This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

A Fully-Pipelined Hardware Design for Gaussian Mixture Models

Conghui He, Haohuan Fu, Ce Guo, Wayne Luk and Guangwen Yang

Abstract—Gaussian Mixture Models (GMMs) are widely used in many applications such as data mining, signal processing and computer vision, for probability density modeling and soft clustering. However, the parameters of a GMM need to be estimated from data by, for example, the Expectation-Maximization algorithm for Gaussian Mixture Models (EM-GMM), which is computationally demanding. This paper presents a novel design for the EM-GMM algorithm targeting reconfigurable platforms, with five main contributions. First, a pipeline-friendly EM-GMM with diagonal covariance matrices that can easily be mapped to hardware architectures. Second, a function evaluation unit for Gaussian probability density based on fixed-point arithmetic. Third, our approach is extended to support a wide range of dimensions or/and components by fitting multiple pieces of smaller dimensions onto an FPGA chip. Fourth, we derive a cost and performance model that estimates logic resources. Fifth, our dataflow design targeting the Maxeler MPC-X2000 with a Stratix-5SGSD8 FPGA can run over 200 times faster than a 6-core Xeon E5645 processor, and over 39 times faster than a Pascal TITAN-X GPU. Our design provides a practical solution to applications for training and explores better parameters for GMMs with hundreds of millions of high dimensional input instances, for low-latency and high-performance applications.

Index Terms—Gaussian Mixture Model, Expectation Maximization, High Performance Computing, Data Flow Engine, Reconfigurable hardware, Algorithms implemented in hardware.

1 Introduction

Gaussian Mixture Models (GMMs) are widely used in many applications such as data mining, signal processing and computer vision, for probability density modeling and soft clustering. For example, Greenspan et al. propose an image segmentation system to identify issues in magnetic resonance images of the brain where GMMs are employed to capture the spatial layout information of brain tissues [1]. In Reynolds's speaker verification system used successfully in several NIST Speaker Recognition Evaluations (SREs), GMMs models the acoustic classes of a speaker's voice [2]. Stauffer et al. design a computer vision system to subtract the background from video streams and GMMs are used to judge whether a pixel belongs to the background in a probabilistic manner [3].

Before working with the GMM probability density model, which is governed by a set of parameters, parameters of the GMM density model need to be estimated in advance. Maximum Likelihood Estimation (MLE) bears much importance in the world of parameter estimation. Expectation Maximization (EM) algorithm is widely used for the MLE estimation. EM for Gaussian Mixture Models (EM-GMM), a particular adaptation of the EM algorithm is a popular estimation solution to estimate the GMM density model [4]. It estimates the parameters of a GMM in an iterative manner. In each iteration, the algorithm computes and collects statistical evidence from the data set and then updates the parameters based on the evidence.

In recent years, the EM-GMM algorithm becomes increasingly computationally demanding since the development of high-definition sensors and novel Internet technology leads to a fast growth of data size. The more efficient the EM-GMM algorithm is, the more data can be used for parameter estimation and will result in better parameters. Some applications expect the EM-GMM algorithm to be a fast response or even low-latency such as real-time object tracking [3].

This paper presents a novel design for the EM-GMM algorithm targeting reconfigurable platforms. We provide a fast parameter estimation system that handles most of the computations in the EM-GMM algorithm in a fully pipelined manner and provide a high performance and low-latency EM-GMM engine for real-world applications. Our major contributions are as follows:

- We restructure the workflow of the original EM-GMM algorithm with algorithmic transformations to enable pipelining of different computation stages, resulting in a pipeline-friendly EM-GMM algorithm.
- We propose a customized design of the Gaussian probability density evaluation unit that minimizes the hardware cost while achieving satisfactory accuracy.
- While the limited hardware resource of an FPGA chip can only fit designs with a small dimension (D) and component (K) parameter, we propose a generalized decomposition scheme that supports a wide range of D and K values.
- We propose a cost and performance model that estimates the possible necessary logic units given a set of configurations with a certain confidence.
- We map our design to two FPGA platforms, the Xilinx Virtex-6 platform and the Altera Stratix-V platform, achieving a maximum of 207 times speedup over a 6-core Xeon E5645 processor, and 39 times over a Pascal TITAN-X GPU solution.

- *C. He, H. Fu and G. Yang are from Tsinghua University, China.*
- *C. Guo and W. Luk are from Imperial College, Lodon.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

The first two contributions map the EM-GMM algorithm to FPGA effectively and efficiently, achieving more than 500x speedup over one CPU core [5]. The third contribution provides a general solution to support a wide range of dimensions or/and components by fitting multiple pieces of smaller dimensions or/and components into the limited space of an FPGA chip, which makes our design a high-performance engine for real-world applications. A cost and performance model also saves users a large amount of time on hardware compilation, and enables them to analyze the possible resource usage and performance gain more quickly.

The rest of this paper is organized as follows. Section 2 provides an introduction to GMMs and the EM-GMM algorithm, and discusses existing high performance computing solutions for the EM-GMM algorithm. Section 3 presents our hardware friendly EM-GMM framework. Section 4 describes our restructured pipeline-friendly EM-GMM algorithm. Section 5 presents our customized evaluation unit for Gaussian probability density function. Section 6 gives us an algorithm to split dimensions and components on hardware if the logic resources are not enough. Section 7 shows some experimental results on accuracy and performance of our design compared with two CPU-based systems and one GPU-based system. This section also provides us an equation to estimate the required logic resources for a given configuration. Finally, Section 8 provides a brief conclusion and discusses possible future work.

2 Background

2.1 Gaussian mixture models

A Gaussian Mixture Model (GMM) is a linear combination of multiple Gaussian distributions. A GMM with K Gaussian components can be represented in the form

$$p(x_n) = \sum_{k=1}^{K} w_k G(x_n|\mu_k, \Theta_k) \tag{1}$$

where

- $x_n = (x_{n1}, x_{n2}, \cdots, x_{nD})$ is a vector representing a data instance with D attributes. It can be considered as a point in a D-dimensional Euclidean space.
- $G(x_n|\mu_k, \Theta_k)$ is a Gaussian probability density controlled by mean vector $\mu_k = (\mu_{k1}, \mu_{k2}, \cdots, \mu_{kD})$ and covariance matrix Θ_k. The probability density function can be mathematically defined by

$$G\left(x_n|\mu_k, \Theta_k\right) = \frac{e^{-\frac{1}{2}(x_n-\mu_k)^T \Theta_k^{-1}(x_n-\mu_k)}}{(2\pi)^{d/2}\left|\Theta_k\right|^{1/2}} \tag{2}$$

  The probability density function $G\left(x_n|\mu_k, \Theta_k\right)$ is referred to as the k-th Gaussian component of the GMM.
- w_k is the mixture weight (or mixture coefficient) of the k-th Gaussian component. The mixture coefficients $w_1 \ldots w_K$ must be non-negative numbers satisfying

$$\sum_{k=1}^{K} w_k = 1 \tag{3}$$

To sum up, a Gaussian mixture model is governed by three parameter sets: mixture weights $\{w_1 \ldots w_K\}$, mean vectors $\{\mu_1 \ldots \mu_K\}$ and covariance matrices $\{\Theta_1 \ldots \Theta_K\}$.

The covariance matrices can either be full rank or constrained to be diagonal. The choice of the configuration is usually determined by the amount of data available for estimating the GMM parameters and how the GMM is used in a particular application. Although the full rank covariance matrices can be used if some degree of correlation exists between the features, the linear combination of diagonal covariance enables basis Gaussians to model the correlations between feature vector elements [6]. In most situations, the density modeling of a D-dimensional full covariance matrix can well be approximated using a larger order diagonal covariance matrix [2] [6] [7] [8].

In addition, the diagonal matrix GMMs outperform full-matrix GMMs empirically in some applications such as the signature verification and speaker identification in terms of accuracy and robustness [9] [10]. Another benefit of using diagonal matrix GMMs comes from the computation complexity since the repeated inversions of a D×D matrix are not required in diagonal-matrix GMMs, which is more computationally efficient especially on reconfigurable platforms [2].

In this study, we assume that the covariance matrices for all Gaussian components are diagonal. Therefore a covariance matrix Θ_k must satisfy

$$\Theta_k = \text{diag}\left(\sigma_k^2\right) \tag{4}$$

where $\sigma_k^2 = (\sigma_{k1}^2, \sigma_{k2}^2, \ldots, \sigma_{kD}^2)$ is a vector of variance value.

This assumption is widely taken by a variety of studies about GMMs such as [3], [11], [12], [13] and [9]. It simplifies the evaluation of Gaussian probability density function (PDF) while keeping or even improving the accuracy and robustness.

2.2 Expectation-maximization for GMMs

One elegant method of parameter estimation is the Expectation-Maximization (EM) algorithm. The EM algorithm is a general way to solve parameter estimation problems in machine learning. A particular adaptation of the EM algorithm, EM for *Gaussian mixture models (EM-GMM)*, can be used to estimate the parameters of a GMM.

The EM-GMM algorithm estimates the parameters of a GMM in an iterative manner. We first choose an initial parameter set arbitrarily and then update the parameter set by alternating between the following two steps until a predefined convergence condition is met:

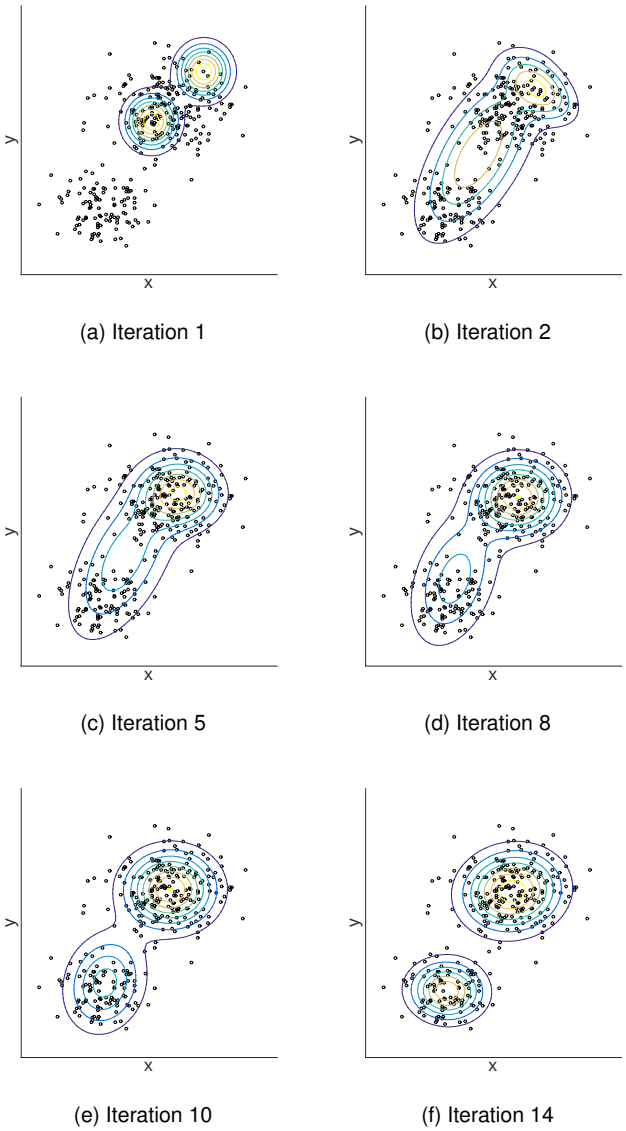
- Expectation step (E step): Compute a responsible value r_{nk} for each data instance x_n with respect to each Gaussian component k using the current estimation of parameter value $\{w_1 \ldots w_K\}$, $\{\mu_1 \ldots \mu_K\}$ and $\{\Theta_1 \ldots \Theta_K\}$. More specifically, the responsible value r_{nk} is defined and computed by

$$r_{nk} = \frac{w_k G(x_n|\mu_k, \Theta_k)}{\sum_{j=1}^{K} w_j G(x_n|\mu_j, \Theta_j)} \tag{5}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

1: **procedure** SERIAL EM-GMM ▷ Original EM-GMM
2: **while** stop condition not met **do**
3: **for** $n \leftarrow 1$ to N **do**
4: $s \leftarrow 0$
5: **for** $k \leftarrow 1$ to K **do**
6: $g_k \leftarrow w_k G(x_n|\mu_k, \Theta_k)$
7: $s \leftarrow s + g_k$
8: **for** $k \leftarrow 1$ to K **do**
9: $r_{nk} \leftarrow \frac{g_k}{s}$
10: $N_k \leftarrow 0$, $w \leftarrow 0$, $\mu \leftarrow 0$, $\sigma \leftarrow 0$
11: **for all** $n \in 1 \dots N, k \in 1 \dots K$ **do**
12: $N_k \leftarrow N_k + r_{nk}$
13: **for all** $k \in 1 \dots K$ **do**
14: $w_k \leftarrow \frac{N_k}{N}$
15: **for all** $n \in 1 \dots N, k \in 1 \dots K, d \in 1 \dots D$ **do**
16: $\mu_{kd} \leftarrow \mu_{kd} + \frac{r_{nk} x_{nd}}{N_k}$
17: **for all** $n \in 1 \dots N, k \in 1 \dots K, d \in 1 \dots D$ **do**
18: $\sigma^2_{kd} \leftarrow \sigma^2_{kd} + \frac{r_{nk}(x_{nd} - \mu_{kd})^2}{N_k}$
19: **for all** $k \in 1 \dots K$ **do**
20: $\Theta_k \leftarrow$diag(σ^2_k)
21: **end procedure**

Fig. 1. Serial EM-GMM algorithm



(a) Iteration 1 (b) Iteration 2

(c) Iteration 5 (d) Iteration 8

(e) Iteration 10 (f) Iteration 14

Fig. 2. Example run of EM-GMM

- Maximization step (M step): Estimate new parameter sets $\{w_1^+ \dots w_K^+\}$, $\{\mu_1^+ \dots \mu_K^+\}$ and $\{\Theta_1^+ \dots \Theta_K^+\}$ with the responsible value obtained in the E step. Then replace the old parameter sets by new ones. More specifically, the new parameter sets are computed by

$$w_k^+ = \frac{N_k}{N} \tag{6}$$

$$\mu_k^+ = \frac{\sum_{n=1}^{N} r_{nk} x_n}{N_k} \tag{7}$$

$$\Theta_k^+ = \frac{\sum_{n=1}^{N} r_{nk}(x_n - \mu_k^+)(x_n - \mu_k^+)^T}{N_k} \tag{8}$$

where

$$N_k = \sum_{n=1}^{N} r_{nk} \tag{9}$$

The original EM algorithm for Gaussian mixture models is shown in Fig. 1. Sufficient details are provided in the pseudocode to expose the patterns in memory access and computations. In Fig. 1, Line 3 to Line 9 correspond to the expectation step; Line 10 to Line 20 correspond to the maximization step.

Fig. 2 shows an example of the EM-GMM algorithm on a set of two-dimensional data instances. The two concentric ellipses stand for two Gaussian components respectively. In each component, the data instance located at the inner of an ellipse has larger responsible value with respect to the Gaussian component represented with the solid ellipse. In our contour, the orange ellipse represents the largest value while the violet one represents the smallest value.

2.3 Existing acceleration system for the EM-GMM

There are some existing acceleration solutions for the EM-GMM algorithm based on different platforms such as clusters and Graphics Processing Units (GPUs).

Kwedlo [14] implement the EM-GMM models on a NUMA system using OpenMP. They use a row-wise block striped decomposition of large arrays storing feature vectors and posterior probabilities. Additionally, some NUMA optimizations are also described. Their experiments show that the algorithm scales very well with respect to the number of cores, achieving over 50 times speedup with 64 cores. Also, the NUMA optimizations significantly improve its performance.

Yang et al. [15] propose and implement a comprehensively optimized parallel learning system on multicore clusters for GMM model, named Distrim. The system promotes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

memory sharing among threads, maximizes the usage of computational resources of multicore clusters and decreases time and space consumption. Their experiments, which is deployed on a 30-node cluster demonstrates that Distrim outperforms Hadoop in terms of both efficiency (2 to 40 times faster) and scalability.

Kumar et al. [11] and Andrew Pangborn [16] present an implementation of EM-GMM using GPUs following the 'single instruction multiple threads' model. The speed of this implementation scales up with the number of GPU cores. In Kumar's experiments, the implementation achieves a maximum of 164 times speedup compared to a naive single-threaded C implementation on the CPU platform. Machlica et al. [17] also presents an efficient and robust implementation of the estimation of GMM statistics used in the EM algorithm on GPU. Compared with [11] and [16], it pays more attention to the memory management of the data adhering to the rules of coalesced access and reuse the data as much as possible, thus achieving at least 5 times speedup over Kumar [11] in nearly the same configuration.

While training the parameters of EM-GMM, the larger number of input samples we trained, the better result we may have. Although a large amount of work focusing on accelerating EM-GMM on CPU and GPU platforms greatly improves the performance, the EM-GMM still expects more computing power to train more inputs with larger dimension or component parameters. The reconfigurable platform such as FPGA is a good alternative to CPU and GPU for EM-GMM training due to its hardware-based computing characteristic. Some work of optimizing EM-GMM on FPGA [18] [19] improves the performance a lot only certain circumstances, such as small dimension parameter, few samples. As the EM-GMM algorithm is not naturally fit into FPGA and the limited space of an FPGA chip can only fit designs with a small dimension (D) and component (K) parameter, there is much potential to further improve the performance if the EM-GMM algorithm can be transformed into a pipeline-friendly one. Also, Designing strategies to support a wide range of dimension and component parameters on reconfigurable platforms makes it a practical HPC EM-GMM engine to real-world applications.

3 A HARDWARE-FRIENDLY EM-GMM DESIGN

We first propose a new algorithm, the pipeline-friendly EM-GMM algorithm that is easier to be adapted and implemented as well as more effective on reconfigurable platforms. This algorithm comprises two parts. The collection procedure of statistical evidence contains the major computation of E steps and M steps, thus it is the most computationally demanding part of the algorithm. The procedure of updating parameters from the result of previous collection step is invoked infrequently compared with the collection procedure. In our hardware framework, we map the procedure of collection to the reconfigurable logics while leaving the process of updating parameters to the CPUs (shown in Fig. 3)

The collection of statistical evidence includes multiplication and addition operations that are proportional to the number of dimensions and components of the EM-GMM

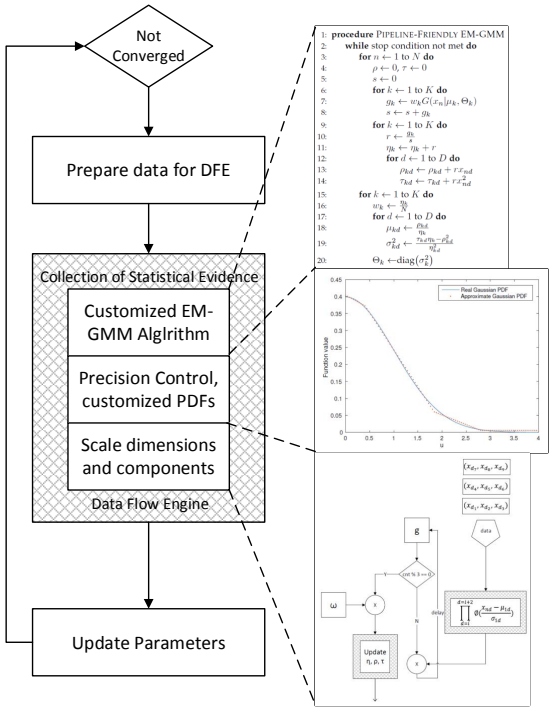


Fig. 3. Hardware-friendly EM-GMM framework

algorithm. After mapping the pipeline-friendly EM-GMM algorithm framework to the reconfigurable platform, it is likely that such kind of design can only support models with a small number of dimensions and components. For real applications like speaker recognition [2] [20] [21] and image segmentation [22] [23] [24] where the dimensions of the data set and the components clustered range from tens to hundreds, directly mapping the pipeline-friendly EM-GMM algorithm to reconfigurable platforms becomes inapplicable because the hardware logic resources are not enough.

By carefully profiling and analyzing the existing hardware design, we summarize that

- all numbers are represented by IEEE standard 32-bit single precision floating point. The floating-point arithmetic operators are quite resource-expensive.
- most logic resources contribute to the computation of Gaussian probability density function. It contains not only simple arithmetic like addition, subtraction, multiplication, and division, but also square root and exponential functions. Note that floating-point square root and exponential function evaluations are extremely expensive on hardware.
- the computational cost of Gaussian PDF is proportional to the product of D and K. So increasing D or K will increase logic resources significantly. In our straightforward design, we only support D=2 and K=2.

To achieve a dedicated hardware engine to support real applications, we propose the following three possible solutions to the challenges mentioned above accordingly.

- We use customized fixed-point number representation. By carefully profiling each variable for a given

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

data set and configuration, fined-grain tuned fixed-point representation results in better performance and less resource usage as well as enough accuracy.
- A customized piece-wise linear function approximation to the Gaussian PDF. Piece-wise linear function outperforms Gaussian PDF in terms of logic resources usage. After carefully choosing the number of pieces and the coefficients of the function, it can achieve an accurate enough or even better result compared to Gaussian PDF.
- We propose algorithms to support a wide range of dimensions or/and components by fitting multiple pieces of smaller dimensions or/and components into the limited space of an FPGA chip. In this way, even a single reconfigurable platform with limited resources can support real applications with large dimensions and components, with a significant speedup over CPU/GPU platforms.

4 A PIPELINE-FRIENDLY EM-GMM ALGORITHM

The original EM-GMM algorithm does not fit into a fully-pipelined hardware design. This is because the data dependency in the original EM-GMM algorithm makes it impossible to stream the data only once in each EM iteration.

We propose a pipeline-friendly EM-GMM algorithm in which the data set is only streamed once in each EM iteration. We named the algorithm *pipeline-friendly EM-GMM* because it can easily be adapted and implemented as a fully-pipelined hardware architecture.

4.1 Updating mixture weights and mean vectors

One important step to compute the new mean vectors is to compute the value $N_1 \dots N_k$. We set up a set of variables $\eta_1 \dots \eta_K$ to collect statistical information about $N_1 \dots N_K$ such that $N_1 \dots N_K$ can be calculated effortlessly at the end of EM iteration.

Before any data instance is processed, $\eta_1 \dots \eta_K$ are initialized to zero. When the n-th data instance x_n is loaded, the responsible value r_{nk} for all Gaussian components k can be computed according to Equation 5 with the current estimation of parameters. Then we update each element in $\eta_1 \dots \eta_K$ by

$$\eta_k \leftarrow \eta_k + r_{nk} \tag{10}$$

When all the N data instances are processed, we have

$$\eta_k = \sum_{n=1}^{N} r_{nk} = N_k \tag{11}$$

By Equation 9 and 10, we can compute the new mixture weights as follows

$$w_k^+ = \frac{N_k}{N} = \frac{\eta_k}{N} \tag{12}$$

On the other hand, we use a set of vectors $\rho_1 \dots \rho_K$ to collect statistical information about the weighted sum of data instances. Initially, we set all members in $\rho_1 \dots \rho_K$ to zero vectors. When a data instance x_n arrives, we update the elements in $\rho_1 \dots \rho_K$ by

$$\rho_k \leftarrow \rho_k + r_{nk} x_n \tag{13}$$

When all the N data instances are processed, we have

$$\rho_k = \sum_{n=1}^{N} r_{nk} x_n \tag{14}$$

Finally, at the end of the EM iteration, the new mean vector for the k-th Gaussian component can be computed by

$$\mu_k^+ = \frac{\rho_k}{\eta_k} \tag{15}$$

4.2 Updating variance vectors

We assume that the covariance matrices of all Gaussian components are diagonal. Therefore, we may decompose the value of each variance value in each Gaussian component by Equation 4 and 8

$$\begin{aligned} \sigma_{kd}^{2+} &= \frac{\sum_{n=1}^{N} r_{nk}(x_{nd} - \mu_{kd}^+)^2}{N_k} \\ &= \frac{\sum_{n=1}^{N} r_{nk}\left(x_{nd}^2 - 2x_{nd}\mu_{kd}^+ + \left(\mu_{kd}^+\right)^2\right)}{N_k} \end{aligned} \tag{16}$$

It can be further derived to

$$\begin{aligned} \sigma_{kd}^{2+} =& \frac{\sum_{n=1}^{N} r_{nk} x_{nd}^2}{N_k} \\ &- \frac{\sum_{n=1}^{N} 2 r_{nk} x_{nd} \mu_{kd}^+}{N_k} \\ &+ \frac{\sum_{n=1}^{N} r_{nk} \left(\mu_{kd}^+\right)^2}{N_k} \end{aligned} \tag{17}$$

We first focus on the second term in Equation 17. Along with equation 7, we have

$$\frac{\sum_{n=1}^{N} 2 r_{nk} x_{nd} \mu_{kd}^+}{N_k} = 2\mu_{kd}^+ \mu_{kd}^+ = 2\left(\mu_{kd}^+\right)^2 \tag{18}$$

We then focus on the third term in Equation 17. We have

$$\frac{\sum_{n=1}^{N} r_{nk} \left(\mu_{kd}^+\right)^2}{N_k} = \left(\mu_{kd}^+\right)^2 \frac{N_k}{N_k} = \left(\mu_{kd}^+\right)^2 \tag{19}$$

Substitute Equation 18 and 19 into Equation 17, we have

$$\sigma_{kd}^{2+} = \frac{\sum_{n=1}^{N} r_{nk} x_{nd}^2}{N_k} - \left(\mu_{kd}^+\right)^2 \tag{20}$$

We consider the transformation in Equation 20 valuable because it enables us to compute the new covariance matrices without streaming the data into the algorithm again. Similar to the computation of the new mean vector, we use a set of vectors, $\tau_1 \dots \tau_K$, to collect statistical information about the first term in Equation 20 and compute the value of term at the end of the iteration.

Initially, we set all members in $\tau_1 \dots \tau_K$ to zero vectors. When a data instance x_n arrives, we update the elements in $\tau_1 \dots \tau_K$ by

$$\tau_k \leftarrow \tau_k + r_{nk} x_n^2 \tag{21}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

procedure PIPELINE-FRIENDLY EM-GMM
  while stop condition not met **do**
    for $n \leftarrow 1$ to N **do**
      $\rho \leftarrow 0, \tau \leftarrow 0$
      $s \leftarrow 0$
      for $k \leftarrow 1$ to K **do**
        $g_k \leftarrow w_k G(x_n|\mu_k, \Theta_k)$
        $s \leftarrow s + g_k$
      for $k \leftarrow 1$ to K **do**
        $r \leftarrow \frac{g_k}{s}$
        $\eta_k \leftarrow \eta_k + r$
        for $d \leftarrow 1$ to D **do**
          $\rho_{kd} \leftarrow \rho_{kd} + r x_{nd}$
          $\tau_{kd} \leftarrow \tau_{kd} + r x_{nd}^2$
    for $k \leftarrow 1$ to K **do**
      $w_k \leftarrow \frac{\eta_k}{N}$
      for $d \leftarrow 1$ to D **do**
        $\mu_{kd} \leftarrow \frac{\rho_{kd}}{\eta_k}$
        $\sigma_{kd}^2 \leftarrow \frac{\tau_{kd}\eta_k - \rho_{kd}^2}{\eta_{kd}^2}$
      $\Theta_k \leftarrow \text{diag}\left(\sigma_k^2\right)$
end procedure

Fig. 4. Pipeline-friendly EM-GMM algorithm.

When all the N data instances are processed, we have

$$\tau_k = \sum_{n=1}^{N} r_{nk} x_n^2 \tag{22}$$

By Equation 15, 20 and 22, the variance vector can be computed by

$$\sigma_{kd}^{2+} = \frac{\tau_{kd}}{\eta_k} - \frac{\rho_{kd}^2}{\eta_k^2} = \frac{\tau_{kd}\eta_k - \rho_{kd}^2}{\eta_k^2} \tag{23}$$

4.3 Algorithm summary

To summarize the algorithm transformations illustrated above, we present the pipeline-friendly EM-GMM algorithm as a piece of pseudocode in Fig. 4.

Note that the algorithmic transformation is lossless. No numerical or algorithmic approximation was taken in the transformation. Therefore, theoretically the pipeline-friendly algorithm should generate exactly the same results as the original one if we use the same dataset and same initial parameter sets, although practically the two result may be slightly different due to the change of computation sequences in certain parts. Such a similarity suggests that the pipeline-friendly algorithm will have similar accuracy and convergence speed as the original one.

The fundamental difference between the original and the pipeline-friendly EM-GMM is that the former requires data to be streamed into the corresponding hardware architecture three times while the latter only once. Other differences include the following.

- The E step and the M step become overlapped in the pipeline-friendly algorithm, while in the original algorithm the M step does not start until the E step ends.
- The original algorithm stores all the responsible value in the expectation step, which is not necessary. When the statistical information is collected in pipeline-friendly EM-GMM, the corresponding responsible value is discarded safely.

The differences suggest that most of the computations in the pipeline-friendly EM-GMM can be mapped to the reconfigurable platforms. In this study, we move the collection procedure of statistical evidence to the FPGA platform because they apply similar operations on a large number of different data instances. We leave the rest of computations to the CPU because they are infrequently invoked in comparison to the collection of statistical evidence. Moving these computations to the FPGA platform will waste valuable logic resources.

Note that although the pipeline-friendly EM-GMM algorithm is designed for the ease of hardware implementation, it can be implemented in a software form. The software implementation may have higher performance than the original EM-GMM as it reduces the number of memory accesses. Related experimental results can be found in Section 7.

5 CUSTOMIZED GAUSSIAN PDF

The most complicated computation of the pipeline-friendly EM-GMM is the evaluation of Gaussian PDF. We design a Gaussian probability density function evaluation unit that uses fixed-point arithmetic to achieve a similar level of accuracy. We aim to achieve both low hardware cost and satisfactory accuracy.

5.1 Function evaluation strategy

Thomas proposes a Piecewise-CLT and Table-Hadamard Gaussian random number generators in FPGAs [25], [26]. Lee et al. [27] and Detrey et al. [28] design and optimize function evaluation units for three elementary functions in fixed-point arithmetic. They also provide a series of valuable suggestion on function evaluation problem in general. In our design, we are not proposing a general approach for evaluating PDF. Instead, we employ a design that is fully customized according to the accuracy requirement and input value range of Gaussian PDF.

The basic method for computing Gaussian PDFs is shown in Equation 2. Direct evaluation of such a function in reconfigurable platforms like FPGAs is extremely expensive as the function includes complicated computations such as matrix determinant, square root, matrix inverse, matrix multiplication and exponent.

Note that the diagonal covariance matrix suggests that the attributes are conditionally independent. Therefore the original PDF can be decomposed into a series of one-dimensional Gaussian distribution functions.

$$\begin{aligned} G\left(x_n|\mu_k, \Theta_k\right) &= \prod_{d=1}^{D} G\left(x_{nd}|\mu_{kd}, \sigma_{kd}^2\right) \\ &= \prod_{d=1}^{D} \frac{1}{\sigma_{kd}} \prod_{d=1}^{D} \phi\left(\frac{x_{nd} - \mu_{kd}}{\sigma_{kd}}\right) \end{aligned} \tag{24}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

where $\phi(u)$ is the standard one-dimensional Gaussian PDF defined by

$$\phi(u) = \frac{e^{-\frac{1}{2}u^2}}{\sqrt{2\pi}} \tag{25}$$

We can reduce the implementation cost of the Gaussian PDF evaluation unit by considering the properties of standard one-dimensional Gaussian PDFs. First, the standard Gaussian PDF $\phi(u)$ is even. We only need to approximate the positive part of $\phi(u)$. For all $u < 0$, we compute $\phi(-u)$ instead. Second, 99.7% of the observations fall within three standard deviations of the mean in theory , and only a small fraction of observations (0.3%) lies outside this range. Therefore we only need to compute a relatively accurate value when $0 \leq u \leq 3$ where $\sigma = 1$ in our cases.

Besides, approximating the Gaussian PDF evaluation units with other functions can further reduce the computational cost. When choosing the right approximation function units, we are following the basic two guidelines:

- Minimize the complexity of the design. The polynomial functions tend to be less expensive than the hyperbolic tangent. The nonpolynomial functions can also be replaced by piecewise linear approximations without losing the benefits of nonpolynomial functions [29].
- The error between the two functions can be calculated through symmetric Kullback-Leibler divergence [30].

Thus, taking into account all these guidelines, we propose a method to approximate the standard Gaussian PDF using piecewise linear approximation. We design our approximation function $\hat{\phi}(x)$ according to the following two main design strategies:

- $\hat{\phi}(x)$ should be as accurate as possible when $0 \leq x \leq 3$. For all $x > 3$, we may manually set $\phi(u)$ to be a small positive constant. This is to keep the statical fidelity of the algorithm.
- The computations involved in $\hat{\phi}(u)$ should be as simple as possible. For a piecewise linear function, the cost of logic resources are highly related to the number of pieces in the function.

To balance the trade-off between the two properties and keep the error as small as possible, we choose to use the following piecewise linear function to approximate the standard one-dimensional Gaussian PDF.

$$\hat{\phi}(u) = \begin{cases} -0.0781u + 0.4041, & 0 \leq u < 0.4053 \\ -0.2183u + 0.4609, & 0.4053 \leq u < 1.8340 \\ -0.0568u + 0.1647, & 1.8340 \leq u \leq 3.0 \\ \hat{\phi}(3.0), & u > 3.0 \end{cases} \tag{26}$$

It can be observed from Fig. 5 that the piecewise linear approximation fits the real Gaussian PDF well. Notes that inaccurate Gaussian PDF value may not lead to negative results in GMM-based systems if the error is properly controlled [31]. Therefore we consider our function approximation accurate enough as the absolute error is below 0.005

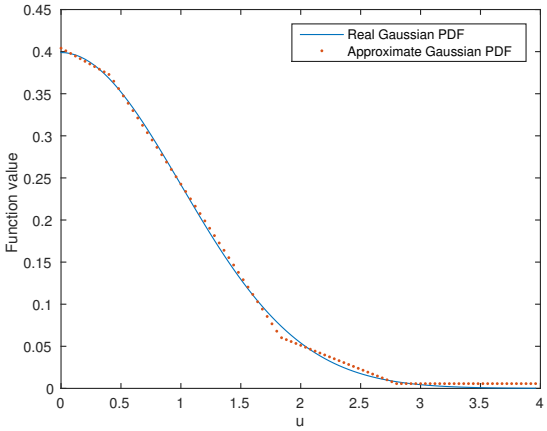


Fig. 5. Approximate one-dimensional Standard Gaussian PDF

for most of the input value. Moreover, the function involves only three multiplication operations, which is inexpensive in terms of resource consumption on most reconfigurable platforms. We will further show the accuracy of the approximation function when applying it to test cases in section 7.

5.2 Precision control with shifting

Consider the product of standard one-dimensional Gaussian PDF value in Equation 24

$$H(x_n|\mu_k, \sigma_k) = \prod_{d=1}^{D} \phi\left(\frac{x_{nd} - \mu_{kd}}{\sigma_{kd}}\right) \tag{27}$$

This product may become a very small positive value when D is large. More specifically, for a data instance x_n in a D-dimensional space, the maximum possible value of this product is

$$\begin{aligned} p_{max}(D) &= \max H(x_n|\mu_k, \sigma_k) \\ &= \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi}} \\ &= (2\pi)^{(-D/2)} \end{aligned} \tag{28}$$

The function decreases exponentially towards zero. We aim to control the value of $p_{max}(D)$ to be a constant for the ease of allocating a proper number of bits to represent a result. Note that if we shift each result of one-dimensional Gaussian PDF evaluation to the left by h bits, the corresponding maximum value is

$$p'_{max}(D) = 2^{hD} \cdot (2\pi)^{-D/2} = 2^{(h-1/2)D} \cdot \pi^{-D/2} \tag{29}$$

Take logarithm on both sides with base 2, we have

$$\log_2 2^{(h-1/2)D} + \log_2 \pi^{-D/2} = \log_2 p'_{max}(D) \tag{30}$$

To control the maximum value of this function, we may set $p'_{max} = 1$ to build constant shifting scheme regardless of the value of D. More specifically, by taking $p'_{max} = 1$, Equation 30 can be simplified as

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

$$\left(h - \frac{1}{2}\right) D - \frac{D}{2} \log_2 \pi = 0 \tag{31}$$

Note that D can be canceled as $D > 0$. We can then solve the equation with respect to h

$$h_0 = \frac{1}{2} \left(\log_2 \pi + 1\right) \tag{32}$$

Note that $h_0 \approx \frac{4}{3}$, which means that shifting 4 bits in 3 evaluations could be an approximation solution. we consider shifting 2 bits every 3 evaluations and 1 bit otherwise. Let h_t be the number of bits shifted in the t-th Gaussian PDF evaluation, then

$$h_t = \begin{cases} 2, & \text{if mod } (t, 3) = 1 \\ 1, & \text{otherwise} \end{cases} \tag{33}$$

Taking this shifting scheme, the maximum value of the product is controlled to be less than 1.5 when $D \leq 20$.

6 Expanding Components and Dimensions

Gaussian mixture model (GMM) is widely used in different applications. In some applications like image segmentation, the major challenge is to feed the high-dimensional observations to the EM-GMM algorithm, while other applications like handwriting recognition are targeting clustering a large number of components. It is highly important that our pipeline-friendly algorithm on reconfigurable platforms is able to support different kinds of real applications.

However, the limited logic resources are the major obstacles when mapping EM-GMM with large dimensions and components on hardware platforms. Even if applying our customized Gaussian PDF evaluation function in fixed points representation, which dramatically reduces logic units, our design can only support observations with D=6 and K=6 at the same time. In this section, we propose our methods to support a wide range of components (K) and dimensions (D) of pipeline-friendly EM-GMM on reconfigurable platforms in order to provide a unified solution to real-world applications.

6.1 Expanding components (K)

For configurations where all components and dimensions cannot be mapped to the hardware, we have to trade off between performance and resources. According to the profiling result mentioned in section 3, the evaluation of Gaussian PDFs costs most of the logic units. Moreover, it is proportional to the number of components directly as we can see from the EM-GMM algorithm (Fig. 4).

In order to achieve the throughput goal of one datum per cycle, the loop is unrolled. It does so at the expense of logic consumption, where the loop body logic, is replicated for every Gaussian component. However, when the number of Gaussian components turns large, the required logic resources may exceed the maximum resources on one FPGA. We propose a supplement to the fully pipelined-friendly algorithm in Fig. 4 in order to deal with the situations where the number of Gaussian components, K, is quite large.

Figure 6 shows the approach to scale the number of components in hardware. Instead of cascading all Gaussian probability density function evaluation units of different Gaussian components into FPGA all at once, only a fixed number (h) of them are deployed and reused for different components. For the simplicity of explanation, we assume that

```
1:  procedure PIPELINED-FRIENDLY EM-GMM-2
2:      while stop condition not met do
3:          for n ← 1 to N do
4:              ρ ← 0, τ ← 0
5:              s ← 0
6:              j ← 1
7:              while j ≤ K do
8:                  for k ← j to j + h do
9:                      g_k ← w_k G(x_n|μ_k, Θ_k)
10:                     s ← s + g_k
11:                 j ← j + h
12:             for k ← 1 to K do
13:                 r ← g_k / s
14:                 η_k ← η_k + r
15:                 for d ← 1 to D do
16:                     ρ_kd ← ρ_kd + r x_nd
17:                     τ_kd ← τ_kd + r x_nd^2
18:         for k ← 1 to K do
19:             w_k ← η_k / N
20:             for d ← 1 to D do
21:                 μ_kd ← ρ_kd / η_k
22:                 σ^2_kd ← (τ_kd η_k − ρ^2_kd) / η^2_kd
23:             Θ_k ← diag(σ^2_k)
24: end procedure
```

Fig. 6. A Pipelined-friendly EM-GMM algorithm dealing with the circumstance when the Gaussian components K is quite large. A tiling alike approach is designed to make a tradeoff between the performance and logic resources.

$$K = h \cdot l \tag{34}$$

When a data instance x_n is transferred to the data flow engine, it takes l cycles to evaluate K Gaussian PDFs, with each cycle collecting h Gaussian PDFs simultaneously. In the first 1 to $l-1$ cycles, the logics for updating η, ρ, τ (from line 12 to 17 in Fig. 6) are idle because the sum of all the Gaussian PDFs (denoted as s at line 10 in Fig. 6) is not ready yet. These logic units work every l cycles when the dependency is satisfied.

In theory, h and l can be set to any value as long as the equation is satisfied. The correctness of the design is guaranteed. However, the value of h stands for the number of Gaussian probability function evaluation units work simultaneously each cycle, thus it greatly influences the overall performance of the hardware design. The larger the value of h is, the more efficient of the design is. Generally, we suggest that h should be set as the largest possible value supported by the available hardware resources.

6.2 Expanding dimensions (D)

For some applications which need high dimensional data, the design has to handle situations when it cannot hold

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

1: **procedure** ABBREVIATED EM-GMM
2: **for** $n \leftarrow 1$ to N **do**
3: **for** $k \leftarrow 1$ to K **do** ▷ compute each PDF
4: $g_k \leftarrow w_k G(x_n|\mu_k, \Theta_k)$
5: $s = \sum_{k=1}^{k} g_k$ ▷ sum PDFs
6: **for** $k \leftarrow 1$ to K **do** ▷ update η, ρ, τ
7: update η, ρ, τ
8: **end procedure**

Fig. 7. Abbreviated EM-GMM algorithm

all dimensions of data when calculating the Gaussian PDF. So, in this subsection, we discuss how to configure a fixed number of dimensions and reuse the logic to iterate through all dimensions.

The operations on each data instance in the pipelined-friendly EM-GMM algorithm (Fig. 4) can be abbreviated to three steps: 1) compute each Gaussian PDF, 2) sum all PDFs, 3) update η, ρ, τ.

Step 1 and Step 2 correspond to the E-Step and Step 3 corresponds to the M-step of the EM-GMM algorithm. The dimension (D) is accessed both in E-step and M-step. We also see from abbreviated EM-GMM algorithm (Fig. 7) that the two references of D are not continuous. This is the major challenge in scaling D on FPGAs. When evaluating g_k, each data instance with its dimensions $D = \{d_1, d_2, \ldots, d_n\}$ have to be scanned at the same cycle according to equation 24. It is impossible to split D-dimensional data to multiple pieces, streaming only part of $D = \{d_1, d_2, \ldots, d_{n/2}\}$ in the first cycle, then streaming the rest of $D = \{d_{n/2+1}, d_{n/2+2} \ldots, d_n\}$ in the second cycle.

If D is extremely large, we need to decrease k inevitably as the logic resources in one FPGA are limited. Performing such kind of operation goes to the opposite side of the expanding K optimization described in the last subsection.

6.2.1 *Case 1:* $k = 1$

In order to illustrate the idea of expanding D, we first discuss a simple case where the number of clusters is 1. Assuming that logic resources in one FPGA can only support the EM-GMM algorithm with the maximum dimension D to 3, but the input data instances are of dimension D of 9. By changing the order of E-step and M-step, the evaluation of Gaussian PDF g can be decomposed to multiple cycles. Equation 35 explains the basic philosophy.

$$\begin{aligned} g_1 &= w_1 \prod_{d=1}^{9} \phi\left(\frac{x_{nd} - \mu_{1d}}{\sigma_{1d}}\right) \\ &= w_1 \prod_{d=1}^{3} \phi\left(\frac{x_{nd} - \mu_{1d}}{\sigma_{1d}}\right) \cdot \prod_{d=4}^{6} \phi\left(\frac{x_{nd} - \mu_{1d}}{\sigma_{1d}}\right) \\ &\quad \cdot \prod_{d=6}^{9} \phi\left(\frac{x_{nd} - \mu_{1d}}{\sigma_{1d}}\right) \end{aligned} \tag{35}$$

Figure 8 shows how each 9D input data instance is split into three 3D ones. In this design, limited logic

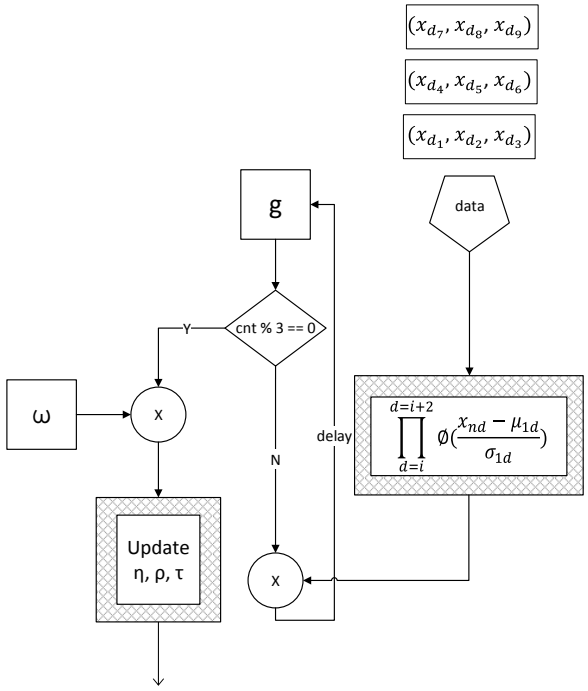


Fig. 8. Scale dimensions (D) when k = 1

resources contributes to the mathematical calculation of $\prod_{d=i}^{i+2} \phi\left(\frac{x_{nd}-\mu_{1d}}{\sigma_{1d}}\right)$, which is repeatedly used 3 times for a 9D data instance. In each cycle, the intermediate result of each product is stored in BRAM of FPGA. In this example, we follow the following steps:

- compute $g_1^{(1)} = w_1 \prod_{d=1}^{3} \phi\left(\frac{x_{nd}-\mu_{1d}}{\sigma_{1d}}\right)$, and store $g_1^{(1)}$ into BRAM.
- compute $g_1^{(2)} = g_1^{(1)} \prod_{d=4}^{6} \phi\left(\frac{x_{nd}-\mu_{1d}}{\sigma_{1d}}\right)$, where $g_1^{(1)}$ is read from BRAM in this cycle. we then put $g_1^{(2)}$ back to BRAM.
- compute $g_1^{(3)} = g_1^{(2)} \prod_{d=7}^{9} \phi\left(\frac{x_{nd}-\mu_{1d}}{\sigma_{1d}}\right)$, where $g_1^{(2)}$ is also read from BRAM in this cycle. Now we have $s = g_1^{(3)}$. In this cycle, we also evaluate η, ρ, τ when s is ready.

6.2.2 *Case 2:* $k > 1$

Now we discuss a more complex situation where the number of clusters k is greater than one and the dimension of the data instances is still quite large. The solution of this situation is a combination of the last two points.

- for $k \geq 1$, we follow the ideas in section 6.1 by calculating a fixed number of clusters in a single cycle in FPGA and repeat this operation in multiple cycles to iterate through all clusters.
- for a large dimension D, we follow the ideas in section 6.2.1. From the view of the algorithm, we reorder the calculation sequence by updating the s in the last step. Then we can configure the FPGA to calculate part of the dimensions of the data instances

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

and store the intermediate results in BRAM. At the cycle while iterating all dimensions and s is ready, we update the parameters η, ρ, τ.

7 Experiment

We choose the Maxeler Data Flow Engine (DFE) as our primary reconfigurable platform for its programming flexibility and performance [32].

The simplest and most widely used initialization strategies, the random initialization, are employed to generate data for our experiments [33] [34]. More specifically, we pick K-cluster data points randomly as mode means followed by initializing the individual covariance as the covariance of the input data. To further maximize the likelihood, we repeat the initialization process multiple times from different random positions and select the solution maximizing the likelihood among those runs [33].

One of the restrictions in our design is the form of the diagonal covariance matrix. However, the effect of using a set of full covariance matrix Gaussians can be equally obtained by using a larger set of diagonal covariance Gaussians [6]. Therefore, our pipelined EM-GMM design can overcome the restriction in some degree if it is able to train GMM parameters from data sets with large dimension (D) and component (K). We generate and test 12 cases with the dimension (D) ranging from 6 to 96 and the component (K) ranging from 2 to 256. 10^6 data instances are sampled for each test case. In each test case, we run the EM-GMM algorithm for 10 iterations and compare the accuracy, performance, power consumption, etc. with the CPU and GPU platforms.

Our design is able to support test cases with much larger D and K configurations. However, the GPU-based EM implementation in [16] whose source code is available at [35] can only support at most 96-dimensional data due to the limited shared memory on the latest GPUs. So we also restrict D to 96 in order to have a fair performance comparison.

7.1 Accuracy Results

We test the accuracy of three implementations: (1) a CPU implementation of the original EM-GMM algorithm; (2) a CPU implementation of the pipeline-friendly EM-GMM algorithm; (3) an FPGA implementation of the pipeline-friendly EM-GMM algorithm.

The accuracy of a system is described by the average log-likelihood value [36] of the estimated parameters with respect to the data set. Larger log-likelihood value suggests better accuracy. Using the same data set and same initial parameters, we take the GMM parameters estimated by the three implementations after each iteration and compute the average log-likelihood respectively. Experimental results about accuracy are plotted in Fig. 9. As the differences in the accuracy between the two CPU implementations are too small to be visible, they are plotted as a single curve in each plot.

We can observe from Fig. 9 that our solution on reconfigurable platforms and the CPU-based solutions lead to similar accuracy after the same iteration for the same data set. The similarity on the accuracy suggests that the approximate Gaussian PDF evaluation unit is reliable. Moreover, our solution sometimes generates more accurate results than the CPU-based ones. We consider it very interesting as we do not expect a system involving approximations to be more accurate than the original one. We do not know the underlying reason behind this observation but it is probably because we use a small constant for small PDF value instead of zero, which may prevent the algorithm from premature convergence. Similar observations can be found in Monto Carlo localization in robotics [37].

7.2 Performance Results and Comparisons

7.2.1 CPU Performance

The performance is described by the number of data instances processed in every second. A data instance is considered to be processed in an iteration when all the computations related to the data instance are done in that iteration. We first test the performance of two CPU implementations, the standard EM-GMM algorithm, and our fully pipelined EM-GMM algorithm. In order to get comparable results, both methods are implemented in C++ and are highly optimized. The optimization strategies include the highest compiler optimization flags (-fast, -O3, -openmp) in the Intel compiler (icpc, version 2013.sp1), the vectorization, and the multi-thread executions with OpenMP.

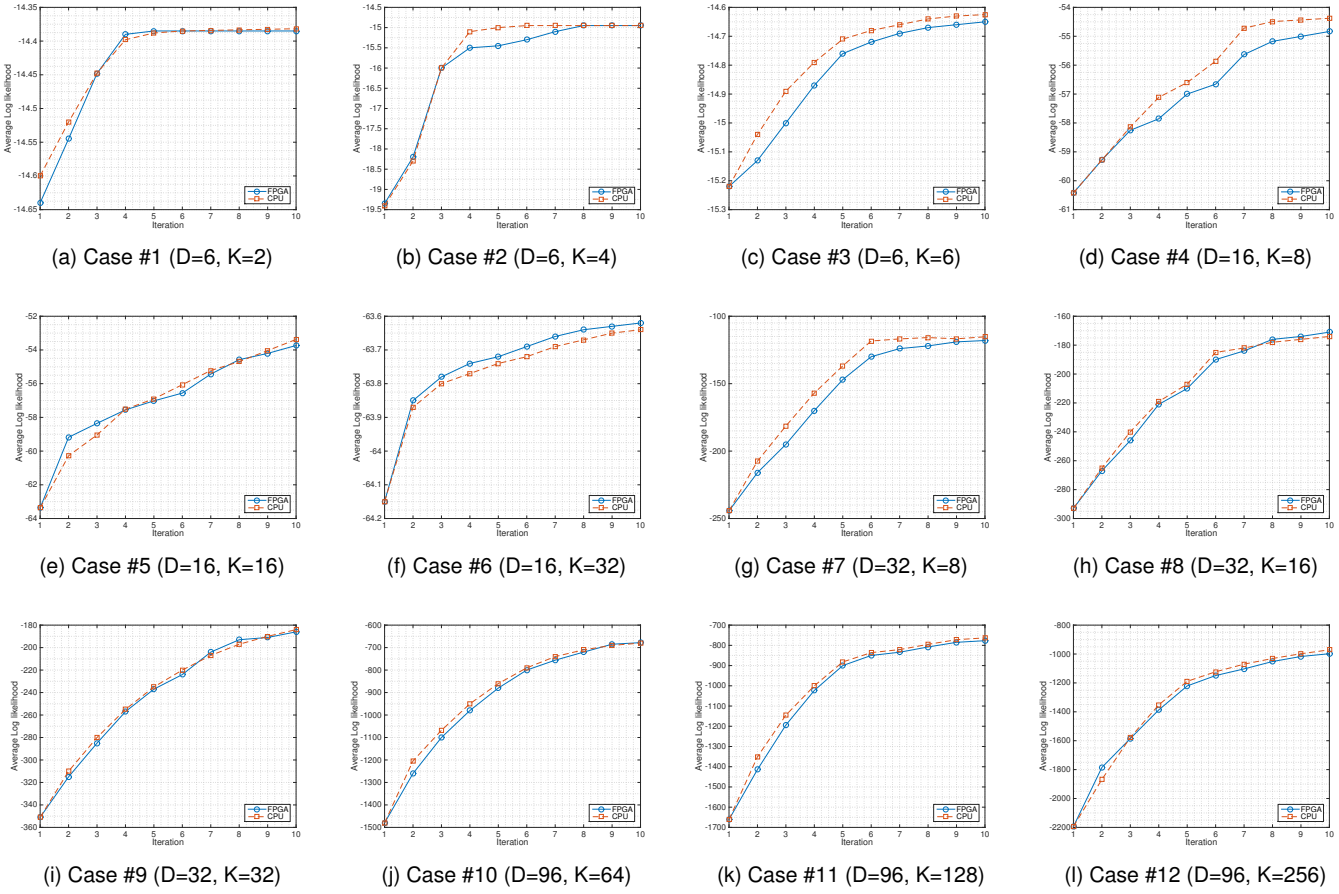
The two CPU implementations are deployed on an Intel Xeon E5645 server, which has 6 physical cores running at 2.4GHz and 24GB DDR3 memory. As the server is virtualized to 12 logical cores, we run the parallel EM-GMM algorithms with 1 to 12 threads and collect the best result as the performance future comparisons.

The performance result of the two CPU implementation is shown in Table 1. The performance of the standard EM-GMM algorithm and the pipelined one are recorded by "EM" and "PL-EM" respectively, and the speedup of P-EM over EM is recorded by "SU_C". The pipelined EM-GMM algorithm achieve better performance than the standard one on the CPU platform, as we predicted in Section 4.3. The performance gain of the pipelined EM-GMM algorithm mainly comes from two aspects. Firstly, the customized PDF evaluation units eliminate the evaluation of complicated math arithmetics such as $\exp$. The multiplication-addition evaluation form in our customized PDF facilitates the use of SIMD instructions. Secondly, the workflow of the pipelined EM-GMM algorithm merges four loops iterating each data instance into only one loop, which enables each data instance to isolate from others, resulting in better cache utilization because of the data locality.

7.2.2 GPU Performance

We also compare the systems with a GPU implementation described in [16]. We select it because the source code is available at [35] and the authors demonstrated that [16] outperforms other reference implementations. In addition, [16] is more suitable to work with data sets configured with large dimensions and components compared with the GPU implementation described in [11] because the latter stores the $K \times D \times N$ matrix completely into the GPU memory

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers



(a) Case #1 (D=6, K=2) (b) Case #2 (D=6, K=4) (c) Case #3 (D=6, K=6) (d) Case #4 (D=16, K=8)

(e) Case #5 (D=16, K=16) (f) Case #6 (D=16, K=32) (g) Case #7 (D=32, K=8) (h) Case #8 (D=32, K=16)

(i) Case #9 (D=32, K=32) (j) Case #10 (D=96, K=64) (k) Case #11 (D=96, K=128) (l) Case #12 (D=96, K=256)

Fig. 9. Accuracy Results (Average Log-Likelihood)

at once in the M-step, which is not always feasible for large data set. Note that the GPU implementation we use is not pipelined [16].

Although [16] can work in both diagonal covariance matrices and full covariance matrices, we configure it to work only for the diagonal covariance to form a fairer comparison because the full-covariance mode needs more computations. Moreover, we compile and run the GPU implementation in [16] on two platforms, the Nvidia Kepler K40C and the Nvidia Pascal TITAN-X, which is the state-of-art and the most powerful GPU currently. Apart from the "-O3" optimization flag, we also add the compilation flags "-arch sm_35" for Kepler K40C and "-arch sm_50" for Pascal TITAN-X to maximize the performance of the latest GPUs. Note that as the Pascal TITAN-X is a dual-GPU design, both GPUs are used in our tests.

The GPU-based performance results of Kepler K40C and Pascal TITAN-X are recorded by "K40C" and "TITAN-X" in Table 1 respectively. The Pascal TITAN-X GPU can run 2.3-2.7 times faster than the Kepler K40C due to the higher clock frequency, more computing cores and higher memory bandwidth. The GPU implementation uses shared memory to cache covariance matrices, but the shared memory is the limited resource. The number of dimensions that the GPU-based implementation supports is restricted by the size of the shared memory of the GPUs. It supports 24-dimensional data in [16] and in our cases, we can support 96-dimensional data. Note that although the Pascal TITAN-X GPU owns 96KB shared memory which is twice as large as the Kepler K40C, each sub GPU in TITAN-X owns 48KB shared memory due to its dual-GPU design.

7.2.3 FPGA Performance

We deploy the FPGA implementation on two different platforms. One is a Maxeler MAX3 acceleration card with a Xilinx Virtex-6 FPGA chip running at 150MHz and 24GB DDR3 onboard memory. The other is on a Maxeler MAX4 acceleration card with an Altera Stratix-V FPGA running at 200MHz to 250MHz and 48GB DDR3 onboard memory.

Experimental results are recorded in Table 1. The performance on Virtex-6 and Stratix-V platforms are denoted by "Virtex-6" and "Stratix-V" respectively. On each FPGA platform, we compare the performance of pipelined EM-GMM algorithm with those on the CPU and GPU platforms. "SU_P" denotes the speedup of FPGA over the pipelined EM-GMM algorithm, which is denoted by "PL-EM" on the CPU platform, while "SU_K" and "SU_T" represent the speedup over the Kepler K40C and Pascal TITAN-X GPU platforms respectively. It shows that our FPGA-based solution is significantly more efficient than other systems at least in our tested cases.

In the first experiment, the FPGA is running at 150MHz for all test cases. In the first three test cases where $D =$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2712152, IEEE Transactions on Computers

TABLE 1
Performance results (Instance per Seconds) of the pipelined EM-GMM algorithm on CPU, GPU and FPGA platforms

		CPU			Nvidia GPU		Maxeler FPGA: Max3(Virtex-6) and Max4(Stratix-V)							
Clock Freq		2.4GHz			745MHz	1.4GHz	150MHz				200Mhz - 250MHz			
Memory		24GB			12GB	12GB	24GB				48GB			
D	K	EM	PL-EM	SU_C	K40C	TITAN-X	Virtex-6	SU_P	SU_K	SU_T	Stratix-V	SU_P	SU_K	SU_T
6	2	4.17E6	4.86E6	1.17×	2.66E7	7.15E7	1.49E8	31×	6×	2×	2.48E8	51×	9×	3×
6	4	2.12E6	2.48E6	1.17×	1.69E7	4.02E7	1.49E8	60×	9×	4×	2.48E8	100×	15×	6×
6	6	1.45E6	1.70E6	1.17×	1.20E7	2.67E7	1.49E8	88×	12×	6×	2.48E8	146×	21×	9×
16	8	4.15E5	4.85E5	1.17×	3.47E6	8.12E6	3.70E7	76×	11×	5×	9.15E7	188×	26×	11×
16	16	2.05E5	2.39E5	1.17×	1.77E6	4.04E6	1.85E7	77×	10×	5×	4.57E7	191×	26×	11×
16	32	1.01E5	1.19E5	1.18×	8.23E5	1.98E6	9.25E6	78×	11×	5×	2.28E7	191×	28×	12×
32	8	1.98E5	2.32E5	1.17×	1.16E6	2.77E6	1.85E7	80×	16×	7×	4.57E7	196×	39×	16×
32	16	9.70E4	1.14E5	1.17×	5.77E5	1.39E6	9.25E6	81×	16×	7×	2.28E7	200×	40×	16×
32	32	5.01E4	5.89E4	1.18×	2.82E5	7.01E5	4.63E6	79×	16×	7×	1.14E7	193×	40×	16×
96	64	8.04E3	9.47E3	1.18×	2.08E4	5.17E4	7.70E5	81×	37×	15×	1.90E6	200×	91×	37×
96	128	3.97E3	4.69E3	1.18×	1.01E4	2.55E4	3.85E5	82×	38×	15×	9.50E5	202×	94×	37×
96	256	1.93E3	2.28E3	1.18×	4.94E3	1.23E4	1.92E5	84×	39×	16×	4.74E5	207×	96×	39×

TABLE 2
The speedup of the EM-GMM algorithm on the GPU and FPGA platforms over the CPU platform in terms of energy efficiency.

Power		80W	235W	250W	22.4W	26.2W
D	K	PL-EM	K40C	TITAN-X	Virtex-6	Stratix-V
6	2	1	1.9×	4.7×	109×	156×
6	4	1	2.3×	5.2×	215×	305×
6	6	1	2.4×	5.0×	313×	445×
16	8	1	2.4×	5.4×	272×	576×
16	16	1	2.5×	5.4×	276×	584×
16	32	1	2.4×	5.3×	278×	585×
32	8	1	1.7×	3.8×	285×	601×
32	16	1	1.7×	3.9×	290×	611×
32	32	1	1.6×	3.8×	281×	591×
96	64	1	0.7×	1.7×	290×	613×
96	128	1	0.7×	1.7×	293×	618×
96	256	1	0.7×	1.7×	301×	635×

6, we can see that the throughput of the solution keeps at a constant level at around 1.5×10^8 instances per second. The performance suggests that the system handles a data instance per cycle without being confronted with memory bottlenecks.

In other test cases, the performance decreases significantly because the logic resources on an FPGA chip are not able to support quite large D and K configurations. We follow the strategy in Section 6 to map a fixed D and K configuration and process each instance in multiple cycles. Configuring the Viertex-6 FPGA board with $D = 8$ and $K = 4$, four cycles are required to process one instance with $D = 16$ and $K = 8$. Similar configurations can be derived in the following test cases.

We also applied the same data set on a newer reconfigurable platform, the Maxeler MAX4 acceleration card with an Altera Stratix-V FPGA. Compared to Xilinx Virtex-6, the Altera Stratix-V FPGA owns much more resources and runs at a higher frequency. The Strativ-V FPGA are able to support processing data instances with $D = 8$ and $K = 8$ in one cycle, which is twice as fast as the Xilinx Virtex-6 FPGA if they are running at the same clock frequency. In the first three cases where D and K are small, the FPGA runs at the clock frequency of 250MHz, while for the rest of the cases, it runs at 200MHz.

The computing complexity of the pipelined EM-GMM design on reconfigurable platforms is $O(DK)$, as it is indicated in the performance results. The computing complexity of the current GPU implementation is (D^2K) because it is designed for both diagonal-covariance GMM and full-covariance GMM [16]. Although this can be refined to $O(DK)$, the GPU implementation also has the restriction of scaling dimension (D) because of the limited shared memory on GPUs.

In the best case, our solution on the Virtex-6 FPGA platform achieves 84 times speedup over a fully optimized CPU solution running on 6 cores, and 39 times speedup over the Nvidia K40C GPU solution, and 16 times speedup over the latest GPU platform, the Pascal TITAN-X. The Altera Stratix-V FPGA runs 1.67 to 2.46 times faster than the Virtex-6 FPGA in our cases, and we obtain promising speedups over the CPU and GPU platforms, which is 207 times over the CPU solution, and 96 times over the Kepler K40C GPU solution, and 39 times over the Pascal TITAN-X GPU solution.

Table 2 shows the speedup of the EM-GMM algorithm on GPU and FPGA platforms over the CPU platform in terms of energy efficiency, which is calculated by Equation 36,

$$E = \frac{IpS}{P} \tag{36}$$

where IpS is the number of instance processed per second and P is the power of each platform. In the best case, our solutions on the Virtex-6 and Stratix-V FPGA platform are 301 and 635 times more energy efficient than the CPU-based platform respectively. We also notice that the GPU implementation on K40C is less energy efficient than the CPU implementation in some cases with large dimension configurations. This is because the the computing complexity of the GPU implementation is higher than the CPU implementation, which is mentioned before.

We believe the reason behind the high speedup of our solution is that the fixed-point arithmetic and customized PDFs save hardware resources on the reconfigurable platform. The customized PDFs and the pipelined EM workflow can also be applied on CPU or GPU platforms. However, it is not feasible to perform fixed-point optimization on CPUs and GPUs. If we have higher logical resources on the reconfigurable platform, we can deploy an even larger number of Gaussian PDF evaluation units to enable more

TABLE 3
Coefficients for equation 37 with 95% confidence

Coeff	LUTs	FFs	BRAMs	DSPs
p_{00}	951.9	1795	-0.2857	0
p_{10}	-73.34	-95.58	1.8	0
p_{01}	1578	3091	0.4286	0
p_{11}	4730	8377	2	16
p_{02}	4.069	12.15	0.1429	0

TABLE 4
Error of Cost model of different logic units

D	K	LUTs	FFs	BRAMs	DSPs
8	2	0.0373%	0.2415%	0.9519%	0
8	3	0.0108%	0.0724%	1.9906%	0
8	4	0.0137%	0.0441%	0.1404%	0
8	5	0.0567%	0.0185%	0.1702%	0
8	6	0.0226%	0.0680%	0.9829%	0
8	7	0.0632%	1.0105%	0.6448%	0
8	8	0.0347%	1.0370%	0.2008%	0

complicated data with larger D and K to be processed by the system. The corresponding acceleration would be more significant.

7.3 Cost and performance model

Selecting good D and K values improves the performance a lot on reconfigurable platforms. Generally, the larger D, K values of a design are, the better performance it will gain. However, for a given reconfigurable platform with limited logic resources, what is the largest D and K values that can be supported? For a given D and K, what is the best strategy to decompose D and K into smaller value? Dinechin et al. [29] propose an architecture generator to output the synthesizable description given the specification of a function as input. When targeting our specific problem, we expect a simpler and more dedicated approach to quickly estimate the hardware cost and the performance according to the D and K values.

A generalized multivariate cost function is introduced in equation 37, where the variable x, y denote D, K and $f(x, y)$ denotes specific logic usage. In order to derive the coefficients, we collect different logic resource (LUTs, FFs, BRAMs and DSP) usage from 42 test cases, with $D = \{2 \dots 7\}, K = \{2 \dots 8\}$. Machine learning methods or similar regression algorithms can be applied to derive the coefficients from these 42 samples. In this case, the coefficients in Table 3 are derived by learning from the existing 42 samples.

$$f(x, y) = p_{00} + p_{10}x + p_{01}y + p_{11}xy + p_{02}y^2 \qquad (37)$$

Equation 37 and the coefficients in table 3 are the utilities for users to analyze the possible resource usage and potential performance gain for a specific configuration. We also set up another 7 test cases proving the representativity of equation 37. Table 4 shows the errors in percentage between the real measured logic resources and evaluated from our equation. It turns out that the DSPs fit perfectly in observation and evaluation. LUTs, FFs, and BRAMS also fit in a very good manner, which can be used as an evidence to prove the validity of our resource estimation equation.

8 Conclusion and Future Work

This paper presents a high-performance engine for Expectation-Maximization for Gaussian Mixture Models (EM-GMM) targeting reconfigurable platforms. By transforming the original EM-GMM algorithm, we first design a fully pipelined EM-GMM algorithm. A customized Gaussian PDF evaluation unit with reliable results but inexpensive resource consumption is then proposed. In order to meet the configuration requirements of real-world applications, we further extend our scheme to support a wide range of dimensions and components. We also derive a cost and performance model that estimates necessary logic resources quickly. Finally, we map our design on two different FPGA platforms to verify the correctness and performance, achieving a maximum of 207x speedup over a fully optimized CPU solution running on 6 cores and 96x speedup over a Kepler K40C GPU-based solution, and 39x speedup over a Pascal TITAN-X GPU-based solution. It provides a practical solution for real applications to train and explore better parameters for GMM with hundreds of millions of high dimensional input instances in a couple of hours, which would be impractical for most applications if it takes hundreds of hours to run.

In the future, we may integrate our solution into specific applications such as object tracking, speech recognition and data visualization to maximize their performance. We are also planning to explore the potentials of other important but computationally demanding machine learning algorithms on reconfigurable platforms.

References

[1] H. Greenspan, A. Ruf, and J. Goldberger, "Constrained Gaussian Mixture Model Framework for Automatic Segmentation of MR Brain Images," *IEEE Transactions on Medical Imaging*, vol. 25, no. 9, pp. 1233–1245, 2006.

[2] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digital Signal Processing*, vol. 10, no. 1, pp. 19–41, 2000.

[3] C. Stauffer and W. E. L. Grimson, "Adaptive Background Mixture Models for Real-time Tracking," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, 1999.

[4] R. Kohavi, "A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection," in *IJCAI*, vol. 14, no. 2, 1995, pp. 1137–1145.

[5] C. Guo, H. Fu, and W. Luk, "A Fully-pipelined Expectation-maximization Engine for Gaussian Mixture Models," in *International Conference on Field-Programmable Technology (FPT)*, 2012, pp. 182–189.

[6] D. Reynolds, "Gaussian Mixture Models," *Encyclopedia of Biometrics*, pp. 827–832, 2015.

[7] H. Gish and M. Schmidt, "Text-independent Speaker Identification," *IEEE Signal Processing Magazine*, vol. 11, no. 4, pp. 18–32, 1994.

[8] F. Bimbot, J.-F. Bonastre, C. Fredouille, G. Gravier, I. Magrin-Chagnolleau, S. Meignier, T. Merlin, J. Ortega-García, D. Petrovska-Delacrétaz, and D. A. Reynolds, "A tutorial on text-independent speaker verification," *EURASIP journal on applied signal processing*, pp. 430–451, 2004.

[9] J. Richiardi and A. Drygajlo, "Gaussian Mixture Models for On-line Signature Verification," in *Proceedings of the 2003 ACM SIGMM workshop on Biometrics methods and applications*, 2003, pp. 115–122.

[10] R. Jourani, K. Daoudi, R. André-Obrecht, and D. Aboutajdine, "Fast Training of Large Margin Diagonal Gaussian Mixture Models for Speaker Identification," in *Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, 2011, pp. 1–4.

[11] N. Kumar, S. Satoor, and I. Buck, "Fast Parallel Expectation Maximization for Gaussian Mixture Models on GPUs Using CUDA," in *IEEE International Conference on High Performance Computing and Communications*, 2009, pp. 103–109.

[12] B.-H. Juang, W. Hou, and C.-H. Lee, "Minimum Classification Error Rate Methods for Speech Recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 3, pp. 257–265, 1997.

[13] Z. Zivkovic, "Improved Adaptive gaussian Mixture Model for Background Subtraction," in *Proceedings of the 17th International Conference on Pattern Recognition*, vol. 2, 2004, pp. 28–31.

[14] W. Kwedlo, "A Parallel EM Algorithm for Gaussian Mixture Models Implemented on a NUMA System Using OpenMP," in *Euromicro International Conference on Parallel Distributed and Network-Based Processing (PDP)*, 2014, pp. 292–298.

[15] R. Yang, T. Xiong, T. Chen, Z. Huang, and S. Feng, "DISTRIM: Parallel GMM Learning On Multicore Cluster," in *IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, vol. 2, 2012, pp. 630–635.

[16] A. D. Pangborn, "Scalable Data Clustering Using GPUs," 2010.

[17] L. Machlica, J. Vaněk, and Z. Zajíc, "Fast Estimation of Gaussian Mixture Model Parameters on GPU Using Cuda," in *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2011, pp. 167–172.

[18] J. Chen, J. Cong, M. Yan, and Y. Zou, "FPGA-accelerated 3D Reconstruction Using Compressive Sensing," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2012, pp. 163–166.

[19] Y. Sun and L. A. Christopher, "3D Image Segmentation Implementation on FPGA Using the EM/MPM Algorithm," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2010, pp. 670–673.

[20] D. A. Reynolds and R. C. Rose, "Robust Text-independent Speaker Identification Using Gaussian Mixture Speaker Models," *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.

[21] D. A. Reynolds, "Speaker Identification and Verification Using Gaussian Mixture Speaker Models," *Speech Communication*, vol. 17, no. 1, pp. 91–108, 1995.

[22] R. Farnoosh and B. Zarpak, "Image Segmentation using Gaussian Mixture Model," *IUST International Journal of Engineering Science*, vol. 19, no. 1-2, pp. 29–32, 2008.

[23] N. Friedman and S. Russell, "Image Segmentation in Video Sequences: A Probabilistic Approach," in *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1997, pp. 175–181.

[24] K. Blekas, A. Likas, N. P. Galatsanos, and I. E. Lagaris, "A Spatially Constrained Mixture Model for Image Segmentation," *IEEE Transactions on Neural Networks*, vol. 16, no. 2, pp. 494–498, 2005.

[25] D. B. Thomas, "FPGA Gaussian Random Number Generators with Guaranteed Statistical Accuracy," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014, pp. 149–156.

[26] ——, "The Table-Hadamard GRNG: An Area-Efficient FPGA Gaussian Random Number Generator," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 4, p. 23, 2015.

[27] D.-U. Lee, R. C. Cheung, W. Luk, and J. D. Villasenor, "Hierarchical Segmentation for Hardware Function Evaluation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 103–116, 2009.

[28] J. Detrey and F. De Dinechin, "Table-based Polynomials for Fast Hardware Function Evaluation," in *IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)*, 2005, pp. 328–333.

[29] F. De Dinechin, M. Joldes, and B. Pasca, "Automatic Generation of Polynomial-based Hardware Architectures for Function Evaluation," in *International Conference on Application-specific Systems, Architectures and Processors*, 2010, pp. 216–222.

[30] S.-i. Amari, A. Cichocki, and H. H. Yang, "A New Learning Algorithm for Blind Signal Separation," *Advances in Neural Information Processing Systems*, pp. 757–763, 1996.

[31] M. Shi and A. Bermak, "An Efficient Digital VLSI Implementation of Gaussian Mixture Models-based Classifier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 9, pp. 962–974, 2006.

[32] H. Fu, L. Gan, R. G. Clapp, H. Ruan, O. Pell, O. Mencer, M. Flynn, X. Huang, and G. Yang, "Scaling Reverse Time Migration Performance Through Reconfigurable Dataflow Engines," *IEEE Micro*, vol. 34, no. 1, pp. 30–40, 2014.

[33] C. Biernacki, G. Celeux, and G. Govaert, "Choosing Starting Values for the EM Algorithm for Getting the Highest Likelihood in Multivariate Gaussian Mixture Models," *Computational Statistics & Data Analysis*, vol. 41, no. 3, pp. 561–575, 2003.

[34] V. Melnykov and I. Melnykov, "Initializing the eM algorithm in Gaussian Mixture Models with an Unknown Number of Components," *Computational Statistics & Data Analysis*, vol. 56, no. 6, pp. 1381–1395, 2012.

[35] A. D. Pangborn, "Expectation Maximization with a Gaussian Mixture Model using CUDA." [Online]. Available: https://github.com/Corv/CUDA-GMM-MultiGPU

[36] J. A. Bilmes, "A Gentle Tutorial of the EM Algorithm and Its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models," *International Computer Science Institute*, vol. 4, no. 510, p. 126, 1998.

[37] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization for Mobile Robots," *Artificial Intelligence*, vol. 128, no. 1, pp. 99–141, 2001.

Conghui He is a PhD candidate in the Department of Computer Science and Technology in Tsinghua University. His research interests include FPGA-based solutions to exploration geophysics and financial applications, focusing on algorithmic development and performance optimizations. Conghui has a BE in software engineering from Sun Yat-sen University.

Haohuan Fu is the deputy director of the National Supercomputing Center in Wuxi, and an associate professor in the Ministry of Education Key Laboratory for Earth System Modeling, and Department of Earth System Science in Tsinghua University. His research interests include design methodologies for highly efficient simulation applications, intelligent data Management, analysis, and data Mining platforms. Fu has a PhD in computing from Imperial College London. The related papers have won honors such as the 27 most Significant Papers of the FPL conference in 25 years, and the Gordon Bell Prize of SC16.

Ce Guo got his PhD degree from the Department of Computing at Imperial College London. He is currently a freelance consultant in predictive systems. His research interests include temporal data analytics, reconfigurable computing and symbolic artificial intelligence.

Wayne Luk is Professor of Computer Engineering and the Director of the EPSRC Cnetre for Doctoral Traning in High Performance Embedded and Distributed Systems. Imperial College London. His current research interests include theory and practice of customizing hardware and software for specific application domains such as genomics and climate modelling, and high-level compilation techniques and tools for highperformance computers and embedded systems. He is a fellow of the Royal Academy of Engineering, the IEEE and the BCS.

Guangwen Yang is a professor in the Department of Computer Science and Technology and the director of the Institute of High Performance Computing at Tsinghua University. His research interests include parallel algorithms, cloud computing, and the earth system model. Yang has a PhD in computer architecture from Harbin Institute of Technology.