Adaptive Programming: Application to a Semi Supervised Point Prototype Clustering Algorithm

H. Lamehamedi
Computer Science Departement
Rensselaer Polytechnic Institute
Troy, NY 12180
lamehh@cs.rpi.edu

D. Kebbal and E-G Talbi
LIFL
USTL
59655- Villeneuve d'Ascq- Cedex - France
Djemai.kebbal@lifl.fr

A. Bensaid
Computer Science Departement
Al AKhawayn University in Ifrane
Ifrane, Morocco
amine@alakhwayn.ma

A. Benllaachia
Computer Science Departement
George Washignton University
Washington DC
abellaachia@sleimantek.com

Abstract

This paper presents an experiment on adaptive programming with a long lifespan application, a clustering algorithm used in the Magnetic Reasoning Imagery. Our goal is to provide tools that allow the application to benefit at the highest degree from the workstations availability in a Network of Workstations with respect to the ownership aspect of such platforms. In addition, the fault risks are important in such environments preventing thus the parallel application from progress. The well suited solution of this problem is the use of a parallel adaptive system. Parallel adaptive systems constraint the application to adapt its parallelism degree to the load provided by the underlying environment.

1 Introduction

Parallel and distributed programming has shown a great porgress both in methods and in implementation. This is due on one hand to the emergence of non costly workstations with constantly increasing performance and on the other hand to the requirements of a large class of compute intensive applications used in different domains (medical treatment, meteoraulogic computing, etc). The trends recently is the use of networks of workstations (NOWs) as platforms for parallel computing rather than specialized more expensive multiprocessors. Using such platforms may lead to two major problems: load balancing and ownership property. The former consists not only of how to balance the load on a predefined set of nodes but rather of how to detect dynamically idle nodes and how to allow the application to benefit dynamically from those nodes. The second problem on the other hand consists of how to retreat the load created by the application when a user reclaims his workstation especially if there is no sufficient nodes on which the load can be relocated. Thus the application must provide tools allowing it to change dynamically its parallelsim degree following the resource availability. Adaptive programming which allow the application to start new processes which must handle additional computation when some nodes become idle and to retreat processes when some nodes become overloaded or owned is a well suited solution to this problem. Some systems based on this paradigm have been developed and evaluated Piranha [1], MARS [2] and CARMI [3].

Clustering techniques refer to identifying the number of c clusters in a data universe X comprised of n data samples, and partitioning X into c clusters. There are three kinds of c-partitions of data: (hard, porobabilistic and fuzzy). Two important issues to consider in this regard are how to measure the similarity between pairs of observations and how to evaluate the partitions once they are formed. One of the simplest similarity measures is distance between all pairs of observations. Bezdeck developed an extremely powerful classification method to accomodate fuzzy data. The fuzzy c-means algorithm, this thechnique is completely unsupervised, at the end of the clustering the algorithm has no way to know which pattern is in which class; this problem can only be resolved by human intervention. One reason for segmenting an image automatically is to avoid the tedious, costly and practically impossible task of having a human assign a tissue to each of these pixels. Therefore the use of new supervised techniques. In this work we consider the Semi Supervised Point Prototype Clustering, ssPPC, algorithm. ssPPC uses a finite set of labeled data to help clustering the set of unlabeled data.

We present the parallelization of clustering techniques with applications to segment MRI's for detection of tumor. Magnetic Resonnance Imaging (MRI) has become a widely-used method of high quality medical imaging. This is especially true for imaging the brain where MR's non-intrusiveness is a clear advantage. The better MRI segmentation techniques are too slow, improving speed via parallelisation will considerably improve their competitiveness. For this purpose we present a parallelisation for the FCM and the ssPPC FCM based algorithm using MARS environment.

2 Clustering Algorithms

Clustering algorithms attempt to optimize the assignment of like objects to homogeneous classes or clusters. Consider a set $X = (x_1, x_2, x_3, ...x_n)$ to be clustered into c groups of like objects. Each $x_i(p)$ is a feature vector consisting of p real-valued measurements describing the p features of the object represented by x_i. The features could be length, width, color, etc. In this work we describe the task of extracting tumor from slices, this corresponds to segmenting MR images of the brain. Each brain slice consists of three feature images: T1-weighted, proton density weighted (PD-weighted), and T2-weighted. Patterns represent pixels with three features. Clustering in that case refers to assigning a tissue label to each of the pixels. In most of cases a small set of labeled data can be available from experts, therefore it is more useful and helpful to use supervised techniques to cluster the images rather than using unsupervised techniques. In this work we address the use of an unsupervised technique the fuzzy c-means, FCM, algorithm [4], and a point prototype clustering algorithm attached to the fuzzy c-means algorithm (ssPPC), ssPPC-FCM. ssPPC algorithm is based on the fuzzy c-means, FCM, algorithm. The two algorithms produce a membership matrix for all the patterns in all the c classes. Sequential execution of these algorithms is much time consuming, thus the idea to parallelize them for a more realistic execution time, and more competitiveness.

A clustering problem consists of a set of data patterns $X = \{x_1, x_2, ..., x_n\} \subset \mathcal{R}^p$ which must be classified into c clusters. Each element x_i is a vector of p real valued measurements describing features of the object represented by x_i. Clustering in unlabeled data set X consists of assigning (in a hard, fuzzy or probabilistic manner) of each of the $x_i \in X$ to one of the c clusters.

2.1 The Fuzzy C-Means Algorithm

FCM is a clustering algorithm [5]. It computes the centers of the clusters from the data set and minimizes the weighted sum of the distances between the patterns and the centers. The membership functions are found by solving the following problem [4]:

minimize:

$$J_m(U,V,X) = \sum_{i=1}^{n}\sum_{j=1}^{c} u_{ji}^m \|x_i - v_j\|_A^2. \qquad (1)$$

subject to: $0 \leq u_{ji} \leq 1,\ for\ 1 \leq j \leq c, 1 \leq i \leq n,$
$\sum_{j=1}^{c} u_{ji} = 1,\ for\ 1 \leq i \leq n,\ and$
$0 < \sum_{i=1}^{n} u_{ji} < n\ for\ 1 \leq j \leq c.$

The distance between a centroid v_i and a pattern x_k is computed as follows:

$$\|x_k - v_i\|_A^2 = (v_i - x_k)^T A^{-1}(v_i - x_k). \qquad (2)$$

Where A is a positive definite $(n \times n)$ matrix which induces the distance metric used. We consider two different distance metrics: the Euclidean and Mahalanobis distances. The choice of a distance metric determines the shape of the clusters that will be recognized by the clustering algorithm [6]. The two distance metrics under consideration are induced (in eqat. 1) by the following choices for matrix A:

$A = I \rightarrow$ Euclidean distance.
$A = C_y \rightarrow$ Mahalanobis distance.

Where I is the identity matrix, C_y is the sample covariance matrix of the data set X.

The following parameters are used throughout the paper:

T : is the iterations limit,
m : determines the fuzziness of the clusters,
$E_t = \|U_t - U_{t-1}\|_{err}$: represents the change in matrix U between iterations $t-1$ and t,
ϵ : is the termination threshold on E_t,
U : is the $c \times n$ fuzzy membership matrix,
V : is the c-tuple of cluster centers,
$\|.\|$: is the norm used, $\|x\|_A^2 = x^T A^{-1} x$.

FCM Algorithm /* Set c, m, T, the inner product determining $\|.\|$ and the stop parameter ϵ. */

FCM.1 Initialize

$$v_{i,o} = m + (i-1/c-1)(M-m)$$
$$i = 1, \ldots, c. \qquad (3)$$

where: $m_j = min_k(x_{jk})$ and $M_j = max_k(x_{jk}); j = 1,2,...,p.$

FCM.2 Compute U_0:

$$u_{ik,0} = [\sum_{j=1}^{c}(\|x_k - v_{i,0}\|_A / \|x_k - v_{j,0}\|_A)^{2/(m-1)}]^{-1}$$
$$\forall i, k. \qquad (4)$$

- **For** $t = 1\ to\ T$

FCM.3 Compute V_t:

$$v_{i,t} = \sum_{k=1}^{n}(u_{ik,t-1})^m x_k / \sum_{k=1}^{n}(u_{ik,t-1})^m$$
$$\forall i. \qquad (5)$$

FCM.4 Compute U_t:

$$u_{ik,t} = [\sum_{j=1}^{c}(\|x_k - v_{i,t}\|_A / \|x_k - v_{j,t}\|_A)^{2/(m-1)}]^{-1}, \forall i, k. \qquad (6)$$

FCM.5 Compute $E_t = \|U_t - U_{t-1}\|_{err}$.
  - **If** $E_t < \epsilon$ **then** stop

- **Endfor**

The overall time complexity of FCM is $O(nc^2p^2)$ with Mahalanobis distance and $O(nc^2p)$ with Euclidian distance.

Parallel FCM, (PFCM) algorithm: The parallel version of the FCM algorithm was developed following a master/slave approach. The computation is iterative and consists of s slaves controled by the master. Indeed, the algorithm is described here:

PFCM.1 Partition the initial set of patterns among the slaves. Each slave will get n/s patterns. Where n is the number of patterns and s is the number of slaves launched.

$\{\underbrace{x_1, x_2, ..., x_{n/s}}_{\text{Slave 1}} | \underbrace{x_{(n/s)+1}, ..., x_{2n/s}}_{\text{Slave 2}} | ....|$
$\underbrace{x_{(j-1)n/s}, ..., x_{jn/s}}_{\text{Slave j}} | ....| \underbrace{x_{(n-1)(n/s)+1}, ..., x_n}_{\text{Slave s}}\}$

PFCM.2 Initialize the v_i's (FCM.1)and broadcast them to the slaves.

PFCM.3 Each slave will receive the value of the v_i's and compute the membership values of the patterns it holds. Each slave j operates separately on its subset of data $\{x_k, k = (j-1)n/s, ..., jn/s\}$, this step is the end of the initialization part.

$u_{ik,0} = [\sum_{j=1}^{c}(\|x_k - v_{i,0}\|_A / \|x_k - v_{j,0}\|_A)^{2(m-1)}]^{-1} \; \forall i,k.$

- For t=1 to T

  PFCM.4 when initiated by the master, the slaves will perform a part of the computations of $V's$. So each slave j will compute:
  $\alpha_{i,j} = \sum_{k=1}^{sizex}(u_{ik,t-1})^m x_k \quad \beta_{i,j} = \sum_{k=1}^{sizex}(u_{ik,t-1})^m$.
  where $sizex = n/s$ is the number of patterns received by each slave.

  PFCM.5 Each slave j sends these results ($\alpha_{i,j}$ and $\beta_{i,j}$) to the master, which then aggregates to compute the V's and broadcasts them to the slaves. $v_{i,t} = (\sum_{j=1}^{s}\alpha_{i,j})/(\sum_{j=1}^{s}\beta_{i,j}) \; i = 1, ..., c$.

  PFCM.6 Each slave receives the value of the cluster centers and computes the membership values of the patterns $\{x'_k s\}$ it holds. Each slave operates separately on its subset of data.
  $u_{ik,t} = [\sum_{j=1}^{sizex}(\|x_k - v_{i,t}\|_A / \|x_k - v_{j,t}\|_A)^{2(m-1)}]^{-1} \; \forall i,k.$

  PFCM.7 At this time the slaves begin computing the error. The portion of error at each slave j is computed and then sent to the master.
  $error_j = \sum_{k=1}^{sizex}\sum_{i=1}^{c}(u_{ik,t-1} - u_{ik,t})^2$ for $j = 1...s$.
  Then at the master level the errors are aggregated to yield:
  $E_t = (\sum_{j=1}^{s} error_j)^{1/2}$.

  - if $E_t < \epsilon$ Then stop.

- Endfor

The time complexity of the parallel version of FCM is $O(c^2pn/s)$ using Euclidean distance and $O(c^2p^2n/s)$ with Mahalanobis distance.

2.2 The Semi Supervised Point Prototype Clustering Algorithm

The performance of clustering algorithms (like FCM) often suffers from one or both of the following problems [6]: (a)the clustering criterion (like the one adopted by FCM in equation 2 is not a good enough measure of the actual quality of a given partition; and (b) often the optimization procedures used only guarantee locally-optimal solutions. The semi supervised point-prototype clustering algorithm (ssPPC) [6] offers a means of improving the performance of any point-prototype clustering algorithm (PPCA) like FCM, through the use of training labeled patterns. Hence in ssPPC, the data set is composed of two parts: n_d labeled patterns ($\{x_k^d\}, 1 \le k \le n_d$) and n_u unlabeled patterns ($\{x_k^u\}, 1 \le k \le n_u$). The labeled patterns guide the classification of the unlabeled patterns onto c target classes. FCM (or any other PPCA) is applied to the unlabeled data with a number of clusters equal to n_d. The resulting n_d clusters are merged into c classes, by assigning each cluster C_i (whose center is v_i) to the same class as the labeled pattern that is nearest (most similar) to.

ssPPC Algorithm: /* Set c with the constraint $n_d \ge c$, m, T, the inner product determining $\|.\|$ and the stop parameter ϵ. */

ssPPC.1: Initialize the cluster centers with the labeled patterns.

ssPPC.2: Apply FCM (skipping FCM.1.1) to the unlabeled data ($\{x_k^u\}, 1 \le k \le n_u$).
Run FCM to termination at (U_f^u, V_f^u).

ssPPC.3 : Compute the nearest neighbor among the labeled patterns to each final $v_{j,f}^u$ ($j = 1, ..., n_d$). Form matrix B:
$B = [U_{nn(1)}^d, U_{nn(2)}^d, ..., U_{nn(nd)}^d]$. Where $nn(j) = argmain_{11 \le s \le nd}\{\|x_s^d - v_{j,f}^u\|\}$, $j = 1, 2, ..., n_d$.

ssPPC.4: Calculate $\hat{U}_f^u = BU_f^u$.

The overall time complexity is then $O(cn_d^2 n_u)$.

Parallel ssPPC (PssPPC) algorithm: In this section, we present the parallel adpative version of the algorithm. As we have mentioned it, implementing an adaptive version of an application consists of structuring the application to react to impredictible changes in the system state.i.e. giving the application a framework to change its parallelism degree in a non deterministic manner.

In MARS [2], the adaptive application composed of a master and some workers must react to two events: the *fold* event generated when a node is overloaded or reclaimed by its owner. The MARS system requires then the application to evacuate the computation previously submitted on that node. The second event is the *unfold* event generated when a node becomes idle. The application may then benefit from this availability by starting a new worker on that node.

The adaptive programming methodology starts by splitting the computation into some work units. Each work unit is a data partition on which some operations must be performed. In addition, a module (work server thread) responsible for allocating work and receving results from workers must be provided. At the worker level, a thread (worker thread) which perform the computation is developed. Moereover, in order to keep track of the partial work in the case of a fold, the user may specify a function which packs and returns back the partially porcessed work.

Due to its iterative aspect, ssPPC alogrithm requires the work server to generate the tasks of each iteration and to wait for their termination.

The adaptive version of the algorithm is described here:

1. The $V's$ are initialized with the labeled patterns and work units are constructed. Each work unit consists of a copy of the vector V and a chunk of the patterns (partition of X).
2. At this stage, we must apply the equivalent of the FCM algortihm to at (U_f^u, V_f^u) on the unlabeled patterns, with n_d classes. Thus each worker computes the membership of the patterns of the unit as in PFCM.3. After which, it returns the partition of the matrix U computed.
3. Upon receiving all $U's$ chuncks, the master proceeds to generate the second type of work units in order to compute the $V's$. Each work unit consists of a chunk of the matrix U, and its corresponding part of the data patterns X. The results of each unit i a part of the computation of $V's$ reprented by α and $Beta$ as in PFCM.4.
4. When receiving the results of all these work units, the master updates V as in PFCM.5 and then proceeds to generate the second type of work units in order to compute the membership matrix U. So, a work unit of this type consists of a copy of the centers vector V and a chunk of the membership matrix X.
5. For each unit, the corresponding partition of the matrix U is computed and returned to the master as in PFCM.6.
6. Upon receiving all chunks, the master computes the error value as in PFCM.7. If the required precision is not reached ($E_t \geq \epsilon$), then it starts another iteration (step 3).
7. When the FCM-equivalent algorithm is terminated, the master computes the nearest neighbor among the labeled patterns to the final V's at the master level, and form B:
   $B = [U^d_{nn(1)}, U^d_{nn(2)}, ..., U^d_{nn(nd)}]$. And then generates the final work units. Each task is composed of a copy of the matrix B and a chunk of the matrix U. For each

work unit, we compute the final membership values (merge the nd clusters into c classes). So at work unit k: $\hat{U}^u_{f,k} = BU^u_{f,k}$. Where $U^u_{f,k} = [U^u_{(k-1)(sizex)+1}, U^u_{(k-1)(sizex)+2}, \ldots, U^u_{k(sizex)}]$. The worker then sends the results to the master.

8. Upond receiving all results, the master reconstructs the final classes.

When a fold operation is started, the partially porcessed work and its related results are packed in a structure and sent back to the master. So this computation can be handled by another worker without loss of already achieved work. The partial work structure is composed of the part of membership matrix U already computed and its corresponding indices (i and j) if the folded worker was working on a membership parition or of parts of α and β if was working on the vector V.

Another aspect related to the failure of nodes on which workers are running. With a classic application composed of a set of processes communicating by exchanging messages, the failure of one process causes the entire application to restart from the begining or recover all the application if a recovery mechanism is available. With the adaptive application, the failure of a worker is resolved by restarting the computation on which the failed worker was working on another worker. So many hours even many days of work can not be lost.

3 Conclusion

The clustering applications are known as time and resource consuming. The use of the MARS environment is very cost efficient though it doesn't have a better speed up than the non-adaptive version. But it is well adapted to networks of workstations because:

- The application runs for as much time as needed. The adaptive programming allows the application to adapt its computation to the load of the underlying envrionment and to respect the ownership property of these platforms.
- The faulty nature of such environments decrease the probability of such applications to complete. The adaptive version, involves only a minimum part of the application in the recovery procedure and a without loss of work.

References

[1] D. L. Kaminsky. *Adaptive Parallelism with Piranha.* PhD thesis, Yale University, 1994.

[2] Z. Hafidi, E-G. Talbi, and J-M. Geib. MARS: Adaptive scheduling of parallel applications in a multi-user heterogeneous environment. In *ESPPE'96 proceedings, Alpe d'Huez, France*, pages 119–122, April 1996.

[3] J. Pruyne and M. Livny. Interfacing Condor and PVM to harness the cycles of workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[4] A. Bensaid and J. Bezdek. Partial Supervision Based on Point-Prototype Clustering Algorithms. In *Fourth European Congress on Intelligent Techniques and Soft Computing*, pages 1402–1406, Aachan, Germany, Sep 1996.

[5] A. Jain and R. Dubes. *Algorithmes for Clustering Data.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

[6] A. Bensaid, L. Hall, J. Bezdek, L. Clarke, M. Silbiger, J. Arrington, and R. Murthagh. Validity-Guided (Re)clustering with Applications to Image Segmentation. *IEEE Trans. on Fuzzy Systems*, 4(2):112–123, 1996.