Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 1047–1052
DOI: 10.15439/2023F9859
ISSN 2300-5963 ACSIS, Vol. 35

Expectation-Maximization Algorithms for Gaussian Mixture Models Using Linear Algebra Libraries on Parallel Shared-Memory Systems

Wojciech Kwedlo
0000-0002-5040-2302
Faculty of Computer Science
Białystok University of Technology
Wiejska 45A, 15-351 Białystok, Poland,
w.kwedlo@pb.edu.pl

***Abstract*—In this paper the problem of parameter estimation of Gaussian mixture models using the expectation-maximization (EM) algorithm is considered. Four variants of the EM algorithm parallelized using the OpenMP standard are proposed. The main difference between the variants is the degree of usage of vendor-optimized linear algebra libraries. The computational experiments were performed using 25 large datasets on a system with two 12-core Intel Xeon processors. The results of experiments indicate that the EM variant using level 3 (matrix-matrix) operations and L3 cache blocking is the fastest one. It is 1.75–2.75 times faster than the naive version using level 2 (matrix-vector) operations. Its parallel efficiency relative to the sequential version is always greater than 83%.**

I. Introduction

FINITE mixture models [1] are a very versatile tool used for modeling complex probability distributions. Gaussian mixture models (GMMs) which assume multivariate normal density of a component, are arguably the most popular mixture models. GMMs have been successfully applied to many problems in engineering, finance, biology and data mining.

The maximum likelihood estimation (MLE), which seeks a maximum of the log-likelihood function, is a method of choice for GMM parameter estimation. The expectation-maximization (EM) algorithm [2] is the most common approach for MLE of GMM parameters. The algorithm is simple and easy to implement. Its important drawback is high computational complexity. The complexity of a single iteration is $O(NKd^2)$, where N is the number of data items, K is the number of mixture components, and d is the dimension of a feature space. These high computational requirements limit the usability of the EM, especially when d is large.

The problem of high computational requirements can be tackled a by parallel realization of the EM for GMMs (e.g., [3]). The importance of parallel formulations of the EM stems from ubiquity of relatively cheap multi-core processors. However, these processors have complex structures with multiple SIMD execution units and two- or three-level hierarchy of cache memory. This complexity makes an efficient implementation of the EM a tedious task. The difficulties in an efficient implementation can be alleviated by using a vendor-optimized matrix algebra libraries, for instance based on the BLAS standard [4].

This paper proposes four such parallel formulations, two of which use level 2 BLAS calls, and the remaining two leverage more efficient level 3 BLAS operations. The proposed algorithms are parallelized using the OpenMP standard, implemented in C++, and employ the Eigen template library[1] which seamlessly invokes the BLAS calls. We also investigate the use of blocking [5] for L3 cache. The computational experiments indicate that this optimization significantly improves the performance of the EM variant based on level 3 BLAS calls.

II. GMM parameter estimation

A finite mixture model with K components has the probability density function given by:

$$f(\mathbf{x}|\mathbf{\Theta}) = \sum_{m=1}^{K} \alpha_m \phi(\mathbf{x};\boldsymbol{\theta}_m), \tag{1}$$

where $\phi(\mathbf{x};\boldsymbol{\theta}_m)$ is the probability density function of the m-th component parameterized on $\boldsymbol{\theta}_m$, and $\alpha_1,\ldots,\alpha_K$ are the mixing proportions which must satisfy the following two conditions: $\alpha_1+\cdots+\alpha_K=1$ and $\alpha_m \geq 0$ for $m=1,\ldots,K$. $\mathbf{\Theta}=\{\alpha_1,\ldots,\alpha_k,\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_K\}$ is the complete set of parameters defining the mixture.

In Gaussian mixture models each component has the following (multivariate normal) probability density function:

$$\phi(\mathbf{x};\boldsymbol{\theta}_m) = \mathcal{N}(\mathbf{x};\boldsymbol{\mu}_m,\mathbf{\Sigma}_m) = \frac{1}{(2\pi)^{d/2}\det(\mathbf{\Sigma_m})^{1/2}} e^{\left[-0.5(\mathbf{x}-\boldsymbol{\mu_m})\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu_m})^T\right]}, \tag{2}$$

with the set of parameters $\boldsymbol{\theta}_m = [\boldsymbol{\mu}_m,\mathbf{\Sigma}_m]$, where d is the dimension of the feature space, $\boldsymbol{\mu}_m \in \mathbb{R}^d$ is the mean

This work was supported by the grant WZ/WI-IIT/4/2023 from Białystok University of Technology. Computations were carried out using the computers of Centre of Informatics Tricity Academic Supercomputer & Network, in Gdansk, Poland.

[1] http://eigen.tuxfamily.org

and $\mathbf{\Sigma_m}$ is the $d \times d$ covariance matrix. Thus, for a GMM the complete set of mixture parameters is given by $\mathbf{\Theta} = \{\alpha_1, \ldots, \alpha_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_K\}$.

Given a set of N independent and identically distributed feature vectors $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, where $\mathbf{x}_i \in \mathbb{R}^d$, the log-likelihood function, corresponding to a K-component mixture is given by:

$$\log f(X|\mathbf{\Theta}) = \sum_{i=1}^{N} \log \sum_{m=1}^{K} \alpha_m \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m, \mathbf{\Sigma}_m). \tag{3}$$

The maximum likelihood estimate of parameters is obtained as: $\mathbf{\Theta}^* = \underset{\mathbf{\Theta}}{\operatorname{argmax}} \log f(X|\mathbf{\Theta})$.

The EM algorithm [2], [6] is an iterative procedure which, given an initial estimate of parameters $\mathbf{\Theta}^{(0)}$, produces a sequence of estimates with increasing log-likelihood (3). j-th iteration of the algorithm consists of two steps called expectation step (E-step) and maximization step (M-step).

In the E-step [7], using the parameters $\Theta = \{\alpha_1, \ldots, \alpha_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_K\}$ from the previous iteration, for each feature vector $\mathbf{x}_i$, $i = 1, \ldots, N$ and for each mixture component m, $m = 1, \ldots, K$ the posterior probability that $\mathbf{x}_i$ was generated from m-th component is calculated as:

$$P(m|\mathbf{x}_i) = \frac{\alpha_m \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_m, \mathbf{\Sigma}_m)}{\sum_{k=1}^{K} \alpha_k \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)}. \tag{4}$$

The M-step [7], using the posterior probabilities $P(m|\mathbf{x}_i)$, computes new estimate of parameters $\mathbf{\Theta}$ as ($m = 1, \ldots, K$):

$$\alpha_m = \frac{1}{N} \sum_{i=1}^{N} P(m|\mathbf{x}_i), \tag{5}$$

$$\boldsymbol{\mu}_m = \frac{\sum_{i=1}^{N} P(m|\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{N} P(m|\mathbf{x}_i)}, \tag{6}$$

$$\mathbf{\Sigma}_m = \frac{\sum_{i=1}^{N} P(m|\mathbf{x}_i)(\mathbf{x}_i - \boldsymbol{\mu}_m)^T(\mathbf{x}_i - \boldsymbol{\mu}_m)}{\sum_{i=1}^{N} P(m|\mathbf{x}_i)}. \tag{7}$$

The E-step and M-step are applied alternately until a convergence criterion is met.

III. Four formulations of the EM using matrix algebra libraries

All formulations of the EM algorithm for GMMs discussed in this section store the training set in a matrix (two-dimensional array in the C++ language) $\mathbf{X} = [x_{i,j}]_{1 \le i \le N, 1 \le j \le d}$, where i-th row, denoted by $x_{i,*}$ stores the feature vector $\mathbf{x}_i$. Similarly, the posterior probabilities are stored in a matrix $\mathbf{P} = [p_{i,j}]_{1 \le i \le N, 1 \le j \le K}$, where $p_{i,j} = P(m|\mathbf{x}_i)$.

Algorithm 1 shows a high-level overview of the EM. The equations (3) and (4) indicate that in order to perform both the convergence check and the E-step we need to compute Gaussian probability density function values multiplied by the corresponding mixing proportions. An obvious optimization is to compute densities weighted by mixing proportions once, store them in a matrix $\mathbf{W} = [w_{i,j}]_{1 \le i \le N, 1 \le j \le K}$, where $w_{i,j} = \alpha_j * \mathcal{N}(x_{i,*}; \boldsymbol{\mu}_j, \mathbf{\Sigma}_j)$ and use them in subsequent E-Step and computation of log-likelihood.

The four variants of the EM discussed in the paper follow this pattern. After the computation of weighted densities $\mathbf{W}$ (line 3 of Algorithm 1), the log-likelihood using (3) is computed (line 4). If the algorithm is not terminated in line 6, then the posterior probability matrix $\mathbf{P}$ using weighted densities $\mathbf{W}$ is obtained by equation (4) (line 8). The computation of the log-likelihood L and the matrix $\mathbf{P}$ based on $\mathbf{W}$ are very straightforward. We have implemented them using Eigen C++ library, which generates an efficient vectorized code. The

Algorithm 1 The pseudocode of the EM algorithm

```
Require: X, Θ^0, M, ε
 1: Θ ← Θ^0
 2: for i ← 1 to M do
 3:    W ← WeightedDensities(X, Θ)
 4:    L' ← L, L ← Loglikelihood(W)
 5:    if i > 1 and ConvergenceCheck(L, L') then
 6:       Terminate the algorithm
 7:    end if
 8:    P ← EStep(W)
 9:    Θ ← MStep(X, P)
10: end for
11: return Θ
```

iterations of the EM algorithm are performed until either the algorithm converges or the maximal number of iterations M is reached.

The four variants of the EM algorithm differ in implementation of `WeightedDensities` and `MStep` functions.

A. Variant I: EM-L2

This variant uses BLAS Level 2 (matrix-vector) calls in implementation of `WeightedDensities` and `MStep`, hence its name EM-L2. The pseudocode of `WeightedDensities` function is shown in Algorithm 2. The function starts (lines 1–3) with the computation of invariants which do not depend on the feature vector. The inverse and determinant of covariance matrices are calculated (line 2) using the Cholesky decomposition [8]. Next, the loop (lines 4–10) iterating over all rows of $\mathbf{X}$ and $\mathbf{W}$ is executed. In this loop, for each mixture component j a squared Mahalanobis distance between the i-th row $x_{i,*}$ and mean vector $\boldsymbol{\mu}_j$ is calculated in lines 6–7. The computations in line 7 are done using `cblas_dsymv` and `cblas_dot` calls [4]. Next (line 8) weighted normal density is calculated.

Algorithm 3 shows the pseudocode of `MStep` function. The function calculates sums in equations (5)–(7) in two passes over rows of matrices $\mathbf{X}$ and $\mathbf{P}$. In the first pass (lines 4–8),

Algorithm 2 `WeightedDensities` function in EM-L2

Require: $\mathbf{X}$, $\mathbf{\Theta}$
1: **for** $j \leftarrow 1$ **to** K **do**
2: $\quad \mathbf{S}_j \leftarrow {\mathbf{\Sigma_j}}^{-1}$, $b_j \leftarrow 1/\left((2\pi)^{d/2}\det(\mathbf{\Sigma_j})^{1/2}\right)$
3: **end for**
4: **for** $i \leftarrow 1$ **to** N **do**
5: $\quad$ **for** $j \leftarrow 1$ **to** K **do**
6: $\quad\quad \mathbf{y} \leftarrow x_{i,*} - \boldsymbol{\mu}_j$
7: $\quad\quad w_{i,j} \leftarrow -0.5\mathbf{y}S\mathbf{y}^T$
8: $\quad\quad w_{i,j} \leftarrow \alpha_j b_j \exp(w_{i,j})$
9: $\quad$ **end for**
10: **end for**
11: **return** $\mathbf{W} = [w_{i,j}]$

Algorithm 3 `MStep` function in EM-L2

Require: $\mathbf{X}$, $\mathbf{P}$
1: **for** $j \leftarrow 1$ **to** K **do**
2: $\quad \mathbf{\Sigma}_j \leftarrow \mathbf{0}$, $\boldsymbol{\mu}_j \leftarrow \mathbf{0}$, $s_j \leftarrow 0$
3: **end for**
4: **for** $i \leftarrow 1$ **to** N **do**
5: $\quad$ **for** $j \leftarrow 1$ **to** K **do**
6: $\quad\quad s_j \leftarrow s_j + p_{i,j}$, $\boldsymbol{\mu}_j \leftarrow \boldsymbol{\mu}_j + p_{i,j}x_{i,*}$
7: $\quad$ **end for**
8: **end for**
9: **for** $j \leftarrow 1$ **to** K **do**
10: $\quad \boldsymbol{\mu}_j \leftarrow \boldsymbol{\mu}_j/s_j$, $\alpha_j \leftarrow s_j/N$
11: **end for**
12: **for** $i \leftarrow 1$ **to** N **do**
13: $\quad$ **for** $j \leftarrow 1$ **to** K **do**
14: $\quad\quad \mathbf{y} \leftarrow x_{i,*} - \boldsymbol{\mu}_j$
15: $\quad\quad \mathbf{\Sigma_j} \leftarrow \mathbf{\Sigma_j} + p_{i,j}\mathbf{y}^t\mathbf{y}$
16: $\quad$ **end for**
17: **end for**,
18: **for** $j \leftarrow 1$ **to** K **do**
19: $\quad \mathbf{\Sigma_j} \leftarrow \mathbf{\Sigma_j}/s_j$
20: **end for**
21: **return** $\mathbf{\Theta} = \{\alpha_1, \ldots, \alpha_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_K\}$

for each component j, the sum of posterior probabilities s_j and the sums in the numerators of (6) are accumulated. Next, the final values of mixing proportions α_j and mean vectors $\boldsymbol{\mu}_j$ are obtained (lines 9–11). In the second pass (lines 12–17), using the mean vectors computed in the first pass, for each component j, the sum in the numerator of (7) is obtained. The computation in line 15 is performed using `cblas_dsyr` call [4], which calculates rank-1 update of a symmetric matrix. Finally, in lines 18–20, the covariance matrices are obtained from accumulated numerators of (7).

B. Variant II: EM-L2-reordered

Our initial experiments with EM-L2 indicated the abysmal performance, where both dimension of the feature space d and the number of mixture components K are high. However, a simple interchange of loops in lines 12–17 of Algorithm 3, which places the loop iterating over the mixture components first was able to significantly improve the performance. We call this variant EM-L2-reordered. It used the same formulation of `WeightedDensities` function as EM-L2.

C. Variant III: EM-L3-blocking

This variant uses Level 3 BLAS operations [4], which usually have have $O(n^2)$ memory complexity and much larger $O(n^3)$ computational complexity, which allows for higher reuse of data and higher level of optimization [5]. Additionally it uses employs blocking (called also loop tiling [5]) to further optimize the most time-critical operations of `WeightedDensities` and `MStep` functions. We apply this technique to process the data in smaller blocks that are more likely to fit in the last level of cache memory. The `WeightedDensities` and `MStep` functions are shown in Algorithms 4 and 5, respectively. These functions require additional parameter, which is the number of blocks. This parameter is denoted by β in `WeightedDensities` function and by γ in the `MStep` function.

In the `WeightedDensities` function the computation of squared Mahalanobis distance is performed for blocks of rows of the data matrix $\mathbf{X}$. To simplify description, we assume that the number of feature vectors N is divisible without remainder by the number of blocks β. In such case the size of each block equals $B = N/\beta$. The outermost loop (line 5) iterates over blocks. It starts by the computation of the indices of the first (i_s) and the last (i_e) row in current l-th blocks. These must satisfy the condition $i_e - i_s + 1 = N/\beta$. The inner loop (lines 7–13) is performed for submatrix of $\mathbf{X}$ consisting of rows $i_s, i_{s+1}, \ldots, i_e$, which we denote as $\mathbf{X}_{i_s:i_e,*}$. All the matrices involved in the inner loop nest including the submatrices of $\mathbf{X}$ and $\mathbf{W}$ have B rows. Since the computations in lines 8–12 are repeated K times, this approach allows for much greater reuse of data in the cache memories.

The code in lines 7-13 computes the squared Mahalanobis distances and stores them in a block of the matrix $\mathbf{W}$. Using the Cholesky decomposition of Σ_j^{-1} the squared distance between a feature vector in i-th row of X and j-th mixture component can be written as:

$$(x_{i,*}-\boldsymbol{\mu}_j)\mathbf{\Sigma}_j^{-1}(x_{i,*}-\boldsymbol{\mu}_j)^T = (x_{i,*}-\boldsymbol{\mu}_j)\mathbf{L}_j\mathbf{L}_j^T(x_{i,*}-\boldsymbol{\mu}_j)^T = \left[(x_{i,*}-\boldsymbol{\mu}_j)\mathbf{L}_j\right]\left[(x_{i,*}-\boldsymbol{\mu}_j)\mathbf{L}_j\right]^T. \quad (8)$$

Lines 8–12 implement this computation efficiently using two temporary matrices. The $B \times d$ temporary matrix $\mathbf{Y}$ is obtained by subtracting $\boldsymbol{\mu}_j$ from each row of the block of $\mathbf{X}$. The temporary $B \times d$ matrix $\mathbf{Z}$, where i-th row is given by $z_{i,*} = (x_{i,*} - \boldsymbol{\mu}_j)\mathbf{L}_j$ is computed in line 7 using `cblas_dtrmm` (Level 3) BLAS call [4], which multiplies a general matrix by a triangular matrix.

The pseudocode of `MStep` function is shown in Algorithm 5. The less time-consuming ($O(NKd)$ computational complexity) computation of mean vectors $\boldsymbol{\mu}_j$ and posterior sums s_j (lines 4–11) is performed similarly to EM-L2 version shown

Algorithm 4 `WeightedDensities` function in EM-L3-blocking

Require: $\mathbf{X}$, $\mathbf{\Theta}$, β
1: **for** $j \leftarrow 1$ **to** K **do**
2: $\quad \mathbf{S}_j \leftarrow {\mathbf{\Sigma_j}}^{-1}$, $\mathbf{L}_j \leftarrow \mathtt{chol}(\mathbf{S}_j)$
3: $\quad b_j \leftarrow 1/\left((2\pi)^{d/2} \det(\mathbf{\Sigma_m})^{1/2}\right)$
4: **end for**
5: **for** $l \leftarrow 1$ **to** β **do**
6: $\quad i_s, i_e \leftarrow \mathtt{BlockIndices}(N, \beta, l)$
7: $\quad$ **for** $j \leftarrow 1$ **to** K **do**
8: $\quad\quad \mathbf{Y} \leftarrow \mathbf{X}_{i_s:i_e,*} - \boldsymbol{\mu}_j$
9: $\quad\quad \mathbf{Z} \leftarrow \mathbf{Y}\mathbf{L}_j$
10: $\quad\quad$ **for** $i \leftarrow i_s$ **to** i_e **do**
11: $\quad\quad\quad w_{i,j} \leftarrow \sum_{k=1}^{d} z^2_{i-i_s+1,k}$
12: $\quad\quad\quad w_{i,j} \leftarrow b_j \alpha_j * \exp(-0.5 * w_{i,j})$
13: $\quad\quad$ **end for**
14: $\quad$ **end for**
15: **end for**
16: **return** $\mathbf{W} = [w_{i,j}]$

Algorithm 5 `MStep` function in EM-L3-blocking

Require: $\mathbf{X}$, $\mathbf{P}$, γ
1: **for** $j \leftarrow 1$ **to** K **do**
2: $\quad \mathbf{\Sigma}_j \leftarrow \mathbf{0}$, $\boldsymbol{\mu}_j \leftarrow \mathbf{0}$, $s_j \leftarrow 0$
3: **end for**
4: **for** $j \leftarrow 1$ **to** K **do**
5: $\quad$ **for** $i \leftarrow 1$ **to** N **do**
6: $\quad\quad s_j \leftarrow s_j + p_{i,j}$, $\boldsymbol{\mu}_j \leftarrow \boldsymbol{\mu}_j + p_{i,j} x_{i,*}$
7: $\quad$ **end for**
8: **end for**
9: **for** $j \leftarrow 1$ **to** K **do**
10: $\quad \boldsymbol{\mu}_j \leftarrow \boldsymbol{\mu}_j / s_j$, $\alpha_j \leftarrow s_j / N$
11: **end for**
12: **for** $l \leftarrow 1$ **to** γ **do**
13: $\quad i_s, i_e \leftarrow \mathtt{BlockIndices}(N, \gamma, l)$
14: $\quad$ **for** $j \leftarrow 1$ **to** K **do**
15: $\quad\quad$ **for** $i \leftarrow i_s$ **to** i_e **do**
16: $\quad\quad\quad y_{i-i_s+1,*} = (x_{i,*} - \boldsymbol{\mu}_j) * \sqrt{p_{i,j}}$
17: $\quad\quad$ **end for**
18: $\quad\quad \mathbf{\Sigma}_j \leftarrow \mathbf{\Sigma}_j + \mathbf{Y}\mathbf{Y}^T$
19: $\quad$ **end for**
20: **end for**
21: **for** $j \leftarrow 1$ **to** K **do**
22: $\quad \mathbf{\Sigma_j} \leftarrow \mathbf{\Sigma_j} / s_j$
23: **end for**
24: **return** $\mathbf{\Theta} = \{\alpha_1, \ldots, \alpha_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_K\}$

in Algorithm 3. The only change is the reordering of loops starting in lines 4–5.

However, sums in numerators of (7) are obtained (lines 12–17) in a completely different way. Similarly to Algorithm 4, the data are processed in γ blocks, with size of block equal $C = N/\gamma$. After a calculation of temporary $C \times d$ matrix $\mathbf{Y}$ in lines 12–15, the numerator of (7) is updated by a single `cblas_dsyrk` level 3 BLAS call [4].

The `WeightedDensities` and `MStep` functions of the EM-L3-blocking variant were designed to delegate the most time-consuming code fragments with $O(Nd^2)$ complexity to Level 3 BLAS calls. `MStep` requires additional $O(N)$ square root calculations (line 14 of Algorithm 5) per single covariance matrix.

The method for choosing numbers of blocks β and γ remains to be described. Denote by L the total capacity of last level cache in bytes. In our implementation, to conserve memory, we store $\mathbf{X}$ using single precision floating point numbers. $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{W}$, $\mathbf{P}$ are stored using double precision numbers. Taking into consideration that a single precision number needs 4 bytes of storage and a double precision 8 bytes, the loop in lines 7–13 of Algorithm 4 needs $4dB$ bytes for storage of $\mathbf{X}_{i_s:i_e,*}$, $8dB$ bytes for $\mathbf{Y}$ and $\mathbf{Z}$ and $8KB$ bytes for the submatrix of $\mathbf{W}$. Assuming, that the total working set in the loop should be equal to L bytes, we have:

$$\beta = \frac{(20 * d + 8K)N}{L}. \tag{9}$$

After performing a similar analysis of the loop in lines 14–19 of Algorithm 5 we get:

$$\gamma = \frac{(12 * d + 8K)N}{L}. \tag{10}$$

D. Variant IV: EM-L3

This variant is a simplification of EM-L3-blocking, which does not perform loop tiling, i.e., sets the number of blocks $\beta = 1$ and $\gamma = 1$. We implemented this variant in order to assess the influence of blocking on the performance of the variant III.

IV. Parallelization for shared-memory systems

All the EM variants described in the previous section can be parallelized using data decomposition approach. We have designed parallel formulation of the algorithms and implemented them using the OpenMP standard [9] for shared-memory architectures.

An OpenMP application can be viewed as a group of cooperating threads. At the begin, only a master thread executes. When this thread encounters the `#pragma omp parallel` directive, the execution of the following block of code is performed by a team of threads. When a team of threads encounters the `#pragma omp for` directive, a succeeding `for` loop is parallelized by the team of threads. In this case each thread executes a subset of the loop iterations. In our approach we use static loop scheduling, where each of t threads is assigned approximately n/t iterations, when n is the total number of loop iterations.

In the parallelization of the `WeightedDensities` and `EStep` functions we use the fact that the rows of output matrices ($\mathbf{W}$ and $\mathbf{\Theta}$, respectively) can be computed using the corresponding rows of the input matrices ($\mathbf{X}$ and $\mathbf{W}$, respectively) without the knowledge about the remaining rows. We employ the data decomposition of the matrices $\mathbf{X}$, $\mathbf{W}$, $\mathbf{P}$ in which each thread is responsible for a block of consecutive rows. In case of EM-L3-blocking, where data are processed in blocks we divide further each of β or γ blocks into t sub-blocks.

Similar decomposition scheme is applied to the parallelization of `Loglikelihood` function, where each thread computes the local sum (3) using its assigned block of rows. Next, the local sums computed by the team of threads are added up giving the final log-likelihood. This is an example of the reduction operation which, for a single variable, can be easily carried-out using the OpenMP `reduction` clause.

The above decomposition scheme can be easily applied to `WeightedDensities` in Algorithm 2, by placing a `#pragma omp parallel for` directive before loops starting in lines 1, 5, and 12. For the `WeightedDensities` function shown in Algorithm 4 we use `#pragma omp for` before lines 1 and 10, and manually divide the rows of matrices $\mathbf{Y}$ and $\mathbf{Z}$ (lines 8–9) into t threads of the OpenMP team.

The parallelization of `MStep` functions is also based on data decomposition. Additionally, we have to tackle the problem of computing the s_j and the sums in numerators of (5), (6) and (7). We use a similar approach to that in the `Loglikelihood` function. Each OpenMP thread calculates local sums which are added up using the reduction operation. Since OpenMP 4.5 does not provide reduction operation for user-defined datatypes we have used the binary tree reduction algorithm [10].

V. Experimental results

All the results reported in this section were obtained using single compute nodes of the Tryton cluster installed in Centre of Informatics Tricity Academic Supercomputer and Network in Gdansk, Poland. A single node of the cluster is equipped with two 12-core Intel Xeon E5-2670 v3 (2.3 GHz) CPUs and 128 GiB of DDR4 RAM. The programs were compiled using the Intel C/C++ compiler (icpc) version 2021.7.1 and linked with the Intel MKL library version 2022.2.1, which provided the BLAS calls. We run the sequential version of EM-L3-blocking on a single core, assuming in (9) and (10) the last level cache size $L = 30 * 2^{20}$ bytes, according to manufacturer specification of the processor. We run parallel versions of all four algorithms using all 24 cores and assuming the last level cache size of multiplied by two: $L = 60 * 2^{20}$ bytes, because two processors were used in the calculations.

The experiments were performed on synthetic datasets obtained by the MixSim simulator proposed in [11]. The experiments were executed as follows. First we chose $d \in \{20, 40, 60, 80, 100\}$ and $K \in \{20, 40, 60, 80, 100\}$. For each of 25 combinations of K and d we generated a single dataset using the MixSim simulator. The number of feature vectors N was chosen to set the total size of the dataset as close of 512MiB ($512 * 2^{20}$ bytes) as possible. Thus, all the datasets were much larger as the total size of last level cache memory in a compute node. For each dataset we generated a single initial solution of the EM algorithm. This solution was used to initialize all the variants of the EM algorithm. Because all the variants started from the same solution, they converged after the same number of iterations. We obtained the average iteration time by dividing the total execution time measured using a system high-precision real time clock by the number of iterations. The shorter average EM iteration time indicated the higher performance of the algorithm.

The results indicated that EM-L3-blocking is the fastest of all parallel algorithms in all the experiments. Due to space limitations we have to omit the presentation of average iteration times in a table. These times ranged from 1.13 second (EM-L3-blocking, $d = 20$, $K = 20$) to 150 seconds (EM-L2, $d = 100$, $K = 100$). Using these measurements we have calculated the algorithmic speedup and parallel efficiency of the EM-L3-blocking. Figure 1 shows the algorithmic speedup of the EM-L3-blocking variant over the EM-L3. For a given dataset. an algorithmic speedup S of EM-L3-blocking over another variant A is defined as: $S = t_{\mathrm{A}}/t_{\mathrm{EM-L3-blocking}}$, where t_{A} and $t_{\mathrm{EM-L3-blocking}}$ denote average iteration times of EM variant A and EM-L3-blocking, respectively. The figure

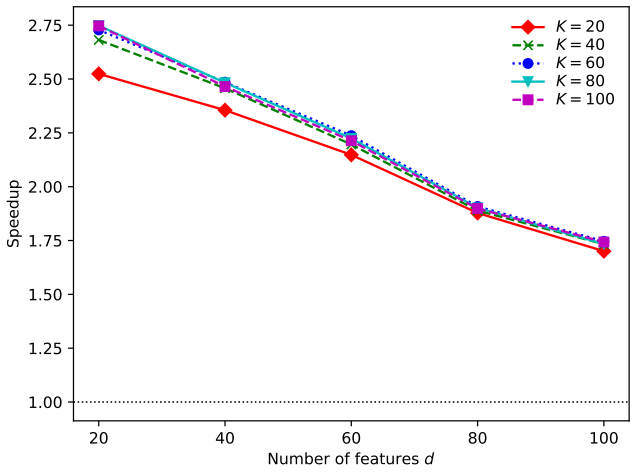

Fig. 1. Algorithmic speedup of the EM-L3-blocking variant over EM-L3 variant. The dotted horizontal line indicates equal speed of both algorithms.

indicates that EM-L3-blocking is faster than EM-L3 for all datasets and its advantage is increased with decreased feature space dimension d.

Figure 2 shows the algorithmic speedup of the EM-L3-blocking over the faster of two variants (EM-L2 and EM-L2-reordered) using level 2 BLAS operations. The plots indicate that EM-L3-blocking is always faster and its advantage is increased with the increase of the dimension d.

We end the presentation of the results by showing the parallel efficiency of EM-L3-blocking with respect to its sequential version. A parallel efficiency (in percent) is defined as the ratio of measured parallel speedup to the ideal linear

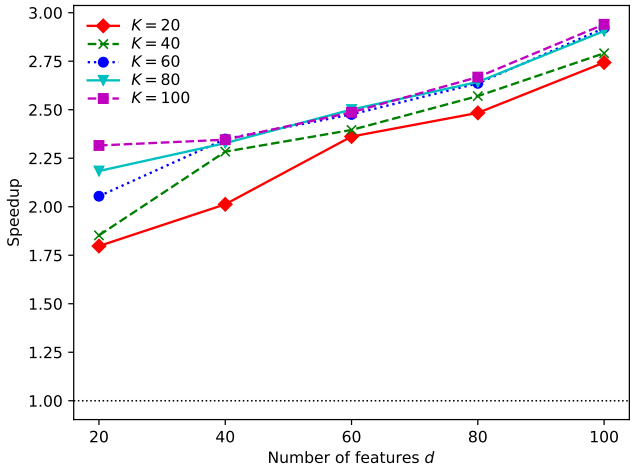

Fig. 2. Algorithmic speedup of the EM-L3-blocking variant over the fastest variant using L2 BLAS operations. The dotted horizontal line indicates equal speed of both algorithms.

speedup (equal to the number of the cores in a compute node). In turn, the parallel speedup is given by the ratio of the iteration time of the sequential version to iteration time of the parallel version of the algorithm. The plots indicate that the

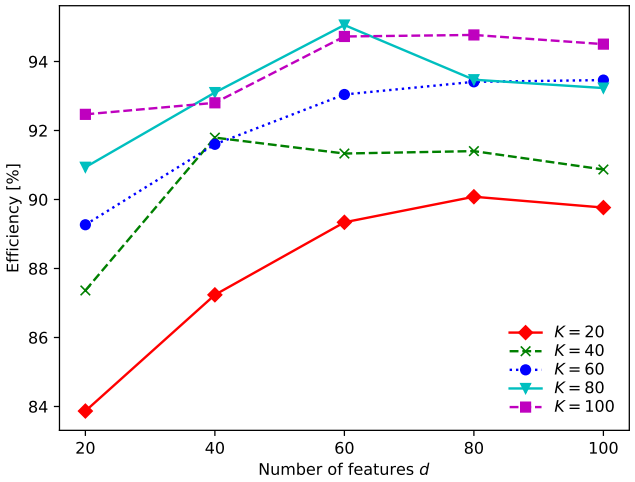

Fig. 3. Parallel efficiency of the EM-L3-blocking variant of the EM algorithm.

EM-L3-blocking variant scales very well with the efficiency higher then 83% in all the cases and higher than 90% where $K \geq 40$ and $d \geq 40$.

VI. Conclusions and Future Work

In the paper we described four variants of the EM algorithm. Two of them use level 2 BLAS operations while the remaining two are based on level 3 BLAS operations which can be implemented more efficiently on the contemporary hardware. We proposed a parallelization scheme for all the variants using OpenMP threads. The results of the study indicate that a combination of level 3 BLAS operations with the blocking for last level cache achieves the shortest runtime for all tested datasets. The resulting algorithm scales very well on a 24-core system.

An obvious extension our work would be a hybrid parallelization using many nodes of the cluster. In this method a parallel application could consists of processes communicating using a message-passing (e.g., MPI [12]) framework. One MPI process would be executed in each compute node of the cluster. Each process would execute in several OpenMP threads, with the number of threads equal to the number of cores in a compute node. A reduction operation in the `MStep` function would be performed hierarchically, first on the process level, then on the MPI application level. We have successfully applied this approach to multi-node parallelization of the well-known K-means algorithm [13].

References

[1] G. McLachlan and D. Peel, *Finite Mixture Models*. New York: Wiley, 2000.
[2] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Royal Stat. Soc. Ser. B*, vol. 39, no. 1, pp. 1–38, 1977.
[3] W. Kwedlo, "A parallel EM algorithm for Gaussian mixture models implemented on a NUMA system using OpenMP," in *Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing PDP 2014*. IEEE CPS, 2014, pp. 292–298.
[4] L. Blackford, J. Demmel, J. Dongarra, I. Duff, S.Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. Whaley, "An updated set of basic linear algebra subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002.
[5] M. Kowarschik and C. Weiß, *An overview of cache optimization techniques and cache-aware numerical algorithms*, ser. Lecture Notes in Computer Science, vol 2625. Springer, 2003, pp. 213–232.
[6] G. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*. New York: Wiley, 2008.
[7] R. A. Redner and H. F. Walker, "Mixture densities, maximum likelihood and the EM algorithm," *SIAM Rev.*, vol. 26, no. 2, pp. 195–239, 1984. [Online]. Available: https://www.jstor.org/stable/2030064
[8] G. H. Golub and C. F. van Loan, *Matrix Computations*. Baltimore, MD: Johns Hopkins, 1996.
[9] OpenMP Architecture Review Board, "OpenMP application program interface version 4.5," http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf, 2015.
[10] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective communication: theory, practice, and experience," *Concurr. Comput. Pract. Exp.*, vol. 19, no. 13, pp. 1749–1783, 2007.
[11] R. Maitra and V. Melnykov, "Simulating data to study performance of finite mixture modeling and clustering algorithms," *J. Comput. Graph. Stat.*, vol. 19, no. 2, pp. 354–376, 2010. [Online]. Available: https://doi.org/10.1198/jcgs.2009.08054
[12] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard Version 3.1," 2015. [Online]. Available: http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
[13] W. Kwedlo and P. J. Czochański, "A hybrid MPI/OpenMP parallelization of K-means algorithms accelerated using the triangle inequality," *IEEE Access*, vol. 7, pp. 42 280–42 297, 2019.