P-EDR: An Algorithm for Parallel Implementation of Parzen Density Estimation from Uncertain Observations

P.E. López de Teruel, J.M. García, M. Acacio, O. Cánovas
Departamento de Ingeniería y Tecnología de Computadores
University of Murcia
Facultad de Informática. Campus de Espinardo, 30080, Murcia (SPAIN)
pedroe@ditec.um.es

Abstract

We have developed a parallel version of a new algorithm for nonparametric density estimation when the input samples are not directly known, or they have some noise. The algorithm is an extension of the Parzen method for exact observations, but management of uncertainty implies heavy computational loads in terms of both calculus and storage. Therefore, a parallel version of the algorithm is more adequate to solve this extended problem in a practical time, specially for samples of medium-large sizes. Our parallel algorithm has been designed in an SPMD style and implemented in a message-passing parallel environment. An efficient treatment of the distribution of the main data structure among processors, together with a low communication cost scheme results in a high scalability of the algorithm. Preliminary performance evaluations in a cluster of workstations show excellent speed-up results.

1. Introduction

Parzen method [6] is a well-known statistical technique used in nonparametric density estimation from observed samples. The application of this technique is very diverse: Pattern matching, machine learning, function approximation, and many other inference tasks, to be applied in many different fields related to statistics, such as biomedicine, engineering, sociology, and so on. The main idea of the original technique is to situate a small 'window' density function on each sample of the dataset, in order to obtain a *regularized* version of the target density that presumably generated the samples. The window function is usually a simple parametrized function, such as a triangular or normal density, that is known as the *kernel* function. In this way, the free parameter that measures the width of the window controls the trade-off between *overfitting* (whose extreme case is that of a density function composed of impulses situated in the samples), and smoothing to achieve *regularization* (that can result in a blurring of the target density).

Of course, many techniques for nonparametric density estimation other than the *kernel approach* exist, such as, for example, the *average shifted histogram*, the *frequency polygon*, the *histospline* approach, *variable partition histograms*, the *statistically equivalent blocks* method, *nearest neighbour* techniques, and many more. Each of these techniques has its own advantages and drawbacks, but perhaps Parzen's is the one that regularizes the target density in a more formal way. A good overview of some of these and many other techniques can be found in [4].

All these techniques, including Parzen's method, are thought to cope only with exact data. But often the datasets that are given as input come in an uncertain form, with some kind of lack of knowledge on the attributes of the examples. Much less approximations have been made to cope with this kind of input in the specific problem of estimating the density function. Lucy's algorithm [5], for example, is based on an histogram approach, that has the serious problem of not being regularizated: there is no trade-off between the amount of smoothness and data fitting. Many other approaches to the management of uncertainty have been proposed, but none of them tries to solve the problem of the estimation of the density. Instead, they directly cope with inference tasks, either in a more or less symbolic way (as in Dempster-Shafer theory [10], or fuzzy rules systems [11]), or in a more probabilistic sense, as in Bayesian inference [2].

We are interested, anyway, in the direct calculus of the probability density function, as in Lucy's method, but without renouncing to regularization. Perhaps Parzen's method is one of the best methods for estimating regularizated densities with exact data, so an adaptation to cope with uncertainty would be a good solution. Recently, we have developed and tested a new algorithm, directly adapted from the classical Parzen method [8]. This algo-

rithm, called EDR (Empirical variable Deconvolution by Regularization), has all the advantages of original Parzen method (regularization, wide range of applicability,…), but it has also one important drawback: the amount of computation and memory needed is quadratic in the number of samples, as a price to pay for the capability of managing uncertainty. This can be a very serious inconvenient when datasets are very large, which is unfortunately the case in practical applications, specially when the input data are more or less uncertain.

Nevertheless, a good organization of the code and the data structures involved allow a very efficient parallel implementation in a distributed-memory environment. In this paper, we present the parallel version of our EDR algorithm for a message-passing parallel machine. We show that our approach can obtain very high speed-up values, as an efficient treatment of memory is performed, resulting in a communication scheme that minimizes the network traffic, in distributed-memory environments, and that reduces local swapping. This makes the parallel algorithm highly scalable, making it capable of managing large databases. Preliminary performance tests are made in a cluster of PCs, that seem to confirm these predictions.

2. Preliminaries

In order to establish a theoretical framework, we will introduce some notation and review some background concepts. Parzen's method consists in approximating the original, unknown density function $p(x)$ with the following estimation (given a sample $\{x_i\}$ of size N of the random variable):

$$p_N(x) \equiv \frac{1}{N\ h_N} \sum_{i=1}^{N} k\left(\frac{x - x_i}{h_N}\right) \tag{1}$$

where $k(x)$ is a smoothing kernel density function. This approximation has the desired properties of smoothness and convergence to the true $p(x)$, if the window width $h_N \to 0$ when $N \to \infty$. The method is a good alternative for nonparametric statistics problems, but it has two principal drawbacks: One is that it is designed to work over samples that are assumed to have no error in the measurement process, and this is usually not the case in many practical problems. The other disadvantage is that it works with the whole sample in the learning and inference stages, thus requiring more computational power. As it was mentioned in the introduction, an extension of this method to cope with inexact measures in the samples, was recently developed and successfully validated [8]. We will summarize, in the rest of this section, the main ideas of this new approach.

First, we will state formally the problem of density estimation through uncertain observations. Let $p(x)$ be the density that we must estimate, through a series of random examples $\{x_i\}$, with $i \in \{1..N\}$, drawn from that density. But we have no access to the true values of the samples x_i. We only have observations that may have experimented a perturbation in the measurement process, or simply they contain some kind of uncertainty: subjective judgments of experts, other kind of external noise, etc. We will call these indirect measures $\{s_i\}$, and they will be related to the true values $\{x_i\}$ by the *likelihood* functions $\{l_i(x_i)\}$, which describe statistically our knowledge of the original sample. The likelihood functions can model the precision of the measurement instrument, the vagueness of the description of an expert, and so on. Then, the density $q(s)$ of the observable magnitude is related to the target density $p(x)$ by:

$$q(s) = \int_{-\infty}^{+\infty} q(s \mid x) p(x) dx \tag{2}$$

The aim is to estimate $p(x)$, but we only have $l_i(x_i)=q(s_i|x_i)$, the likelihoods of the observed measures, and we can estimate $q(s)$ through the whole set of examples. What we have to do is to *deconvolve* this observed density $q(s)$ with a kind of inverse operator that annuls the noise of the observed samples. The expression for this inverse operator depends on the unknown $p(x)$, but the idea is to use an iterative procedure in which an initial estimation of $p(x)$ is roughly calculated, and a refinement process is performed, in which the initial likelihood functions with a particular location and uncertainty quantity are 'moved' to new locations with new uncertainty values, using the Bayes theorem and the *a priori* current estimation of $p(x)$. Also, during the operation of the algorithm, the regularization process through the kernel functions must be accomplished. This justifies the name given to the algorithm: EDR, that stands for Empirical variable Deconvolution by Regularization. The iteration is performed until convergence of $p(x)$ to a desired precision.

We focus in a case in which gaussian normal densities are used to model both the likelihoods and the kernel function, to obtain a practical algorithm that will serve to solve most of practical reasonable situations. These normality conditions can be summarized as follows:

a) The smoothing kernel (Parzen windows) is gaussian, with the following parameters:

$$k_h(x) = N(x, 0, h) \tag{3}$$

b) The likelihood functions (to model uncertainty) are gaussian, with the following parameters:

$$l_i(x) = N(x, s_i, \varepsilon_i) \tag{4}$$

Observe that $l_i(x)$ functions can be used to model uncertain values in the form $x_i \cong s_i \pm 2\varepsilon_i$, including exact data ($\varepsilon_i=0$) and missing values ($\varepsilon_i=\infty$). We will estimate the target density from *uncertain* samples, where ε_i measures the degree of uncertainty in the observation.

c) The components of the mixture (through iteration of the algorithm) will be approximated by gaussian densities (ψ'_i stands for a component without regularization; ψ_i is the same component after regularization):

$$\psi'_i(x) = N(x, \mu_i, \sigma'_i) \quad (5)$$

$$\psi_i(x) = N(x, \mu_i, \sigma_i) \quad (6)$$

$$\sigma_i = \sqrt{\sigma'^2_i + h^2} \quad (7)$$

This is a reasonable assumption since, as we will see later, they will be obtained as the result of a weighted sum of a large number of functions.

3. Operation

In this section we outline the operation of the algorithm. We will center the discussion in a procedural description, to analyze lately the possibilities for parallelism and data distribution. Further mathematical details and justifications can be found in [8] and [9].

The algorithm takes advantage of the conjugate property of the gaussian density [2]. It allows us analytic computation of the required products and integrals, avoiding intensive numerical techniques. Using it, we can write:

$$N(x, s_i, \varepsilon_i) N(x, \mu_j, \sigma_j) = N(x, \eta_{i,j}, \theta_{i,j}) N(s_i, \mu_j, \sqrt{\varepsilon_i^2 + \sigma_j^2}) \quad (8)$$

where:

$$\eta_{i,j} = \frac{\varepsilon_i^2 \mu_j + \sigma_j^2 s_i}{\varepsilon_i^2 + \sigma_j^2} \quad (9)$$

$$\theta_{i,j} = \sqrt{\frac{\varepsilon_i^2 \sigma_j^2}{\varepsilon_i^2 + \sigma_j^2}} \quad (10)$$

Now we define the following coefficients:

$$\beta_{i,j} \equiv N(s_i, \mu_j, \sqrt{\varepsilon_i^2 + \sigma_j^2}) \quad (11)$$

and the normalized weights:

$$\omega_{i,j} \equiv \frac{\beta_{i,j}}{\sum_{j=1}^{N} \beta_{i,j}} \quad (12)$$

in such a way that the components ψ' will be given by

$$\psi'_i(x) = \sum_{j=1}^{N} \omega_{i,j} N(x, \eta_{i,j}, \theta_{i,j}) \quad (13)$$

Therefore, the parameters of each component ψ', approximated by gaussians, are updated by

$$\mu_i \leftarrow \sum_{j=1}^{N} \omega_{i,j} \eta_{i,j} \quad (14)$$

$$\sigma'_i \leftarrow \sqrt{\sum_{j=1}^{N} \omega_{i,j} (\eta_{i,j}^2 + \theta_{i,j}^2) - \mu_i^2} \quad (15)$$

The parameters of the final components $\psi = k_h * \psi'$ (smoothed) are obtained from (7), once the smoothing parameter h has been selected. The new $p(x)$ is given by:

$$\frac{1}{N} \sum_{i=1}^{N} k_h(x) \cdot \psi'_i(x) = \frac{1}{N} \sum_{i=1}^{N} \psi_i(x) = \frac{1}{N} \sum_{i=1}^{N} N(x, \mu_i, \sigma_i) \quad (16)$$

Given this update rule for each iteration, we must apply it successively until a *fixed point* is reached. Starting from a first approximation to $p(x)$ with $\mu_i = s_i$ and $\sigma_i = 0$, (which is equivalent to ignoring the uncertainty in the observations) we iteratively update μ_i and σ_i from the observations $\{(s_i, \varepsilon_i)\}$, until some kind of convergence is achieved. A good measure can be the L_1 norm between the estimated functions $p(x)$ in two successive iterations. This value is simply the area between the graphics of both estimations, and can be calculated as the integral of the absolute value of the difference between them:

$$L_1 = \int_{-\infty}^{\infty} \left| p^{(k)}(x) - p^{(k+1)}(x) \right| dx \quad (17)$$

When this measure falls below a given tolerance threshold, we can suppose that the two estimations are virtually the same, and that convergence has been reached. So, at the end of each iteration, a numerical integral of the area between the current and the preceding estimation must be calculated. A simple integration technique is valid, i.e. the *trapezoids* method, as no big discontinuities in this area are expected. Recall that the density is smooth, as it uses smoothing kernels[1].

To summarize, we can see that the sequential complexity order of each iteration of the algorithm for the calculus of the μ_j and σ_j values is quadratic in the number of samples N, $O(k_1 \cdot N^2)$. The value of the constant k_1 will be determined by the calculations performed in each iteration, explained in fig. 1. We have to compute a NxN table of values β_{ij}, η_{ij} and θ_{ij}, with some elemental floating point operations (sums, divisions, square roots and exponenciation for the calculus of the normal formula) applied to the current values of μ_j, σ_j, s_i and ε_i. Then, a grouping by columns must be performed, for the calculus of the ω_{ij} values. With these, and the previously obtained η_{ij} and θ_{ij}, final values of the new μ_j and σ_j can be obtained.

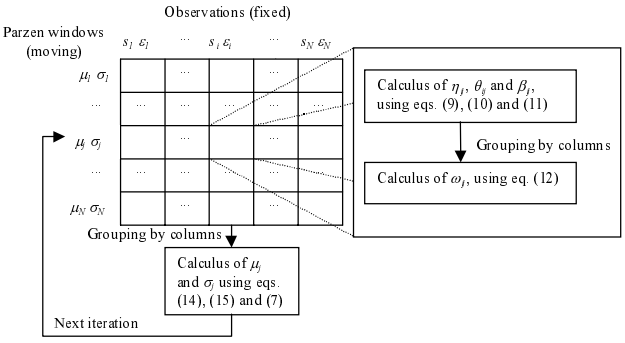

Figure 1. Calculus of the table values in each iteration of the EDR algorithm.

The complexity for the convergence test can be expressed as $O(k_2 \cdot N \cdot I)$, being I the chosen number of inter-

[1] The procedure described here is for unidimensional samples. Extension of this algorithm for multidimensional data is straightforward, and it is based in the conjugate properties of the factorized gaussian functions used. The main loop of the sequential algorithm is essentially the same, but including a multiplication for each dimension.

vals for the numerical integral, and k_2 a constant indicating the execution time of the substraction of two values of a normal distribution (corresponding to $p^{(k+1)}(x)$ and $p^{(k)}(x)$).

Of course, the final complexity order of the algorithm must be obtained multiplying the obtained complexity for each iteration by K, the number of iterations needed to achieve convergence. This value depends on many factors (precision of the convergence test, distribution of the input data, etc.), so it can not be determined beforehand.

4. Parallel Algorithm

It can be extracted, from the previous section, that the most important effort is done when values of means and deviations (μ_j, σ_j) at each iteration are recalculated. In order to do it, the table of $N \times N$ elements with intermediate values shown in fig. 1 must be filled in. As the computational complexity and the size of the main data structure are quadratic with respect to this number, application of this technique in a single processor may introduce time and storage problems[2]. This justifies the attempt of a parallel approach. Hence, it is necessary to study how these intermediate values are related, to find out data dependencies.

At each iteration, computation of the values (μ_j, σ_j) is performed using only the intermediate data stored in column j of the table. Thus, these means and deviations can be calculated without communications if a distribution by columns of this table is done. Each process will be assigned a set of columns of the table, and it will obtain the values of means and deviations for such columns. However, some communication will be needed, as every process must know the values worked out by the rest of processes in the previous iteration to compute the new ones, and also to make the convergence test. Therefore, when a process obtains its local values, it will transmit them to the rest. A first synchronization point emerges.

Furthermore, the convergence test can also take advantage of parallelism. This test is based on the calculus of the L_1 norm, the value that measures the difference between two successive estimations of $p(x)$. It is obtained splitting a sufficiently wide interval (that captures the whole set of examples, with some margin at the extremes) into I small subintervals, and subsequently performing a numerical integration by the *trapezoids* method. Thus, these subintervals can be distributed between all processes, so each of them would calculate a local value of the L_1 norm. Nevertheless, calculation of the L_1 norm requires means and deviations to have been obtained, so it will be initiated when every process has calculated its corresponding subset of these values, and has sent them to the rest of processes. Once each process has computed the numerical value of the integral in the subintervals that it has associated, the global L_1 norm is obtained as the sum of the local values calculated by every process. A new communication appears, and, therefore, a second synchronization point.

Fig. 2 shows in pseudo-code the message-passing program. We have adopted the SPMD style. For simplicity, we suppose that N is an exact multiple of P, but extensions to manage the general case will simply distribute remaining columns among processes. Fig. 3 summarizes graphically the data flow in the P-EDR algorithm (Parallel Empirical Deconvolution by Regularization).

Algorithm: Message-Passing P-EDR.
Input: s_i and ε_i values, $i=1..N$.
Output: μ_i and σ_i values, $i=1..N$.
Initialization:
Process *0* reads the input data –values (s_i and ε_i)–, and broadcasts them to processes *1..P*.
Repeat (*Main Loop*):
Table Calculation:
- Each process *k*, with $k \in [1..P]$, computes the values η_{ij}, θ_{ij} and β_{ij}, using eqs. (8), (9) and (10), for $j=1..N$, and $i=(k-1)\cdot(N/P)+1..k\cdot(N/P)$ (*Column distribution*).
- Each process *k*, with $k \in [1..P]$, computes the values ω_{ij}, using eq. (12), again for $j=1..N$, and $i=(k-1)\cdot(N/P)+1..k\cdot(N/P)$ (*Aggregation by columns on each cell of the column*).

Parzen Windows Update:
- Each process *k*, with $k \in [1..P]$, computes the values μ_i and σ_i, using eqs. (14), (15) and (7), for $i=(k-1)\cdot(N/P)+1..k\cdot(N/P)$ (*Accumulation by columns*).

Broadcast of Values:
- Each process *k*, with $k \in [1..P]$, broadcasts its corresponding updated values μ_i and σ_i for $i=(k-1)\cdot(N/P)+1..k\cdot(N/P)$ to all the rest of processes *1..P* (*Broadcast*).

Convergence Test:
- Each process *k*, with $k \in [1..P]$, computes its corresponding numerical integration of I/P intervals, L_k, from value $a+(k-1)\cdot(b-a)/P$ to $a+k\cdot(b-a)/P$, being a and b the limits of the integral (where the density function becomes zero; that is, the distribution tails). The computation is made according to the trapezoids method, using eq. (17) and where $p^{(k)}$ y $p^{(k+1)}$ are computed with eq. (16) with the former and the current values of μ_i and σ_i (*Distribution of integral by intervals*).
- Process *0* gathers the partial L_k results, and sums them to calculate the L_1 value (*Integral aggregation*).

Until L_1 < Precision Threshold.

Figure 2. Pseudo-code of the P-EDR algorithm.

The P-EDR algorithm is expected to decrease execution time regarding the sequential version, as: a) The number of columns that each process has been assigned is much smaller than in the sequential program. And b) The number of subintervals per process employed in the calculation of the L_1 norm is less than in the sequential version.

[2] In fact, this is not completely true for the EDR algorithm described in this paper, as column calculus summarized in fig. 1 can be overlapped, and the resulting sequential algorithm would be much less memory intensive. But global maintenance of the whole table structure is needed for some variations on the aggregation procedure, resulting in interesting EDR variations, and, besides, all values contained in that table have an interesting statistical interpretation, that can be useful in further postprocessing (see [8] and [9] for details).

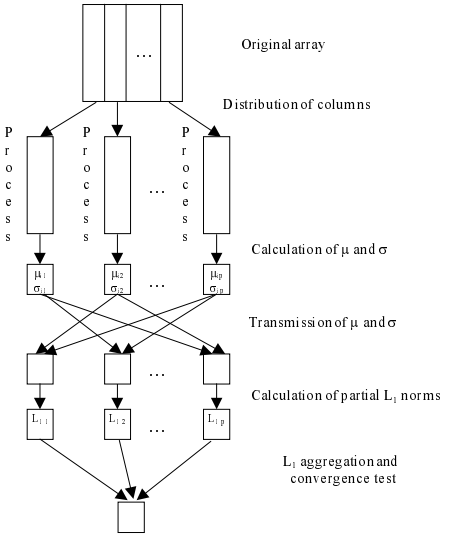

Figure 3. Data flow in P-EDR implementation.

However, parallel algorithm implies some extra communications between processes that, obviously, do not exist in the sequential version. More precisely, it is necessary to interchange the values of means, deviations and local values of the L_1 norm. This fact will be decisive, as good results will only be achieved if the cost of these communications is lower than the benefit obtained with the above reduction in computation time.

In order to predict the gain of the parallel implementation, we will analyze the complexity of the parallel version. P-EDR algorithm involves a reduction on the number of columns, in the calculus of the table, and of subintervals, in the calculus of the L_1 norm for the convergence test. This reduction states the execution order for each process (suppressing communications, that will be taken into account later) as stated in the next table (I/P is the number of intervals for each process, being I the total number of intervals for the calculus of the L_1 norm):

Task	Complexity
Main iteration (η_{ij}, θ_{ij}, β_{ij}, ω_{ij} μ_j and σ_j)	$O(k_1 \cdot N^2/P)$
Calculus of local L_1 norm	$O(k_2 \cdot N \cdot I/P)$

Again, these orders are calculated for each iteration, so they must be multiplied by K, the number of iterations. Observe that the execution order of all the computational tasks involved appears divided by P. Therefore, good speed-up values can be expected, if the communications do not introduce important delays.

We study now the influence of the communications in the parallel version. In first place, the process 0 has to broadcast the initial data to the rest of processes. This is made just once, so this value is constant and it is independent of the number of iterations of the algorithm. In the other place, each process will transmit the following data at each iteration:

- M means and deviations estimated. This information is sent to the rest of processes.
- Local L_1 norm value. At the end of each iteration, partial L_1 norm values must be added to determine when convergence is reached.

Two observations can be made from this analysis:

1. The quantity of bytes transmitted in each iteration is reasonably small (N values of means and deviations, and P values for the convergence test).
2. As the number of processes participating in computation increases, execution order decreases quickly whereas transmitted bytes hardly increase.

With these results we can be optimistic with respect to the parallel implementation of this algorithm. It is expected to obtain an important speed-up when executing parallel version with a certain number of processors. Especially, scalability of the algorithm for distributed-memory environments seems very good, as increment of communications size is unappreciable when the number of processors grows up. So, the traffic in the communication network is expected not to suppose an important bottleneck, even in a network of PCs connected by a LAN with a lower bandwidth [1], [7].

5. Preliminary Performance Results

We made a particular implementation of the parallel program, and carried out some tests, in order to obtain some preliminary results to confirm our predictions. The parallel machine was a cluster of Intel Pentium 200 MHz processor, 32 MB main memory, 256 KB cache memory and Fast Ethernet 3Com 905-network adapter with O.S. Linux 2.0.32. The programming was made in C language with the standard MPI library [3] (MPICH v.1.0.13 implementation). P-EDR was tested using samples of 400, 800, 1000 and 1500 unidimensional items, and adequate values of h and accuracy factor for convergence. The next table summarizes the results obtained with each sample size and using distinct number of processors (one process per processor). Fig. 4 shows also graphically these results.

Procesess	400 items		800 items		1000 items	1500 items
	Time	*Speed-up*	*Time*	*Speed-up*	*Time*	*Time*
1	13.50 s.	1	36.27 s.	1	--	--
2	13.50 s.	1	33.11 s.	1.09	43.00 s.	--
3	6.96 s.	1.94	17.41 s.	2.08	21.87 s.	58.40 s.
4	5.73 s.	2.36	12.86 s.	2.82	15.87 s.	41.36 s.
5	5.00 s.	2.70	10.31 s.	3.52	12.89 s.	31.77 s.
6	5.00 s.	2.70	9.42 s.	3.85	10.80 s.	26.24 s.
7	4.797 s.	2.81	9.20 s.	3.94	9.63 s.	22.68 s.

We can observe that sequential version of EDR algorithm was not executed with samples of 1000 or 1500 items. In these cases more than 32 Mbytes of RAM were

needed to solve the problem, thus involving a big amount of swapping (see footnote 1 for some considerations on this). This fact illustrates one of the main advantages of clusters of workstations: the exploitation of distributed-memory resources (RAM). Though it is not possible to obtain the speed-up for these cases, we can appreciate that time employed to solve the problem with 7 processors is less than half the time needed to do it with 3 processors, indicating a very good scalability in the implementation.

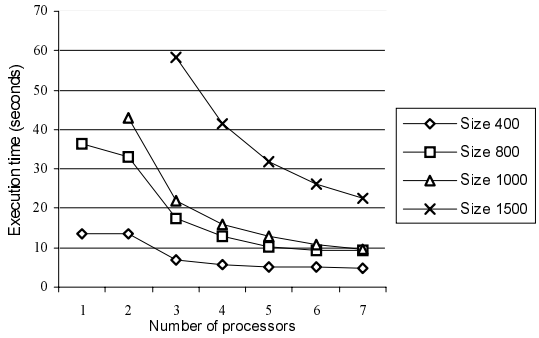

Figure 4. Experimental results.

6. Conclusions and Future Work

We have presented a parallel version of the EDR algorithm, a recent and innovative algorithm for non-parametric density estimation in presence of uncertainty in the data. The technique has many advantages, both in practical and theoretical senses –modeling of uncertainty, regularization feature, and the fact that it is a natural extension of the well known Parzen method–. But its main drawback is that it is computationally intensive in both memory and processing necessities. We exploit the symmetry property of the calculus and the data structures involved in the parallel version, P-EDR, to obtain a message-passing algorithm with very good speed-up and scalability properties.

We made a particular implementation of the algorithm in a distributed-memory parallel machine, a cluster of workstations communicated with a fast network, using the well-known standard MPI. This particular implementation confirms the expected good results in speed-up and scalability. Also, the tests performed seem to justify the use of low-cost parallel environments in this and many other kind of tasks in which the amount of data uses to be very big and, therefore, are very computationally expensive.

In summary, the most important contributions of this paper are the following: a) The design, analysis and implementation of the EDR parallel algorithm. This parallel algorithm will allow formal statistic treatment of uncertainty in big databases, solving the problem in practical execution times b) Our parallel developed algorithm (in SPMD style) has a low communication cost and good scalability, so it seems adequate to execute in a low-cost parallel environment c) Finally, we have implemented our algorithm in a network of PCs, obtaining very promising results in terms of both performance and speed-up.

As a future work, we will take advantage of the MPI portability, in order to implement and test the algorithm in other parallel computers, such as MPPs, like IBM SP2 or Cray T3D. A study with a more exhaustive performance evaluation, extensions to manage multidimensional data, and other variations, will also be fulfilled. Finally, our group is also working on parallel implementations for other statistical tasks with big computational cost, specially on those cases in which low communication algorithms can be developed, and, subsequently, efficient implementations in low cost parallel machines.

Acknowledgments

This work has been partially supported by the Spanish CICYT under grants TIC97-0897-C04-02 and TIC98-0559. The authors would also like to thank Alberto Ruiz for his very helpful comments and suggestions.

References

[1] Anderson, T.E., Culler, D.E. and Patterson, D.A. *"A Case for NOW"*. IEEE Micro, Vol. 15, No 1, pp. 54-64, 1995.

[2] Berger, J. *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, 1985.

[3] Gropp, W., Lusk, E., and Skjellum, A. *"Using MPI: Portable Parallel Programming with the Message Passing Interface"*. MIT Press, 1994.

[4] Izenman, A.J. *"Recent Developments in Nonparametric Density Estimation"*. Journal of American Statistical Association. Vol. 86, No. 413, pp. 205-224, 1991.

[5] Lucy, L.B. *"An Iterative Technique for the Rectification of the Observed Distributions"*. The Astronomical Journal. Vol. 79, No. 6, pp. 745-754, 1974.

[6] Parzen, E. *"On Estimation of a Probability Density Function and Mode"*. Annals of Mathematical Statistics. Vol. 33, pp. 1065-1076, 1962.

[7] Piernas, J., Flores, A. and García, J.M. *"Analyzing the Performance of MPI in a Cluster of Workstations Based on Fast Ethernet"*. 4th European PVM/MPI Users' Group Meeting, LNCS, Vol. 1332, pp. 17-24. Springer-Verlag, 1997.

[8] Ruiz, A., López de Teruel, P.E. and Garrido, M.C. *"Kernel Density Estimation from Heterogeneous Indirect Observations"*. Proc. of the Learning'98, Madrid, pp. 86-96, 1998.

[9] Ruiz, A., López de Teruel, P.E. and Garrido, M.C. *"Probabilistic Inference from Arbitrary Uncertainty using* Mixtures *of Factorized Generalized Gaussians"*. Journal of Artificial Intelligence Research, Vol 9, pp. 167-217, 1998.

[10] Shafer, G. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[11] Zadeh, L.A. *"Fuzzy Sets as a Basis for a Theory of Possibility"*. Fuzzy Sets and Systems,Vol. 1, pp. 3-28, 1978.