International Journal of Future Generation Communication and Networking
Vol. 13, No. 2, (2020), pp. 1710 – 1716

Implementation of Parallel Programming Method that Support Various Machine Learning Algorithms

[1]Eriki Venkata Karthik, [2]Puttu Chandra Sekhar
[1]Research Scholar, Shri Jagdishprasad Jhabarmal Tibrewala University,Rajasthan
[2]Assistant Professor, Annamacharya Institute of Technology and Sciences, Kadapa- Andhra Pradesh

ABSTRACT:

As we are towards the start of the multiprocessing period were there are lot of advances in the field of technology, PCs will have progressively numerous centers and computing capacity however there is still horrible programming structure for these designs, and therefore no straightforward and brought together path for AI to exploit the possibly accelerate. In this paper, we build up a method that is comprehensively applicable equal programming technique, one that is effortlessly applied to a wide range of learning calculations in the field of machine learning. Our work is in unmistakable differentiation to the convention in AI of structuring approaches to accelerate a solitary calculation at once. In particular, we show that calculations that fit the Statistical Query model can be written in a specific "summation structure," which permits them to be effectively standard allelized on multicore PCs. We propose the methods for parallel programming that direct relapse k-implies, strategic regressioninnocent Bayes, Gaussian discriminant examination and back propagation.

Keywords:*Parallel programming method*

INTRODUCTION:

Recurrence scaling on silicon—the capacity to drive chips at ever higher clock rates—is starting to hit a force limit as gadget geometries contract because of spillage, and just in light of the fact that CMOS devours power each time it changes state. However Moore's law, the thickness of circuits multiplying each age, is anticipated to last somewhere in the range of 10 and 20 additional years for silicon based circuits . By keeping clock recurrence fixed, however multiplying the quantity of preparing centers on a chip, one can keep up lower power while multiplying the speed of numerous applications. This has constrained an industry-wide move to multicore.

We consequently approach a time of expanding quantities of centers per chip, however there is up 'til now no decent edge work for AI to exploit huge quantities of centers. There are many equal programming dialects, for example, Orca, Occam ABCL, SNOW, MPI and PARLOG, yet none of these methodologies make it evident how to parallelize a specific calculation. There is a tremendous writing on appropriated learning and information mining , yet almost no of this writing centers around our objective: A gen-eral methods for programming AI on multicore. Quite a bit of this writing contains a long also, recognized convention of growing (regularly quick) approaches to accelerate or parallelize individ-ual learning calculations, for example fell SVMs . In any case, these yield no broad parallelization strategy for AI and, all the more

ISSN: 2233-7857 IJFGCN

practically, specific executions of famous calculations infrequently lead to across the board use. A few instances of progressively broad papers are: Caregeaet. al. give some broad information appropriation conditions for parallelizing AI, however confine the concentration to choice trees; Jin and Agrawal give a general AI programming ap-proach, yet just for shared memory machines. This doesn't fit the engineering of cell or lattice type multiprocessors where centers have neighborhood reserve, regardless of whether it tends to be progressively reallocated.

LITERATURE SURVEY:

Many programming frameworks are possible for the summation form, but inspired by Google's success in adapting a functional programming construct, map-reduce [7], for wide spread parallel programming use inside their company, we adapted this same construct for multicore use. Google's map-reduce is specialized for use over clusters that have unreliable communication and where indi- vidual computers may go down. These are issues that multicores do not have; thus, we were able to developed a much lighter weight architecture for multicores.

A high level view of our architecture and how it processes the data. The map-reduce engine is responsible for splitting the data by training examples (rows). The engine then caches the split data for the subsequent map-reduce invocations. Every algorithm has its own engine instance, and every map-reduce task will be delegated to its engine Similar to the original map-reduce architecture, the engine will run a master which coordinates the mappers and the reducers. The master is responsible for assigning the split data to different mappers, and then collects the processed intermediate data from the mappers After the intermediate data is collected, the master will in turn invoke the reducer to process it and return final results Note that some mapper and reducer operations require additional scalar information from the algorithms. In order to support these operations, the mapper/reducer can obtain this information through the query info interface, which can be customized for each different algorithm.

ISSN: 2233-7857 IJFGCN

PARALLEL PROGRAMMING METHOD:

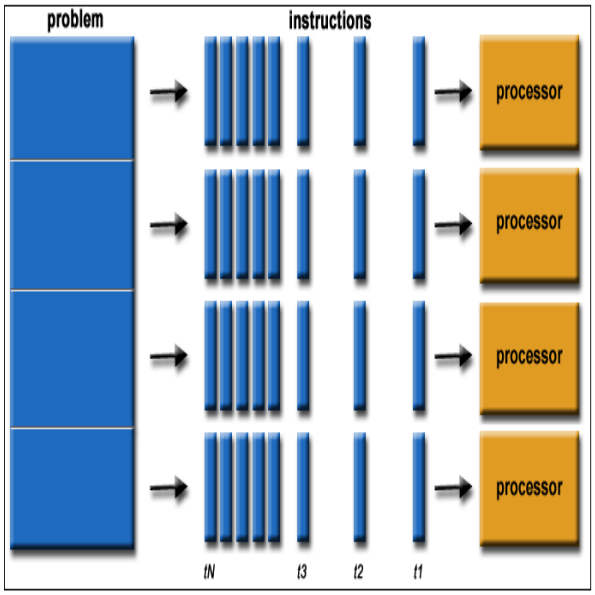

Fig:1- Parallel Programming Model

We will briefly discuss the algorithms we have implemented based on our framework. These algorithms were chosen partly by their popularity of use in NIPS papers, and our goal will be to illustrate how each algorithm can be expressed in summation form. We will defer the discussion of the theoretical improvement that can be achieved by this parallelization to Section 4.1. In the following, x or xi denotes a training vector and y or yi denotes a training label.the solution of the normal equations Aθ= b, where A =Σmmi=1wi(xixT) and b =set to compute subgroup wi(xiyi). Two reducers respectively sum up the partial values

for A and b, and the algorithm finally computes the solution θ = A−1b. Note that if wi = 1, the algorithm reduces to the case of ordinary least squares (linear regression).

Naive Bayes (NB) In NB [17, 21], we have to estimate P (xj = k y = 1), P (xj = k y = 0), and P (y) from the training data. In order to do so, we need to sum over xj = k forsubgroup 1 y = 1 and subgroup 1 y = 0 . The reducer then sums up intermediate results to get the final result for the parameters.

Gaussian Discriminative Analysis (GDA) The classic GDA algorithm [13] needs to learn the following four statistics P (y), μ0, μ1 and Σ. For all the summation forms involved in these computations, we may leverage the map-reduce framework to parallelize the process. Each mapper will handle the summation (i.e. Σ 1 yi = 1 , Σ 1 yi = 0 , Σ 1 yi = 0 xi, etc) for a subgroup of the training samples. Finally, the reducer will aggregate the intermediate sums and calculate the final result for the parameters.

ISSN: 2233-7857 IJFGCN

k-means In k-means [12], it is clear that the operation of computing the Euclidean distance between the sample vectors and the centroids can be parallelized by splitting the data into individual subgroups and clustering samples in each subgroup separately (by the mapper). In recalculating new centroid vectors, we divide the sample vectors into subgroups, com- pute the sum of vectors in each subgroup in parallel, and finally the reducer will add up the partial sums and compute the new centroids.

Logistic Regression (LR) For logistic regression [23], we choose the form of hypothesis as $h\theta(x) = g(\theta T\ x) = 1/(1 + \exp(\ \theta T\ x))$ Learning is done by fitting θ to the training data where the likelihood function can be optimized by using Newton-Raphson to update$\theta := \theta - H{-}1\nabla\theta A(\theta)$. $\nabla\theta A(\theta)$ is the gradient, which can be computed in parallel mappers summing up $\Sigma subgroup(y(i) - h\theta(x(i)))x$ each NR step i. The computation(i)of the hessian matrix can be also written in a summation form of $H(j, k) := H(j, k) + h\theta(x(i))(h\theta(x(i)) - 1)x(i)x(i)$ for the mappers. The reducer will then sum up the valuesfor gradient and hessian to perform the update for θ.Neural Network (NN) We focus on backpropagation [6] By defining a network struc- ture (we use a three layer network with two output neurons classifying the data into two categories), each mapper propagates its set of data through the network. For each train- ing example, the error is back propagated to calculate the partial gradient for each of the weights in the network. The reducer then sums the partial gradient from each mapper and does a batch gradient descent to update the weights of the network.

IMPLEMENTATION:

We led a broad arrangement of examinations to look at the accelerate on informational indexes of different sizes (table 2), on eight regularly utilized AI informational collections from the UCI Machine Learning store and two different ones from a [anonymous] research gathering (Helicopter Control and sensor information). Note that not all the examinations bode well from a yield see – relapse on unmitigated information – however our motivation was to test speedup so we ran each calculation over all the information.

ISSN: 2233-7857 IJFGCN

	single
LWLR	$O(mn^2 + n^3)$
LR	$O(mn^2 + n^3)$
NB	$O\ mn\quad nc$
NN	$O(mn + nc)$
GDA	$O(mn^2 + n^3)$
PCA	$O(mn^2 + n^3)$
ICA	$O(mn^2 + n^3)$
k-means	$O(mnc)$
EM	$O(mn^2 + n^3)$
SVM	$O(m^2n)$

Table:1-Time complexity survey

Data Sets	samples (m)	features (n)
Adult	30162	14
Helicopter Control	44170	21
Corel Image Features	68040	32
IPUMS Census	88443	61
Synthetic Time Series	100001	10
Census Income	199523	40
ACIP Sensor	229564	8
KDD Cup 99	494021	41
Forest Cover Type	581012	55
1990 US Census	2458285	68

Table:2-Dataset size and details

The speedup on dual processors over all the algorithms on all the data sets. As can be seen from the table, most of the algorithms achieve more than 1.9x times performance improvement. For some of the experiments, e.g. gda/covertype, ica/ipums, nn/colorhistogram, etc., we obtain a greater than 2x speedup. This is because the original algorithms do not utilize all the cpu cycles efficiently, but do better when we distribute the tasks to separate threads/processes.

Figure 2 shows the speedup of the algorithms over all the data sets for 2,4,8 and 16 processing cores. In the figure, the thick lines shows the average speedup, the error bars show the maximum and minimum speedups and the dashed lines show the variance.

ISSN: 2233-7857 IJFGCN

At last, the above are runs on multiprocessor machines. We wrap up by revealing some affirming outcomes and better on a restrictive multicore test system over the sensor dataset.2 NN speedup was [16 centers, 15.5x], [32 centers, 29x], [64 centers, 54x]. LR speedup was [16 centers, 15x], [32 centers, 29.5x], [64 centers, 53x]. Multicore machines are commonly quicker than multiprocessor machines since correspondence inside to the chip is substantially less exorbitant.

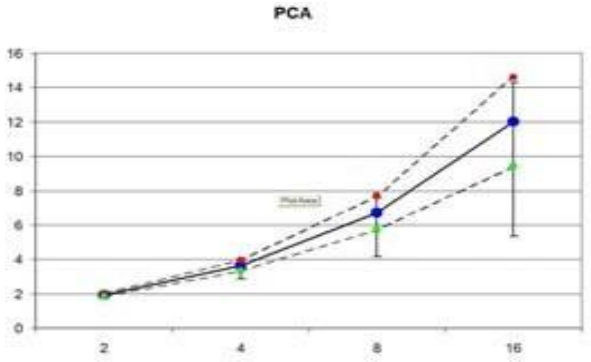

Fig:2-Show the speedup from 1 to 8 processors of all the algorithms over all the data sets.

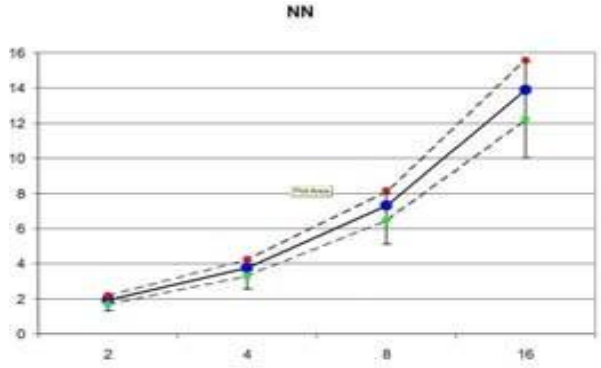

Fig:3-Show the speedup from 8 to 16 processors of all the algorithms over all the data sets.

CONCLUSION:

For AI to keep harvesting the abundance of Moore's law and apply to ever bigger datasets and issues, it is essential to embrace a programming design which exploits multicore. In this paper, by exploiting the summation structure in a guide diminish system, we could parallelize a wide scope of AI calculations and accomplish a 1.9 occasions speedup on a double processor on up to multiple times speedup on 64 centers. These outcomes are in accordance with the multifaceted nature investigation in Table 1. We note that the speedups accomplished here included no unique improvements of the calculations themselves. Hence the proposed methods for parallel programming that direct relapse k-implies, strategic regression innocent Bayes, Gaussian discriminant examination and back propagationhas been described.

ISSN: 2233-7857 IJFGCN

REFERENCES:

[1] Sejnowski TJ. Bell AJ. An information-maximization approach to blind separation and blind deconvolution. In Neural Computation, 1995.

[2] O. Chapelle. Training a support vector machine in the primal. Journal of Machine Learning Research (submitted), 2006.

[3] W. S. Cleveland and S. J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. In J. Amer. Statist. Assoc. 83, pages 596–610, 1988.

[4] L. Csanky. Fast parallel matrix inversion algorithms. SIAM J. Comput., 5(4):618–623, 1976.

[5] A. Silvescu D. Caragea and V. Honavar. A framework for learning from distributed data using sufficient statistics and its application to learning decision trees.

International Journal of Hybrid Intelligent Systems, 2003.

[6] R. J. Williams D. E. Rumelhart, G. E. Hinton. Learning representation by back-propagating errors. In Nature, volume 323, pages 533–536, 1986.

[7] J.Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. Operating Systems Design and Implementation, pages 137–149, 2004.

[8] N.M. Dempster A.P., Laird and Rubin D.B.

[9] D.J. Frank. Power-constrained cmos scaling limits. IBM Journal of Research and Development, 46, 2002.

[10] P. Gelsinger. Microprocessors for the new millennium: Challenges, opportunities and new frontiers. In ISSCC Tech. Digest, pages 22–25, 2001.

[11] Leon Bottou Igor Durdanovic Hans Peter Graf, Eric Cosatto and VladimireVapnik. Parallel support vector machines: The cascade svm. In NIPS, 2004.

[12] J. Hartigan. Clustering Algorithms. Wiley, 1975.

[13] T. Hastie and R. Tibshirani. Discriminant analysis by gaussian mixtures. Journal of the Royal Statistical Society B, pages 155–176, 1996.

[14] R. Jin and G. Agrawal. Shared memory parallelization of data mining algorithms: Techniques, programming interface, and performance. In Second SIAM International Conference on Data Mining,, 2002.

ISSN: 2233-7857 IJFGCN