A Fully-Pipelined Expectation-Maximization Engine for Gaussian Mixture Models

Ce Guo [#*1], Haohuan Fu [*2], Wayne Luk [#3]

[#] *Department of Computing, Imperial College London*
United Kingdom
[1] ce.guo10@imperial.ac.uk
[3] w.luk@imperial.ac.uk

[*] *Center for Earth System Science, Tsinghua University*
People's Republic of China
[2] haohuan@tsinghua.edu.cn

***Abstract*—Gaussian Mixture Models (GMMs) are powerful tools for probability density modeling and soft clustering. They are widely used in data mining, signal processing and computer vision. In many applications, we need to estimate the parameters of a GMM from data before working with it. This task can be handled by the Expectation-Maximization algorithm for Gaussian Mixture Models (EM-GMM), which is computationally demanding. In this paper we present our FPGA-based solution for the EM-GMM algorithm. We propose a pipeline-friendly EM-GMM algorithm, a variant of the original EM-GMM algorithm that can be converted to a fully-pipelined hardware architecture. To further improve the performance, we design a Gaussian probability density function evaluation unit that works with fixed-point arithmetic. In the experiments, our FPGA-based solution generates fairly accurate results while achieving a maximum of 517 times speedup over a CPU-based solution, and 28 times speedup over a GPU-based solution.**

I. Introduction

Gaussian Mixture Models (GMMs) are powerful tools for probability density modeling and soft clustering. They originally came from statistical machine learning and they play key roles in a large number of applications in data mining, signal processing and computer vision. For example, Reynold *et al.* [1] present a speaker verification system where GMMs are used to model the characters of a speaker's voice. Stauffer *et al.* [2] design a computer vision system to subtract the background from video streams where GMMs are used to judge whether a pixel belongs to the background in a probabilistic manner. Greenspan *et al.* [3] propose an image segmentation system to identify tissues in magnetic resonance (MR) images of the brain where GMMs are employed to capture the spatial layout information of brain tissues.

A GMM is a probability density model governed by a set of parameters. We need to estimate the parameters from data before working with the model in many applications. One popular parameter estimation solution is a particular adaptation of the Expectation-Maximization (EM) algorithm: EM for Gaussian Mixture models (EM-GMM) [4]. The EM-GMM algorithm 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.

The time spent on each run of the EM-GMM algorithm directly depends on the data size. 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. Moreover, it is sometimes desirable for the algorithm to be executed within a short period of time. This is especially true when the algorithm has to be invoked for multiple times (e.g. cross-validation [5] and boosting [6]) or when fast response is in great need (e.g. real-time object tracking [7]).

In this paper, we present our FPGA-based solution for the EM-GMM algorithm. We aim to provide a fast parameter estimation system that handles most of the computations in the EM-GMM algorithm in a fully-pipelined manner. Our major contributions are as follows:

- We restructure the work flow 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.
- We overcome the precision problem in Gaussian PDF evaluation with bit shifting and successfully deploy fixed-point arithmetic throughout our system.

The rest of this paper is organized as follows. Section II provides an introduction to GMMs and the EM-GMM algorithm, and discusses existing high performance computing solutions for the EM-GMM algorithm. Section III describes our restructured pipeline-friendly EM-GMM algorithm. Section IV presents our customized evaluation unit for Gaussian probability density functions. Section V shows some experimental results on accuracy and performance of our design compared with two CPU-based systems and one GPU-based system. Finally, Section VI provides a brief conclusion of this

978-1-4673-2845-6/12/$31.00 © 2012 IEEE

study and discusses possible future work.

II. Background

A. 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

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

where

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

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

  The probability density $\mathcal{G}(x_n|\mu_k, \Theta_k)$ 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 $\{\Theta_1 \ldots \Theta_K\}$.

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

$$\Theta_k = \mathrm{diag}(\sigma_k^2) \tag{4}$$

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

This assumption is widely taken by a variety of studies about GMMs such as [2], [8], [9], [10] and [11]. It simplifies the evaluation of the Gaussian probability density function (PDF) while keeping or even improving the accuracy and robustness.

B. 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 responsibility value r_{nk} for each data instance x_n with respect to each Gaussian component k using the current estimation of parameter values $\{w_1 \ldots w_K\}$, $\{\mu_1 \ldots \mu_K\}$ and $\{\Theta_1 \ldots \Theta_K\}$. More specifically, the responsibility value r_{nk} is defined and computed by

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

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

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

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

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

  where

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

The original EM algorithm for Gaussian mixture models is shown in Algorithm 1. Sufficient details are provided in the pseudocode to expose the patterns in memory access and computations. In Algorithm 1, Line 2 to Line 8 correspond to the expectation step; Line 9 to Line 19 correspond to the maximization step.

Fig 1 shows an example of the EM-GMM algorithm on a set of 2-dimensional data instances. The two ellipses stand for two Gaussian components respectively. In each expectation step, a data instance is plotted to be darker if it has larger responsibility values with respect to the Gaussian component represented with solid ellipse. In the example run, we follow the assumption that all covariance matrices are diagonal.

C. Existing Acceleration Systems for the EM-GMM

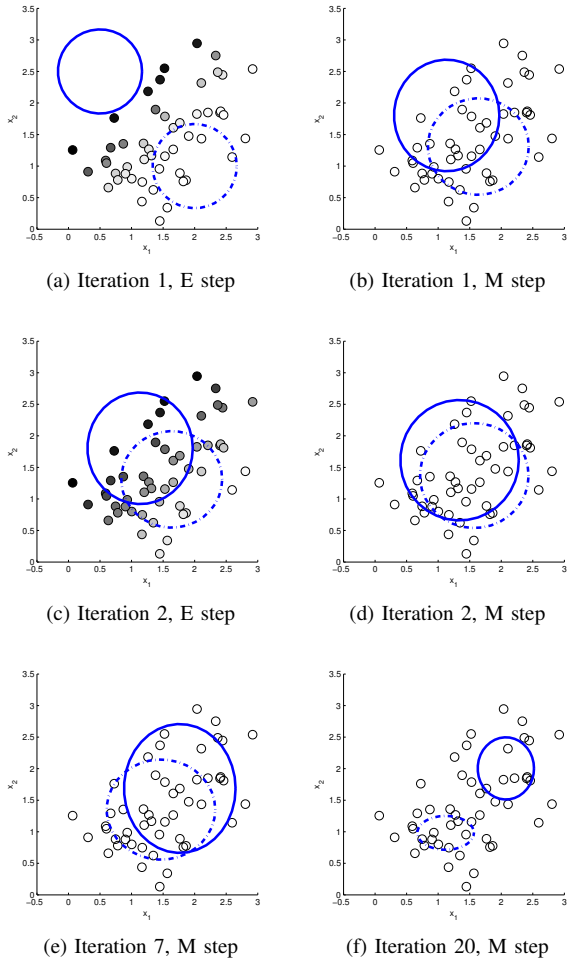
There are some existing acceleration solutions for the EM-GMM algorithm based on different platforms such as clusters and Graphics Processing Units (GPUs).

López-de-Teruel *et al.* [12] propose a programming framework for the parallel EM algorithm with standard Message Passing Interface (MPI) and they apply the framework to GMMs. Their implementation of EM-GMM is shown to have acceptable scalability as the number of processors increases.

Zhou *et al.* [13] present a distributed system for clustering data streams based on EM-GMM. They put their effort on reducing communication cost to eliminate performance bottlenecks. They also propose a series of optimizations to reduce

Algorithm 1 ORIGINAL EM-GMM

while stop condition not met **do**
  for $n \leftarrow 1$ to N **do**
    $s \leftarrow 0$
    for $k \leftarrow 1$ to K **do**
      $g_k \leftarrow w_k \mathcal{G}(x_n|\mu_k, \Theta_k)$
      $s \leftarrow s + g_k$
    for $k \leftarrow 1$ to K **do**
      $r_{nk} \leftarrow \frac{g_k}{s}$
  $N_k \leftarrow 0$, $w \leftarrow 0$, $\mu \leftarrow 0$, $\sigma \leftarrow 0$
  for all $n \in 1..N, k \in 1..K$ **do**
    $N_k \leftarrow N_k + r_{nk}$
  for all $k \in 1..K$ **do**
    $w_k \leftarrow \frac{N_k}{N}$
  for all $n \in 1..N, k \in 1..K, d \in 1..D$ **do**
    $\mu_{kd} \leftarrow \mu_{kd} + \frac{r_{nk} x_{nd}}{N_k}$
  for all $n \in 1..N, k \in 1..K, d \in 1..D$ **do**
    $\sigma^2_{kd} \leftarrow \sigma^2_{kd} + \frac{r_{nk}(x_{nd} - \mu_{kd})^2}{N_k}$
  for all $k \in 1..K$ **do**
    $\Theta_k \leftarrow \mathrm{diag}(\sigma^2_k)$



(a) Iteration 1, E step (b) Iteration 1, M step

(c) Iteration 2, E step (d) Iteration 2, M step

(e) Iteration 7, M step (f) Iteration 20, M step

Fig. 1: Example Run of EM-GMM

CPU time and memory consumption for the computing nodes in the system.

Gu *et al.* [14] consider the problem of building GMMs using data collected from distributed sensor networks. They propose a distributed EM-GMM algorithm for sensor networks in which the parameters of a GMM can be estimated locally at each network node in parallel without requiring the data to be collected and processed at the central location.

Kumar *et al.* [8] 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 the experiments, the implementation achieves a maximum of 164 times speedup compared to a naïve single-threaded C implementation on the CPU platform.

III. 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 pipelined-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.

A. Updating Mixture Weights and Mean Vectors

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

Before any data instance is processed, $\eta_1 \ldots \eta_K$ are initialized to zero. When the n-th data instance x_n is loaded, the responsibility values 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 $\eta_1 \ldots \eta_K$ by

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

When all the N data instance 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 $\rho_1 \ldots \rho_K$ to collect statistical information about the weighted sum of the data instances. Initially, we set all members in $\rho_1 \ldots \rho_K$ to zero vectors. When a data instance x arrives, we update the elements in $\rho_1 \ldots \rho_K$ by

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

When all the N data instance 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

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

B. 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

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

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

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

We first focus on the second term in Equation 18. The constant 2 and the variable μ_{kd}^+ can be moved out of the summation operation because they are not related to n

$$\frac{\sum_{n=1}^{N} 2 r_{nk} x_{nd} \mu_{kd}^+}{N_k} = 2\mu_{kd}^+ \frac{\sum_{n=1}^{N} r_{nk} x_{nd}}{N_k} \tag{19}$$

On the other hand, by Equation 7

$$\frac{\sum_{n=1}^{N} r_{nk} x_{nd}}{N_k} = \mu_{kd}^+ \tag{20}$$

Substitute Equation 20 into Equation 19

$$\frac{\sum_{n=1}^{N} 2 r_{nk} x_{nd} \mu_{kd}^+}{N_k} = 2\mu_{kd}^+ \mu_{kd}^+ = 2(\mu_{kd}^+)^2 \tag{21}$$

We then focus on the third term in Equation 18. The variable μ_{kd}^+ can be moved out of the summation operation as it is independent of n

$$\frac{\sum_{n=1}^{N} r_{nk} (\mu_{kd}^+)^2}{N_k} = (\mu_{kd}^+)^2 \frac{\sum_{n=1}^{N} r_{nk}}{N_k} \tag{22}$$

Substitute Equation 9 into Equation 22

$$\frac{\sum_{n=1}^{N} r_{nk} (\mu_{kd}^+)^2}{N_k} = (\mu_{kd}^+)^2 \frac{N_k}{N_k} = (\mu_{kd}^+)^2 \tag{23}$$

Substitute Equation 21 and 23 into Equation 18, we have

$$\sigma_{kd}^{2+} = \frac{\sum_{n=1}^{N} r_{nk} x_{nd}^2}{N_k} - 2(\mu_{kd}^+)^2 + (\mu_{kd}^+)^2 \tag{24}$$

$$= \frac{\sum_{n=1}^{N} r_{nk} x_{nd}^2}{N_k} - (\mu_{kd}^+)^2 \tag{25}$$

We consider the transformation in Equation 25 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, $\tau_1 \ldots \tau_K$, to collect statistical information about the first term in Equation 25 and compute the value of term at the end of the iteration.

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

$$\tau_k \leftarrow \tau_k + r_{nk} x_n^2 \tag{26}$$

When all the N data instance are processed, we have

$$\tau_k = \sum_{n=1}^{N} r_{nk} x_n^2 \tag{27}$$

By Equation 15, 25 and 27, the new variance vector can be computed by

$$\sigma_{kd}^{2+} = \frac{\tau_{kd}}{\eta_k} - \frac{\rho_{kd}^2}{\eta_k^2} = \frac{\tau_{kd}\eta_k - \rho_{kd}^2}{\eta_k^2} \tag{28}$$

C. Algorithm Summary

To summarize the algorithm transformations illustrated above, we present the pipelined-friendly EM-GMM algorithm as a piece of pseudocode in Algorithm 2.

Algorithm 2 PIPELINE-FRIENDLY EM-GMM

```
1:  while stop condition not met do
2:    for n ← 1 to N do
3:      ρ ← 0, τ ← 0
4:      s ← 0
5:      for k ← 1 to K do
6:        g_k ← w_k G(x_n|μ_k, Θ_k)
7:        s ← s + g_k
8:      for k ← 1 to K do
9:        r ← g_k / s
10:       η_k ← η_k + r
11:       for d ← 1 to D do
12:         ρ_kd ← ρ_kd + r x_nd
13:         τ_kd ← τ_kd + r x_nd^2
14:   for k ← 1 to K do
15:     w_k ← η_k / N
16:     for d ← 1 to D do
17:       μ_kd ← ρ_kd / η_k
18:       σ_kd^2 ← (τ_kd η_k − ρ_kd^2) / η_k^2
19:     Θ_k ← diag(σ_k^2)
```

Note that the algorithmic transformation is lossless. No numerical or algorithmic approximation were 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 data set and the same initial parameter sets, although practically the two results may be slightly different due to numerical precision issues. Such a

similarity suggests that the pipelined-friendly algorithm will have similar accuracy and convergence speed as the original one.

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

- The expectation step and the maximization step become overlapped in the pipeline-friendly algorithm. In the original algorithm, the maximization step does not start until all the data instances are processed in the expectation step. In the pipeline-friendly algorithm however, the data set is handled in a per-instance manner. Statistical information of the new parameter set is updated once the data instance arrives.
- The original algorithm stores all the responsibility values in the expectation step. The pipeline-friendly algorithm computes the responsibility values for a newly arrived data instance and update the statistical information of the new parameter set. In this case, it is not necessary for the algorithm to store all responsibility values. When the statistical information about a data instance is collected, the corresponding responsibility values can be discarded safely.

The differences suggest that most of the computations in the pipeline-friendly EM-GMM can be moved to FPGAs. In this study, we move the collection procedure of statistical evidence (Line 2 to Line 13 in Algorithm 2) 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 pipelined-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 amount of memory accesses. Related experimental results can be found in Section V.

IV. Customized Function Evaluation Unit for Gaussian PDF

We can observe from Algorithm 2 that the most complicated computation of the pipeline-friendly EM-GMM is the evaluation of the Gaussian PDF. We design a Gaussian probability density function evaluation unit that works with fixed-point arithmetic. We aim to achieve both low hardware cost and satisfactory accuracy in our design.

A. Approximation Strategy

Lee *et al.* [15] design and optimize function evaluation units for three elementary functions in fixed-point arithmetic. They also provide a series of valuable suggestions 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 the Gaussian PDF.

The basic method for computing Gaussian PDFs is shown in Equation 2. Direct evaluation of such a function in FPGAs is extremely expensive as the function includes complicated computations such as matrix determinants, square roots, matrix inverses, matrix multiplications and exponential functions.

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.

$$\mathcal{G}(x_n|\mu_k,\Theta_k) = \prod_{d=1}^{D} \mathcal{G}(x_{nd}|\mu_{kd},\sigma_{kd}^2) \tag{29}$$

$$= \prod_{d=1}^{D} (\frac{1}{\sigma_{kd}} \phi(\frac{x_{nd}-\mu_{kd}}{\sigma_{kd}})) \tag{30}$$

$$= \prod_{d=1}^{D} \frac{1}{\sigma_{kd}} \prod_{d=1}^{D} \phi(\frac{x_{nd}-\mu_{kd}}{\sigma_{kd}}) \tag{31}$$

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

$$\phi(u) = \frac{\exp\{-\frac{1}{2}u^2\}}{\sqrt{2\pi}} \tag{32}$$

In Equation 31, the products of the reciprocals of standard divisions do not change in an EM iteration. Therefore they can be computed in the host PC. However, the product of standard one-dimensional Gaussian PDF values have to be computed within FPGAs.

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 can merely approximate the positive part of $\phi(u)$. For all $u < 0$, we compute $\phi(-u)$ instead. Second, the function value is meaningful when the corresponding input value does not exceed 3. Therefore we only need to compute a relatively accurate value when $0 \leq u \leq 3$.

We propose a method to compute the approximate value of the standard Gaussian PDF based on polynomial functions. We expect that the approximate function $\hat{\phi}(x)$ to be equipped with the following properties:

- $\hat{\phi}(x)$ 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 statistical fidelity of the algorithm.
- The computations involved in $\hat{\phi}(u)$ should be as simple as possible. This is to reduce the consumption of logic resources on FPGAs.

To trade-off between the two properties, we choose to use the following piecewise linear function to approximate the

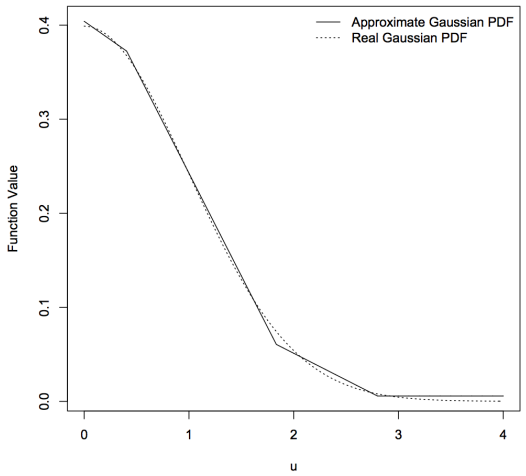


Fig. 2: Approximate one-dimensional Standard Gaussian PDF

standard one-dimensional Gaussian PDF

$$\hat{\phi}(u) = \begin{cases} -0.0781u + 0.4041, \text{if } 0 \leq u < 0.4053 \\ -0.2183u + 0.4609, \text{if } 0.4053 \leq u < 1.8340 \\ -0.0568u + 0.1647, \text{if } 1.8340 \leq u \leq 2.8 \\ \hat{\phi}(2.8), \text{if } u > 2.8 \end{cases} \tag{33}$$

The positive part of this approximate function is plotted with the real Gaussian PDF in Fig 2.

It can be observed from Fig 2 that the piecewise linear approximation fits the real Gaussian PDF well. We noted that inaccurate Gaussian PDF values may not lead to negative results in GMM-based systems if the error is properly controlled [16]. Therefore we consider our function approximation accurate enough as the absolute error is below 0.005 for most of the input values. Moreover, the function involves only three multiplication operations, which is inexpensive in terms of resource consumption on the FPGA platform.

B. Precision Control with Shifting

Consider the product of standard one-dimensional Gaussian PDF values in Equation 31

$$\mathcal{H}(x_n|\mu_k, \sigma_k) = \prod_{d=1}^{D} \phi(\frac{x_{nd} - \mu_{kd}}{\sigma_{kd}}) \tag{34}$$

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

$$p_{max}(D) = \max_{x_n} \mathcal{H}(x_n|\mu_k, \sigma_k) \tag{35}$$

$$= \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi}} \tag{36}$$

$$= (2\pi)^{-D/2} \tag{37}$$

The value of this 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-dimension Gaussian PDF evaluation to the left by h bits. The corresponding maximum value is

$$p'_{max}(D) = 2^{hD} \cdot (2\pi)^{-D/2} = 2^{(h-1/2)D} \cdot \pi^{-D/2} \tag{38}$$

Take logarithm on both sides with base 2, we have

$$\log_2 2^{(h-1/2)D} + \log_2 \pi^{-D/2} = \log_2 p'_{max}(D) \tag{39}$$

To control the maximum value of this function, we may set $p'_{max}(D)$ to any constant number. We propose to take $p'_{max}(D) = 1$ because this setting enables us to build a constant shifting scheme regardless of the value of D. More specifically, by taking $p'_{max}(D) = 1$, Equation 39 can be simplified as

$$(h - \frac{1}{2})D - \frac{D}{2} \log_2 \pi = 0 \tag{40}$$

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

$$h_0 = \frac{1}{2}(\log_2 \pi + 1) \tag{41}$$

As a result, if we can shift each one-dimensional Gaussian PDF evaluation result by $\frac{1}{2}(\log_2 \pi + 1)$ bits, we can control the value of $\mathcal{H}(x_n|\mu_k, \sigma_k)$ to be less than 1. However, we are not able to shift the result by $\frac{1}{2}(\log_2 \pi + 1)$ bits because $\frac{1}{2}(\log_2 \pi + 1) \approx 1.325748$ which is not an integer.

Note that $h_0 \approx \frac{4}{3}$, which means that shifting 4 bits in 3 evaluations could be an approximate solution. We consider to shift 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

$$h_t = \begin{cases} 2, \text{if } \mod(t, 3) = 1 \\ 1, \text{otherwise} \end{cases} \tag{42}$$

Taking this shifting scheme, the maximum value of the product is controlled to be less than 1.5 when $D \leq 20$. Note that it is safe to replace the product of standard one-dimensional Gaussian PDF values in Equation 31 by the shifted result without shifting back. This is because the responsibility values computed in Line 9 in Algorithm 2 will stay unchanged even if we take shifted probability values.

V. Experiments

We follow the strategy described in [8] to generate data for our experiments. More specifically, we use MATLAB to randomly sample data sets from real GMMs with random noise. We generate six data sets with $D = 3, 6$ and $K = 2, 4, 6$ respectively. 10^6 data instances are sampled for each data set.

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

be named 'CPU1', 'CPU2' and 'FPGA' for short in our experimental records.

In the experiments about performance, we compare the systems with a GPU implementation described in [8]. As we do not have the experimental results of this system on our data, the performance is estimated according to the results and trends described in [8]. As we are not using the same data sets with the ones in [8], for each of our data set, we take the best performance record in [8] with similar data size. Note that the performance estimation may not be very meaningful and we provide the results here only for reference. This configuration will be named 'GPU' for short in our experimental records.

The two CPU implementations are deployed in a PC with an Intel Core i3 CPU (running at 2.93GHz) and 4GB DDR3 memory. Both implementations are coded in the C programming language on a single CPU core and compiled with the highest compiler optimization in the Intel C compiler. The FPGA implementation is deployed in a Maxeler MAX3 acceleration card with a Xilinx Virtex-6 FPGA running at 150MHz and 48GB DDR3 on-board memory.

A. Accuracy Results

We describe the accuracy of a system by the average log-likelihood value [4] of the estimated parameters with respect to the data set. Larger log-likelihood values suggest better accuracy. Using the same data set and the 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 3. As the difference on the accuracy between the two CPU implementations are too small to be visible, they are plotted as a single curve in each plot.

B. Performance Results

We describe the performance 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. experimental results on performance are recorded in Table I. The last two columns in the table record the speedup values of our FPGA-based solution over the CPU-based solution (the original EM-GMM) and the GPU based solution respectively.

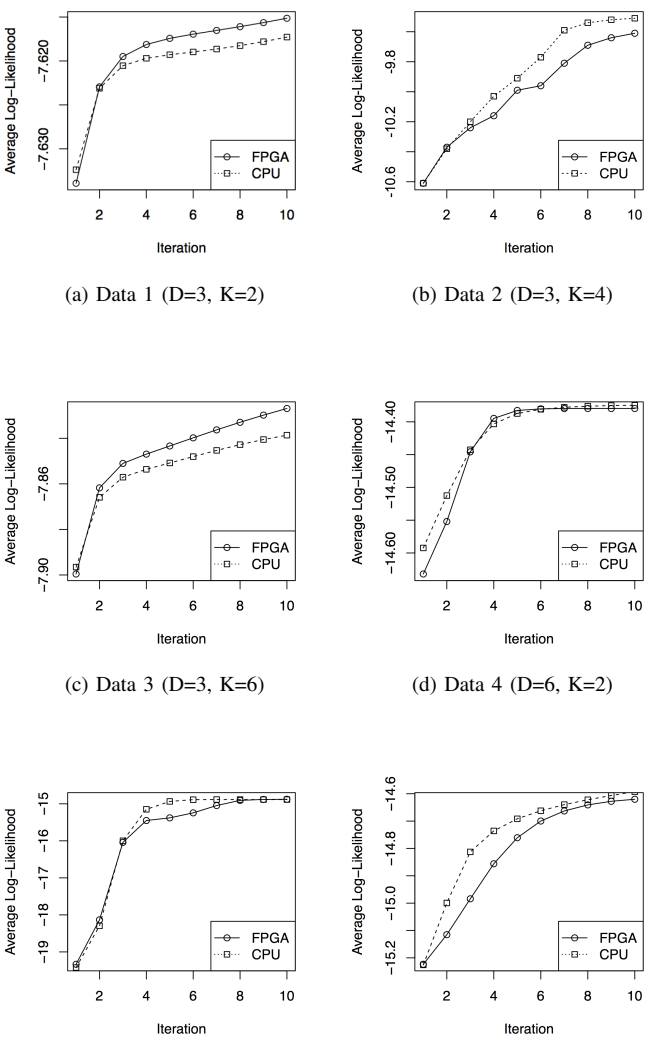
TABLE I: Performance Results (Instances per Second)

Data	CPU1	CPU2	GPU	FPGA	SU_{C1}	SU_G
1	$1.732e6$	$2.040e6$	$3.081e7$	$1.493e8$	86x	5x
2	$8.565e5$	$1.014e6$	$1.541e7$	$1.492e8$	174x	9x
3	$5.689e5$	$6.671e5$	$1.027e7$	$1.492e8$	262x	15x
4	$8.714e5$	$1.023e6$	$1.541e7$	$1.487e8$	171x	9x
5	$4.310e5$	$5.079e5$	$7.704e6$	$1.487e8$	346x	15x
6	$2.883e5$	$3.402e5$	$5.136e6$	$1.487e8$	517x	28x

C. Discussion

We can observe from Fig 3 that our FPGA-based solution 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 FPGA-based 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 values instead of zero, which may prevent the algorithm from premature convergence. Similar observations can be found in Monto Carlo localization in robotics [17].

Table I shows that our FPGA-based solution is significantly more efficient than other systems at least in our tested cases. In the best case we tested, the FPGA solution achieves 517 times speedup over a CPU-based solution (the original EM-GMM) and 28 times speedup over the GPU-based solution.



(a) Data 1 (D=3, K=2) (b) Data 2 (D=3, K=4)

(c) Data 3 (D=3, K=6) (d) Data 4 (D=6, K=2)

(e) Data 5 (D=6, K=4) (f) Data 6 (D=6, K=6)

Fig. 3: Accuracy Results (Average Log-Likelihood)

The throughput of the FPGA-based solution keeps at a constant level at around 1.5×10^8 instances per second. As the FPGA runs at 150MHz, the performance suggests that the system handles a data instance per cycle without being confronted with memory bottlenecks.

We believe the reason behind the high speedup of our FPGA-based solution is that the fixed-point arithmetic saves hardware resources on the FPGA platform. This enables us to deploy up to 36 Gaussian PDF evaluation units in the pipeline, which exploits available FPGA resources well. However, it is not feasible to perform similar optimization on CPUs and GPUs. If we have richer logical resources on the FPGA platform, we can deploy an even larger number of Gaussian PDF evaluation units to enable more complicated data with larger D and K to be processed by the system. The corresponding acceleration would be more significant.

Moreover, we can see from Table I that the pipeline-friendly EM-GMM algorithm leads to better performance than the original one on the CPU platform, as we predicted in Section III-C.

VI. Conclusion and Future Work

This paper presents our FPGA-based solution for the Expectation-Maximization for Gaussian Mixture Models (EM-GMM), which is a widely used but computationally demanding algorithm. We restructure the work flow 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 also design a Gaussian probability density function evaluation unit with fixed-point arithmetic in which the precision of internal results are under proper control. This unit is inexpensive in terms of resource consumption of the FPGA platform and provides reliable results.

In the experiments, the FPGA-based solution is shown to be able to provide accurate results with higher performance than all the tested solutions which are based on CPUs and GPUs. It achieves 517 times speedup over a CPU-based solution running the original EM-GMM, and 28 times speedup over a GPU-based solution.

Possible future work based on this study includes the following. First, we may generalize our design to handle non-diagonal covariance matrices. This may enhance the descriptive power of the GMMs generated by our system. Second, we may integrate our solution into specific applications such as object tracking, speech recognition and data visualization. Further optimization can be applied according to the requirements of the applications. Third, in addition to the EM-GMM algorithm, there are a variety of other important but computationally demanding machine learning algorithms based on EM iterations. Designing FPGA-based solutions for these algorithms could be very rewarding.

VII. Acknowledgement

The authors would like to thank the anonymous referees for their valuable suggestions. This research is supported in part by the UK EPSRC, by the European Union Seventh Framework Programme under Grant agreement number 248976, 257906 and 287804, by the HiPEAC NoE, by the Maxeler University Program, and by Xilinx. The first author is financially supported by the CSC-Imperial Scholarship co-founded by Imperial College London and China Scholarship Council.

References

[1] D. Reynolds, T. Quatieri, and R. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digital signal processing*, vol. 10, no. 1-3, pp. 19–41, 2000.

[2] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2. IEEE, 1999.

[3] 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.

[4] J. Bilmes, "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models," ICSI, Tech. Rep. TR-97-021, 1997.

[5] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence*, vol. 14, 1995, pp. 1137–1145.

[6] Y. Freund and R. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational learning theory*. Springer, 1995, pp. 23–37.

[7] H. Wang, W. Goh, C. Chua, and C. Sim, "Real-time object tracking," in *IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, vol. 2. IEEE, 1995, pp. 1366–1371.

[8] 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*. IEEE, 2009, pp. 103–109.

[9] B. Juang, W. Hou, and C. Lee, "Minimum classification error rate methods for speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 3, pp. 257–265, 1997.

[10] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *International Conference on Pattern Recognition*, vol. 2. IEEE, 2004, pp. 28–31.

[11] J. Richiardi and A. Drygajlo, "Gaussian mixture models for on-line signature verification," in *ACM SIGMM workshop on Biometrics methods and applications*. ACM, 2003, pp. 115–122.

[12] P. López de Teruel, J. García, and M. Acacio, "The parallel EM algorithm and its applications in computer vision," in *International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999, pp. 571–578.

[13] A. Zhou, F. Cao, Y. Yan, C. Sha, and X. He, "Distributed data stream clustering: A fast EM-based approach," in *IEEE International Conference on Data Engineering*. IEEE, 2007, pp. 736–745.

[14] D. Gu, "Distributed EM algorithm for gaussian mixtures in sensor networks," *IEEE Transactions on Neural Network*, vol. 19, no. 7, pp. 1154–1166, 2008.

[15] D. Lee, R. Cheng, W. Luk, and J. Villasenor, "Hierarchical segmentation for hardware function evaluation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 103–116, January 2009.

[16] 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.

[17] 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.