify this distribution (for instance, we have $P(w,t_a,d,z,t_s,t_r) = P(t_r|w,t_a,d,z,t_s)P(w,t_a,d,z,t_s) = P(w,t_a,d,z,t_s)$ if $t_a = t_s + t_r$, and 0 otherwise). In the following, we will mainly use t_s and t_a for convenience. In practice, we allow the motifs to start anytime between 1 to T_{ds} time steps, where $T_{ds} = T_d - T_z + 1$. Accordingly, the joint distribution is given by:

$$P(w,t_a,d,z,t_s) = P(d)P(z|d)P(t_s|z,d) \\ P(w|z)P(t_a - t_s|w,z) \quad (1)$$

Our final goal is to discover the topics and their starting times given the set of documents $\mathrm{n}(w,t_a,d)$. This is a difficult task since the topic occurrences in the documents overlap temporally, as illustrated in Figure2a. The estimation of the model parameters Θ can be done by maximizing the log-likelihood of the observed data $\mathcal{D}$, which is obtained through marginalization over the hidden variables $Y = \{t_s, z\}$ (since $t_r = t_a - t_s$, see discussion above):

$$\mathcal{L}(\mathcal{D}|\Theta) = \sum_{d=1}^{D}\sum_{w=1}^{N_w}\sum_{t_a=1}^{T_d} n(w,t_a,d) \\ \log\sum_{z=1}^{N_z}\sum_{t_s=1}^{T_{ds}} P(w,t_a,d,z,t_s) \quad (2)$$

The above equation can not be solved directly due to the summation terms inside the log. Thus, we employ an Expectation-Maximization (EM) approach and maximize the expectation of the complete log-likelihood instead, which is given by:

$$E[\mathcal{L}] = \sum_{d=1}^{D}\sum_{w=1}^{N_w}\sum_{t_a=1}^{T_d}\sum_{z=1}^{N_z}\sum_{t_s=1}^{T_{ds}} \mathrm{n}(w,t_a,d) \\ P(z,t_s|w,t_a,d)\log P(w,t_a,d,z,t_s) \quad (3)$$

with

$$P(z,t_s|w,t_a,d) = \frac{P(w,t_a d,z,t_s)}{P(w,t_a,d)} \quad (4)$$

and

$$P(w,t_a,d) = \sum_{z=1}^{N_z}\sum_{t_s=1}^{T_{ds}} P(w,t_a,d,z,t_s) \quad (5)$$

In the E-step, the posterior distribution of hidden variables is then calculated as: where the joint probability is given by Eq. 1. Then, in the M-step, the model parameters (the probability tables) are updated according to (using the most convenient time variables, see end of Section 3.1):

$$P(z|d) \propto \sum_{t_s=1}^{T_{ds}}\sum_{t_r=0}^{T_z-1}\sum_{w=1}^{N_w} \mathrm{n}(w,t_s+t_r,d)P(z,t_s|w,t_s+t_r,d) \quad (6)$$

$$P(t_s|z,d) \propto \sum_{w=1}^{N_w}\sum_{t_r=0}^{T_z-1} \mathrm{n}(w,t_s+t_r,d)P(z,t_s|w,t_s+t_r,d) \quad (7)$$

$$P_w(w|z) \propto \sum_{d=1}^{D}\sum_{t_s=1}^{T_{ds}}\sum_{t_r=0}^{T_z-1} \mathrm{n}(w,t_s+t_r,d)P(z,t_s|w,t_s+t_r,d) \quad (8)$$

$$P_{t_r}(t_r|w,z) \propto \sum_{d=1}^{D}\sum_{t_s=1}^{T_{ds}} \mathrm{n}(w,t_s+t_r,d)P(z,t_s|w,t_s+t_r,d) \quad (9)$$

In practice, the EM algorithm is initialized using random values for the model parameters and stopped when the data log-likelihood increase is too small. A closer look at the above equations shows that qualitatively, in the E-step, the responsibilities of the topic occurrences in explaining the word pairs (w,t_a) are computed (where high responsibilities will be obtained for informative words, i.e. words appearing in only one topic and at a specific time), whereas the M-steps aggregates these responsibilities to infer the topic occurrences and the topic patterns. It is important to notice that thanks to the E-steps, the multiple occurrences of an activity in documents are implicitly aligned in order to learn its pattern.

Once the topics are learned, their time occurrences in any new document (represented by $P(z|d_{new})$ and $P(t_s|z,d_{new})$) can be inferred using the same EM algorithm, but using only Eq. 6 and Eq. 7 in the M-step.

The flowchart in Figure 1 shows how PLSM is applied on real-life videos. In order to apply the PLSM model on videos, we need to define the words w forming its vocabulary. One possibility would be to define some quantized low-level motion features and use these as our words. However, due to the complexity of PLSM inference (cf.1, 3), typically a dimensionality reduction step relying on PLSA is introduced. The topics from PLSA are then directly used as words PLSM, while the word counts are obtained by measuring the amount of each PLSA topic present in the temporal window.

4 GPU PLSM

PLSM involves computation of likelihood and probabilities of a topic occurring in each iteration, which involves several iterations of computationally intensive steps looping over the number of documents (D), number of topics (N_z), vocabulary size (N_w), starting time (T_{ds}). The overall complexity per iteration may be given by $O(N_w * N_z * D * T_{ds} * T_d)$. Every iteration is dependant on previous iterations' results, eventually leading to the model parameters. The algorithm terminates when convergence is attained or (and) maximum number of iterations (N_{itr}) is reached.

To ensure coalesced access of data, document array, initialised arrays layout were designed to facilitate inner loops and effectively use shared memory. We implement the CUDA-accelerated GPU-PLSM algorithm, which can be divided into three distinct stages of operation. We stored relevant variables on device memory to minimise data transfers and avoid any duplicate evaluations. We have taken two approaches to solve this problem depending on the number of non-zero word count in the input document.

4.1 Dense GPU-PLSM

4.1.1 Stage 1 P(w, t_a, d, z, t_s)

Computing the joint distribution P(w,t_a,d,z,t_s) is a key step in *PLSM* for Expectation-Maximization (EM) to eventually compute the complete log-likelihood. This evaluation comprises of nested looping along D, N_z, N_w, T_{ds} and T_z. For every GPU kernel a grid, block size is decided before processing the data. A grid is a collection of 2D/3D blocks which in turn is further divided into 2D/3D set of threads which belong to a particular block. Based on the GPU architecture we have mapped N_w, D onto Grid(x,y) respectively and block threads would be mapped to T_{ds}. We provide each thread sufficient work to loop over ranged parameters T_z, N_z processing them sequentially. All words generated by a topic starting at time t_s occur within a document; hence t_s takes values between 1 and T_{ds}, where T_{ds} = T_d - $T_z + 1$. However, we can also assume that topics are partially observed beginning or end are missing the frequency matrix. We had to be careful to avoid any race conditions. It can be seen that multiple pairs of (t_s,t_r) would write to a single t_d(t_s+t_r-1). In order to avoid concurrent write we fixed t_r in every block and exploited parallelism over t_s in batches. We effectively used the shared memory feature of Pascal architecture by loading common accessed variables by the block threads to reduce the global clock cycles. We were able to achieve occupancy of 100% effectively using shared memory to maximise the throughput.

Algorithm 1: Cuda Kernel for P(w,t_a,d,z,t_s).

$B_{idx} \leftarrow$ Number of Documents (d) on BlockId x
$B_{idy} \leftarrow$ CorpusLength (w) on BlockId y
$T_{idx} \leftarrow$ Timestamp (T_{ds}) on ThreadId x
$t_r \leftarrow$ Topic window (T_r)
$z \leftarrow$ number of Topics (N_z)
$S_{Pd} \leftarrow P_d[B_{idx}]$
for t_r<T_z **do**
    $S_{Pwtad} = 0$
    for z<N_z **do**
        $S_{Pzd} \leftarrow P_{zd}[z, B_{idx}]$
        $S_{Ptszd} \leftarrow P_{tszd}[T_{idx}, z, B_{idx}]$
        $S_{Pwz} \leftarrow P_{wz}[B_{idy}, z]$
        $S_{Ptrwz} \leftarrow P_{trwz}[t_r, B_{idy}, z]$
        SyncThreads()
        $S_{Pwtad} += S_{Pd} * S_{Pzd} * S_{Ptszd} * S_{Pwz} * S_{Ptrwz}$
    SyncThreads()
    $P_{wtad} += S_{Pwtad}$

4.1.2 Stage 2: $P(t_s|z,d_{new})$, $P(z|d_{new})$

This is the M-step, where topics are learned and their time of occurrence are inferred. This step is computed by looping over N_w, T_z, N_w , D and it is additionally looped over T_{ds} to compute $P(z|d_{new})$. N_z and D are mapped on the Grid(x,y). N_w and T_z are looped in chunks of 16,16 along thread dimension x,y. Global arrays $P_d, P_{tszd}, P_{zd}, P_{wz}, P_{trwz}$ are stored partially on device's shared memory $S_{Pd}, S_{Ptszd}, S_{Pzd}, S_{Pwz}, S_{Ptrwz}$. We are able to achieve occupancy of 75% using 40 registers.

4.1.3 Stage 3 $P(t_r|w,z)$, $P(w|z)$

This computation is done only during training by looping over T_z, D, T_d, N_w, N_z. The approach is similar to that of $P(t_s|z,d)$ kernel. N_w and N_z are mapped onto Grid(x,y). T_{ds} is mapped on ThreadIdx. Sequential looping is done over $t_r$and D. $P(w|z)$ is then computed by summing over t_r loop. A similar approach is taken as used for computing P_{tszd}. We were able to achieve occupancy of 75% using 40 registers.

4.2 Sparse GPU-PLSM

When a document $n(w,t_a,d)$ is generated there are a number of words whose frequency count is 0 in the set of given documents. Only non-zero indices contribute towards the computation and can be identified while reading the term document. The idea is to process only these non zero indices and skip the rest of the

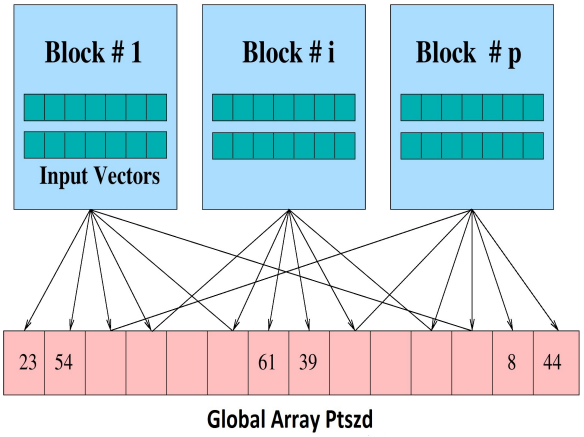


Figure 3: Layout of the CUDA blocks. Each block (128) evaluates the document word count(w,t_a,d) contribution to $P(t_s|z,d)$ for possible values of T_s,z.

Algorithm 2: Cuda Kernel for $P(t_s|z,d_{new})$ and $P(z|d_{new})$.

$B_{idx} \leftarrow$ number of Topics (N_z) on BlockId x
$B_{idy} \leftarrow$ Number of Documents (D) on BlockId y
$T_{idx} \leftarrow$ CorpusLength (N_w) on ThreadId x
$T_{idy} \leftarrow$ Topic Window (T_z) on ThreadId y
$t_s \leftarrow$ Document Time window (T_{ds})
$B_X \leftarrow$ Block Width
$B_Y \leftarrow$ Block Height
$S_{Pd} \leftarrow P_d[B_{idx}]$
for t_s<T_{ds} **do**
  $S_{Ptszd} = 0$
  $S_{Pzd} \leftarrow P_{zd}[B_{idx},B_{idy}]$
  $S_{Ptszd} \leftarrow P_{tszd}[t_s,B_{idx},B_{idy}]$
  $S_{Pwz} \leftarrow P_{wz}[T_{idx},B_{idx}]$
  $S_{Ptrwz} \leftarrow P_{trwz}[T_{idy},T_{idx},B_{idx}]$
  $w_id \leftarrow [T_{idx},t_s+T_{idy},B_{idy}]$
  SyncThreads()
  $S_{Ptszd_{new}} = S_{Pd} * n[w_{id}] * S_{Pzd} * S_{Ptszd} * S_{Pwz} * S_{Ptrwz}/(Pwtad[w_{id}]+epsilon)$
  for i=$B_x * B_y/2$, i>=1, i>>=1 **do**
    $S_{tszd_{new}}[T_{idx}] += S_{tszd_{new}}[T_{idx}+i]$
  SyncThreads()
  $P(t_s|z,d_{new}) += S_{tszd_{new}}[0]$
$P(z|d_{new}) += P_{tszd_{new}}$

evaluation. For every non zero entry it can be mapped to an entry in (w,t_a,d) tuple. For every t_a there will be multiple pairs of (t_s,t_r). We process set of non zero word count $n(w,t_a,d)$ in each CUDA block as shown in Figure 3. Global array $P(t_s|z,d_{new})$ would get multiple contributions from various blocks giving rise to concurrent writes. We made use of fast atomic operations to ensure values are updated appropriately.

We experimented by storing $n(w,t_a,d)$ contribution to $P(w,T_d,d,t_s)$ for various possibilities of T_s in a larger array and then shrink the array in a serial fashion to $P(t_s|z,d_{new})$. This proved to be costly in terms of storage. It would not scale to the increasing set of parameters. We do one time book keeping of all possible pairs that exist for every value of t_d. All these possible t_s are stored in a single array and accessed based on t_d of the word. This is significantly helpful in $P(t_s|z,d)$ and $P(t_r|w,z)$ P_{trwz} evaluation which consume major chunk of computation load. The problem comes in while updating the tuple (t_s,z,d) where in multiple words w and topic window T_r write to same global location. Partitioning all such collisions into respective bins would not be load balanced and also give rise to divergence of threads. In order to resolve concurrent write issue we used fast atomic operation. This way all such global locations which face concurrent write problem are updated sequentially avoiding any loss of data.

Algorithm 3: Sparse GPU-PLSM.

$T_{idx} \leftarrow$ Non zero index
$T_{idy} \leftarrow$ Number of Topics nZ
$Ts_{id} \leftarrow$ Possible values of Ts for Tidx
$B_X \leftarrow$ Block Width 64
$S_{Ptszd} = 0$
$S_{Pzd} \leftarrow P_{zd}[idx]$
$S_{Ptszd} \leftarrow P_{tszd}[T_{idx}]$
for t_s<Ts_{id} **do**
  $S_{Ptszd_{new}} += S_{Pd} * Doc[w_id] * S_{Pzd} * S_{Ptszd} * S_{Pwz} * S_{Ptrwz}/(Pwtad[w_id]+epsilon)$
atomicAdd($P(t_s|z,d_{new})$,$S_{Ptszd_{new}}$)
atomicAdd($P(z|d_{new})$,$S_{Ptszd_{new}}$)

5 EXPERIMENTAL RESULTS

We evaluated the performance of our GPU implementation two GPUs with varying capacity: i) NVIDIA's GTX Titan X, and ii) Quadro K620. The sequential implementation was run on Intel(R) Xeon(R) CPU 3.50GHz. NVIDIA's GTX Titan X for GPU provides 11 teraflops of FP32 performance, powered with 3072 CUDA cores. Pascal architecture enables shared memory of 49152 bytes per block and L2 cache memory of 3145728 bytes. Quadro K620 comes with 384 cores and 2GB of device memory. We initialize the CUDA hardware, allocating the appropriate host and device memory. We also took into account the available device memory to avoid memory leak. For a typical set of parameters $N_{itr} = 50$, $N_w = 75$, $D = 140$, $T_{ds} = 100$, $N_z = 25$, $T_z = 15$ it would require a memory size of 200 MB. The sequential implementation by (Varadarajan et al., 2010) has been used as

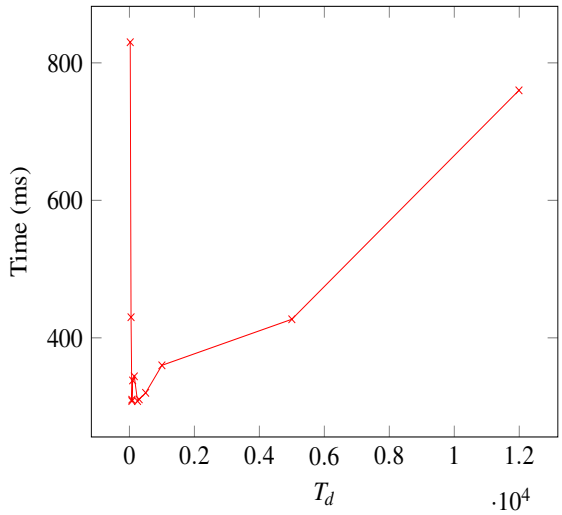


Figure 4: Run time performance of Dense-GPU implementation on GTX Titan X obtained by varying the document length T_d.

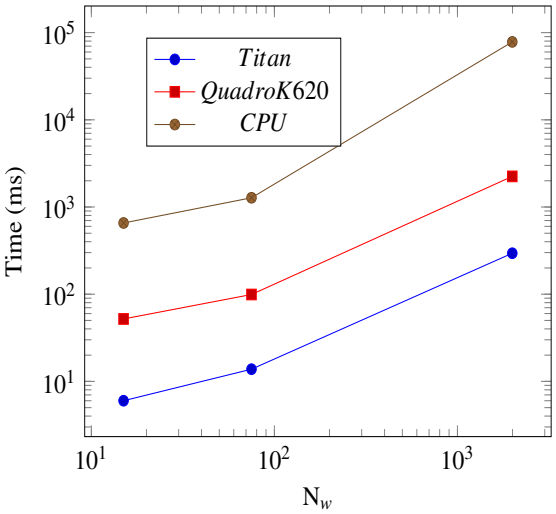


Figure 5: Performance plot for PLSM per iteration against Vocabulary size.

a benchmark to verify the accuracy of the parameters and performance. The timings given in Table 1 are average over 50 iterations of PLSM. Each iteration includes one complete EM step. We have performance timing for vocabulary size upto 15000 and document length of almost 2000. We evaluated the performance by varying the vocabulary size N_w, document length T_d and number of documents N_d. Experiments were carried out on low level features generated from actual surveillance video. We refer to (Varadarajan et al., 2010) for details on how the low-level features are obtained. In the Dense-GPU approach, we run through the complete document tuple $n(w,t_a,d)$ to perform PLSM. We were able to exploit the GPU architecture and reduce the computational complexity from $O(N_w N_z D T_{ds} T_d)$ to $O(\log(N_w) N_z T_s)$.

The comparison of performance on CPU, GPU have been done on actual video data. Table 1 shows *PLSM* timings per iteration on CPU, Quadro K620 and GTX TitanX.

Table 1: Per iteration timings (ms) for *PLSM* with T_d=100, N_z=25, T_z=15.

Parameters(w,d)	CPU	K620	Titan
15,12	673	48.7	7.6
75,5	1298.9	97.3	13.12
75,140	47154.3	2479.3	334.1
1994,5	78545.3	2247	296.2

Figure 4 shows performance of the Dense-GPU implementation obtained by varying the document length T_d. Since, the duration of the video is fixed, increasing the document length will reduce the total number of documents D. We observed the per iteration timing by varying document length from 25 to 11991 (actual document length) by fixing the number of topics to $N_z = 25$ and vocabulary size to $N_w = 75$. We found that best performance is obtained when T_d is around 75. T_s is mapped on to the block threads in P_{wtad} kernel, P_{tszd} threads internally loop over T_s. It is clear that increasing T_d would certainly increase the computation load on P_{tszd} kernel. Also since processing is done in warps, we observe an increase in the throughput whenever T_s is close to power of 2. So it would be ideal to choose a T_s value that is a power of 2 that would also give rise to adequate number of documents.

Figure 5 shows performance comparison of GPU-PLSM against the CPU PLSM for varying size of the vocabulary on GPU Titan X and Quadro K620. We observed that with increasing vocabulary size performance on Titan X saw a boost by giving a speedup of 145X. The number of cores scale well with increasing N_w. The scalability in the number of cores of the GPU can be seen on low end card like Quadro K620 with 384 cores and device memory (2 GB). So in this we have limited our vocabulary size and compared the individual performance of K620 (peak performance 863 GFLOPS) with that of TitanX (peak performance of 1TB). For low input size, the performance of K620 compared to other high end card is shown in Figure 5. GTX Titan boosted the speed on an average by a factor of 7.6 compared to Quadro K620. The significant points are that the Quadro K620 also was able to give good performance and the algorithm scales well with increasing number of cores.

Figure 6 shows comparison of sparse and dense implementation of GPU-PLSM on Titan X for various values of N_w, *i.e.,* vocabulary size. For small N_w,

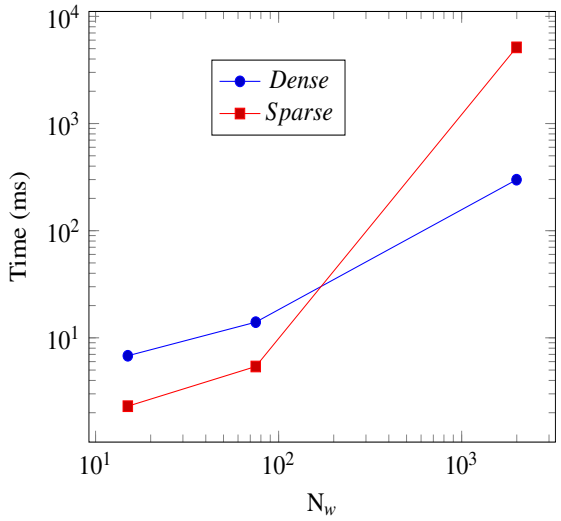


Figure 6: Performance plot for Sparse and Dense GPU-PLSM per iteration on GTX Titan.

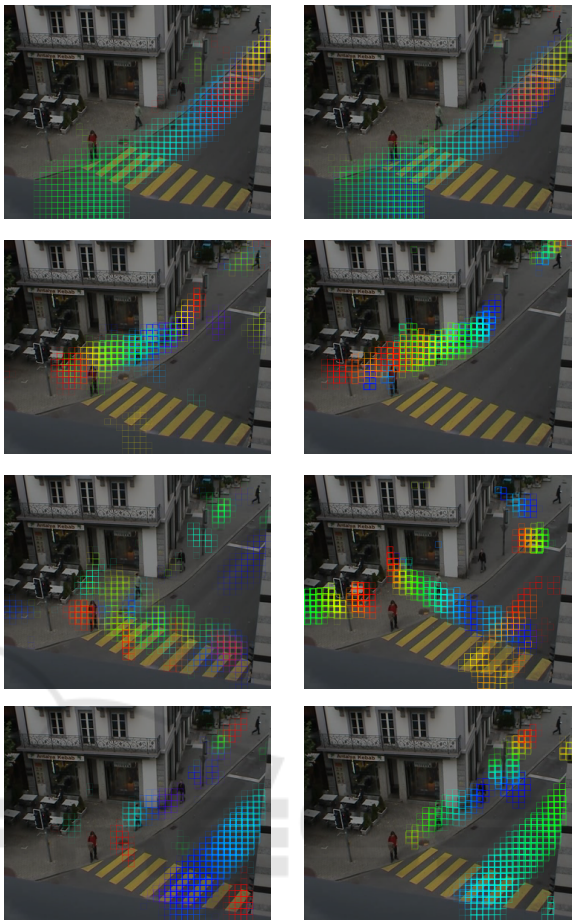
the sparse implementation does well providing 2.3 times speedup compared to the usual dense approach. But for larger values of N_w the dense approach performance better than the sparse approach. The main reason behind this counter-intuitive behaviour for large N_w is the large number of collisions in the *atomicAdd* operation while updating the global variables. However, one could chose either of the implementations based on the input parameters and number of non-zero entries in the document.

5.1 Visualization

The **Traffic Junction** video (see Figure 7) is 45 minutes long and captures a portion of a busy traffic-light-controlled road junction. Typical activities include people walking on the pavement or waiting before crossing over the zebras, and vehicles moving in and out of the scene. The data set videos have a frame size of 280×360.

The first column (Figure 7a) shows visual results using PLSA+PLSM and the second column (Figure 7b) shows results from our GPU-PLSM. The discovered patterns are superimposed on the scene image, where the colors represent the relative time from the start of the activity, i.e., violet indicates the first time step and red indicates the last time step of the activity. We can observe that results from PLSA-PLSM and GPU-PLSM are indeed quite similar indicating that there is no loss in the output of GPU-PLSM when low level features are directly fed to the model.

The **Far Field** video from (Varadarajan et al., 2010) (see Fig. 8) contains 108 minutes of a three-



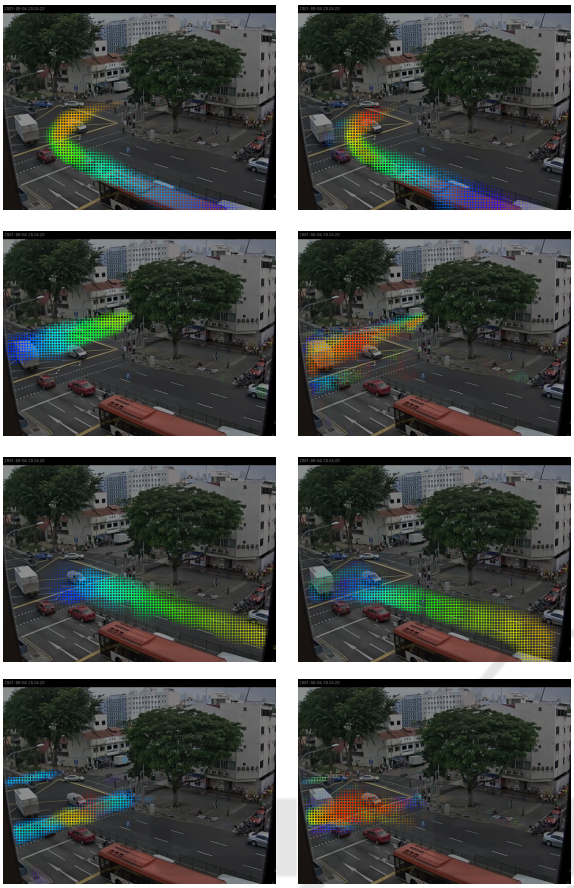
(a) (b)

Figure 7: Traffic Junction a) Sequential Motif using PLSA,PLSM b) Sequential Motif using only GPU-PLSM on low level features. Colors represent the relative time from the start of the activity, i.e., violet indicates the first time step and red indicates the last time step of the activity.

road junction captured from a distance, where typical activities are moving vehicles. As the scene is not controlled by a traffic signal, activities have large temporal variations. For this event detection task, we labelled a 108 minute video clip from the far field scene, distinct from the training set Figure 8a shows visual results obtained using PLSA+PLSM as done in (Varadarajan et al., 2010), and Figure 8b shows results obtained using GPU-PLSM. We can observe from the visualization that the results from both the implementations are comparable.

5.2 Abnormality Measure

We also compared the two approaches quantitatively, to validate our GPU based implementations. For this we used the mean absolute document reconstruction error (MADRE) proposed by ((Emonet and Odobez,



(a) (b)

Figure 8: Far Field a) Sequential Motif using PLSA+PLSM b) Sequential Motif using GPU-PLSM directly on the low level features.

2012)). More precisely, given the observed word-count matrix $n(w,t_a,d)$, and the reconstructed (reconstructed via inference) document word-count matrix $P(w,t_a|d)$, the abnormality measure is defined as:

$$MADRE(t_a,d) = \sum_w |\frac{n(w,t_a,d)}{n(d)} - P(w,t_a|d)| \quad (10)$$

$$P(w,t_a|d) = \sum_z \sum_{ts} P(w,t_a,d,z,t_s) \quad (11)$$

In order to compare the GPU-PLSM with PLSA-PLSM, we show a scatter plot of the MADRE values obtained by the two methods in Figure 9. From the scatter plot, we find that the two methods exhibit a strong correlation. In order to ensure that the detections from the two methods will be the same, they need to have a strong positive correlation. We observed that the values obtained from the two methods have a correlation coefficient of 0.7979 indicating a strong positive linear relationship between them.

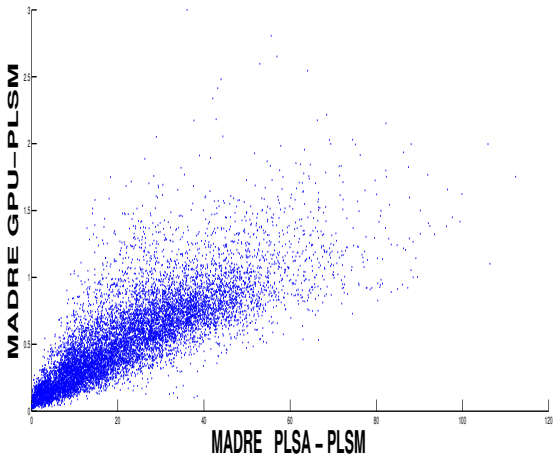


Figure 9: Scatter plot using MADRE for PLSM against PLSA+PLSM.

6 CONCLUSIONS

In this paper, we presented a GPU-PLSM approach to address the running time inefficiencies found in PLSM method used for video based activity analysis applications. To this end, we proposed two variants of the GPU-PLSM, namely, dense and sparse GPU-PLSM, based on whether the non-zero entries are used in the computation or not in the EM computation respectively. Through experiments done on two different GPU platforms, we were able to achieve a top speed up of 366X compared to its CPU counterpart. We further validated our results from GPU-PLSM using both qualitative and quantitative comparisons and showed that the results from GPU-PLSM correlate well with the vanilla PLSM implementation. We believe that our contribution will encourage real time analysis and detection of abnormal events from videos. In future work, we plan to work more on optimizing the sparse approach for large vocabulary sizes to bring down the computation time and improve memory optimization.

ACKNOWLEDGEMENTS

This study is supported by the research grant for the Human Centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center from Singapore's Agency for Science, Technology and Research (A*STAR). The authors thank NVIDIA for donating GPUs to support this research work.

REFERENCES

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Emonet, R. and Odobez, J.-M. (2012). *Intelligent Video Surveillance Systems (ISTE)*. Wiley-ISTE.

Emonet, R., Varadarajan, J., and Odobez, J. (2011). Multi-camera open space human activity discovery for anomaly detection. In *8th IEEE International Conference on Advanced Video and Signal-Based Surveillance, AVSS*, pages 218–223.

Hasan, M., Choi, J., Neumann, J., Roy-Chowdhury, A. K., and Davis, L. S. (2016). Learning temporal regularity in video sequences. *CoRR*, abs/1604.04574.

Hofmann, T. (2001). Unsupervised learning by probability latent semantic analysis. *Machine Learning*, 42:177–196.

Hong, C., Chen, W., Zheng, W., Shan, J., Chen, Y., and Zhang, Y. (2008). Parallelization and characterization of probabilistic latent semantic analysis. In *2008 37th International Conference on Parallel Processing*, pages 628–635.

Li, J., Gong, S., and Xiang, T. (2008). Global behaviour inference using probabilistic latent semantic analysis. In *British Machine Vision Conference*.

Varadarajan, J., Emonet, R., and Odobez, J. (2010). Probabilistic latent sequential motifs: Discovering temporal activity patterns in video scenes. In *British Machine Vision Conference, BMVC 2010, Aberystwyth, UK, August 31 - September 3, 2010. Proceedings*, pages 1–11.

Varadarajan, J. and Odobez, J. (2009). Topic models for scene analysis and abnormality detection. In *ICCV-12th International Workshop on Visual Surveillance*.

Wang, X., Ma, X., and Grimson, E. L. (2009). Unsupervised activity perception in crowded and complicated scenes using hierarchical bayesian models. *IEEE Trans. on PAMI*, 31(3):539–555.

Xiang, T. and Gong, S. (2008). Video behavior profiling for anomaly detection. *IEEE Trans. on PAMI*, 30(5):893–908.

Xu, D., Ricci, E., Yan, Y., Song, J., and Sebe, N. (2015). Learning deep representations of appearance and motion for anomalous event detection. In Xianghua Xie, M. W. J. and Tam, G. K. L., editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 8.1–8.12. BMVA Press.

Yan, F., Xu, N., and Qi, Y. (2009). Parallel inference for latent dirichlet allocation on graphics processing units. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 2134–2142. Curran Associates, Inc.

Yu, L. and Xu, Y. (2009). A parallel gibbs sampling algorithm for motif finding on gpu. In *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 555–558.