A Data Partition Method for Parallel Self-Organizing Map

Ming-Hsuan Yang and Narendra Ahuja
Department of Computer Science and Beckman Institute
University of Illinois at Urbana-Champaign
Urbana, IL 61801
Email:{mhyang, ahuja}@vision.ai.uiuc.edu
Web Page: http://vision.ai.uiuc.edu

Abstract

We propose a method to partition training vectors into clusters for a parallel implementation of Self-Organizing Map (SOM) algorithm. The proposed algorithm assigns a cluster to a processor such that, in updating weights, the neighborhoods of a winning node in a cluster do not overlap the neighboring nodes of some winning nodes in other clusters. It reduces the overheads caused by synchronization (i.e. maintaining coherency) of the weight matrices in the processors since the proposed algorithm allows multiple vectors to find their winning nodes and update weights in parallel. Our experimental results show that an average speedup of 3.15 for a parallel implementation of a four-processor simulation.

1 Introduction

The Self-Organizing Map (SOM) algorithm has attracted much of interest among researchers from computer vision, image compression, to information retrieval. SOM has proved to be one of the most robust and useful algorithms for classification and dimensionality reduction. However, the computation involved in the algorithm is extremely high, especially for the problem in which the number of input vectors and the size of the map are large. For example, it takes supercomputers several days to compute a large map in a large scale text classification experiment[18]. Therefore, it is necessary and important to develop efficient parallel SOM algorithm for real-world applications.

Many algorithms have been proposed to parallelize Kohonen's SOM algorithm. One type of the parallel algorithms is to use the characteristics of the architecture. Many parallel SOM algorithms have been proposed on different architecture, such as transputer [1] [23], eight neighbor processor array [22], connection machine [20], parallel coprocessor [17], single instruction multiple data machine [11], multiple instruction multiple data machine [26], systolic array [10] [25], and recently PVM [2] [7]. Another type of parallel SOM algorithm is to partition the data into several chunks such that these chunks can be executed by different processors in parallel. The advantage of this approach over the machine-dependent algorithm is its universal applicability [13] [27] [8] [4] [19] [3] [21].

For a good statistical accuracy, the complete learning process may require an a appreciable number, say, 100,000 steps. However, the number of available samples is usually much smaller than the required steps, it is obvious that the samples must be reused iteratively. Several alternatives have been proposed: the sample may be applied cyclically or in a randomly permuted order, or picked up at random from the basic set (i.e. bootstrap learning). It turns out in practice that order cyclic application is not noticeably worse than the order, mathematically better justifiable methods [6].

The proposed algorithm is based on the observation of the dynamic behavior and the characteristics of the neighborhood function in Kohonen's SOM algorithm. Kohonen points out that the period during which a rough order in the SOM is obtained is usually relatively short, on the order of 1000 steps [6]. Whereas most of the computing time is spent for the final convergence phase, in order to achieve a sufficiently good statistical accuracy. Furthermore, the starting neighborhood function, $N_c(0)$, is recommended to be at least half of the largest diagonal of the map for the ordering phase. Meanwhile, the neighborhood function in the second phase decreases with respect to time, i.e. $N_c(t) \leq N_c(t-1)$.

Let N be the number of processors, I be the number of input vectors and M^2 be the number of nodes in the map, the idea is to partition the input vectors into N clusters such that a processor can take one cluster as its own inputs and all the processors can form their maps concurrently. Meanwhile, we also want to balance the load of a processor as much as possible, i.e. every processor has approximately $\frac{I}{N}$ input vectors. In other words, the proposed algorithm decomposes the original problem into a set of subproblems such that these subproblems can be solved simultaneously (i.e. divide-and-conquer). The proposed algorithm is developed based on the observation that a rough order in the SOM is obtained after a short ordering period, We divide the map into N submaps equally (i.e. a submap has $\frac{M^2}{N}$ nodes) and cluster the input vectors whose winning nodes are in ev-

0-7803-5529-6/99/$10.00

ery submap at the end of the ordering phase. Consequently, we divide the original problem into N subproblems. Every subproblem has roughly $\frac{I}{N}$ inputs for a $\frac{M^2}{N}$ nodes and these subproblems can be solved simultaneously because they are roughly independent.

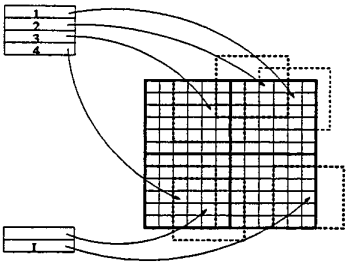
Nevertheless these processors need to communicate with others to keep consistency of the map because the neighboring nodes of a winning node may belong to a submap assigned to other processor. In other words, one processor needs to inform the other processors to update the weights of their nodes when the neighboring function of the winning node contains nodes in other processors. The hope of the proposed algorithm is that we can partition the problem such that the frequency of such case is rare.

2 Proposed Method

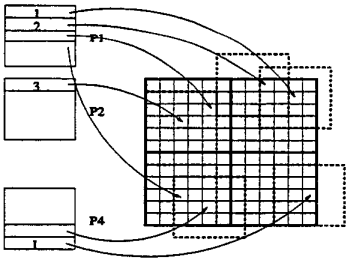
Kohonen's SOM [6] is usually trained in two phases. The first of them is the ordering phase (coarse training) during which the reference vectors (i.e. weight vectors) are roughly ordered. During the second phase the values of the reference vectors are fine tuned. In other words, the reference vectors in a node converge to their "correct" values. The second phase is usually much longer (more than 10 times) than the first phase. Also, the neighborhood radius is also smaller than that in the first phase.

The proposed data partition algorithm is based on the observation that the mapping from the input vectors to the nodes on the map stabilizes after the first phase and the radius of the neighborhood function is smaller than that in the first phase. Let N be the number of processors in the parallel machine and M^2 be the size of a rectangular map, we divide the map into N non-overlapping clusters where a cluster has $\frac{M^2}{N}$ nodes. The training vectors are then grouped into N clusters based on the mapping after the ordering phase. The map itself is relatively stable after the first phase, i.e. the winning nodes of an input vector in two consecutive epochs are close to each other. In the second phase, the vectors in every cluster can be presented to the map in parallel since they, with high probability, will be mapped to the same node as the first phase or neighboring nodes within the same cluster. If an input vector is mapped to a node in another cluster, then it is necessary to send the newly updated weights to other processors to keep the coherency of the weigh matrices. However, the proposed data partition reduces the occurrence of such mapping and thus reduces the overheads caused by synchronization of weight matrices.

Figure 1 summarizes the idea of the proposed method. At the beginning phase of ordering, the winning node of an input vector can be anywhere on the map as shown in Figure 1(a). At the end of ordering phase, the reference vectors (i.e. weight vectors) are ordered. The proposed method then clusters the input vectors whose winning nodes fall in a submap and divides the original map into N subproblems. In the second phase, the reference vectors of the nodes in the submap are being fine tuned. In other words, the winning nodes of an input vector do not change much between epochs, which means that the winning nodes tend to fall in the same cluster as shown in Figure 1(b). The main idea of the proposed method is to divide the original problem into N subproblems such that all of them can be processed in parallel, except the necessary inter process communication when the neighboring nodes of the winning node of an input vector are not within the same submap.



(a) At beginning of the ordering phase, the neighborhood of a winning node of an input vector appears anywhere on the original map and the neighborhoods of winning nodes tend to overlap each other.



(b) After ordering phase, the neighborhood of a winning node becomes smaller and tend to localize in one submap.

Figure 1: Interactions of the neighborhoods of winning nodes in the ordering and convergence phases.

3 Experimental Results

We simulate the partition method on a SGI machine with 4 processes on a image coding problem in that the task is to

map 2048 training vectors into a 8×8 SOM map[1]. Our experimental results show that the probability that we need to synchronize the weight matrices is roughly 5%. The speed up, compared with a conventional SOM algorithm running on a single processor, is 3.15. The experimental results show that the proposed method achieves good speedup.

Figure 2 shows the decompressed image using SOM for compression. Note that the results from parallel implementation is the same as those from conventional SOM. In other words, the proposed method does not trade in performance for speedup. On the other hand, we will discuss some methods to further increase the speedup if the performance (e.g. quality of the decompressed image) is not the highest priority.

4 Discussion and Conclusion

We present a partition method that reduces the overheads caused by synchronizing the weight matrices of different processors. Experimental results show that our method has a linear speedup close to N where N is the number processors.

Although we have demonstrated the potential of the proposed method, we have not gathered good results on PVM. We conjecture that there is some technical problems on the synchronization functions in the PVM library. Our future work will include implementations and experiments on PVM as well as SGI Power Challenge. Furthermore, we will also verify our algorithm with experiments on parallel machines with different number of processors and different problems.

The proposed algorithm can be modified to further increase speedup. One possible way to improve the speedup is to first sort the input vectors in ascending order such that the ordering phase may take less time. However, it is not known how this will affect the resulting map. For example, it is clear how sorting affects the quality of the decompressed images in our experiments. Another possible way to speed up the learning process is to disallow inter process communication while suffers certain degraded performance. However, it can be used when a very large dataset needs to be trained promptly and the quality of the map is not critical. We will explore these methods in a large scale experiments to better understand the effects of ordering and inter process communications.

References

[1] J. M. Auger. Parallel implementation on transputer of Kohonen's algorithm. In D. Gassilloud and J. C. Grossetie, editors, *Computing with Parallel Architectures: T. Node*, pages 215–226, Dordrecht, Netherlands, 1991. Kluwer.

[2] H. Guan, C. kwong Li, T. yat Cheung, and S. Yu. Parallel design and implementation of SOM neural computing model in PVM environment of a distributed system. In *Proceedings of Advances in Parallel and Distributed Computing (Cat. No. 97TB100099)*, pages 26–31. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1997.

[3] T. Hamalainen, H. Klapuri, J. Saarinen, and K. Kaski. Mapping of SOM and LVQ algorithms on a tree shape parallel computer system. *Parallel Computing*, 23(3):271–89, 1997.

[4] H. Heiss and M. Dormanns. Partitioning and mapping of parallel programs by self-organization. *Concurrency: Practice and Experience*, 8(9):685–706, 1996.

[5] T. Kohonen. Internal representations and associative memory. In R. Eckmiller, G. Hartman, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 177–182. Elsevier, Amsterdam, Netherlands, 1990.

[6] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen, and K. Torkkola. LVQ_PAK: The Learning Vector Quantization program package. Report A30, Helsinki University of Technology, Laboratory of Computer and Information Science, Jan. 1996.

[7] J. S. Lange, P. Schonmeier, and H. Freiesleben. Parallelization of analyses using self-organizing maps with PVM. *Nuclear Instruments and Methods in Physics Research A*, 389:274–76, 1997.

[8] T. Li, S. Klasa, and Y. Y. Tang. Data mapping for parallel programs with changing size windows. In *Seventh International Conference on Parallel and Distributed Computing Systems*, pages 640–3. Int. Soc. Comput. & Their Appl. -ISCA, Raleigh, NC, USA, 1994.

[9] T. Li and L. Tao. Topological feature maps on parallel computers. *International Journal of High Speed Computing*, 7(4):531–46, 1995.

[10] R. Mann and S. Haykin. A parallel implementation of Kohonen's feature maps on the warp systolic computer. In *Proc. IJCNN-90, Int. Joint Conf. on Neural Networks, Washington, DC*, volume II, pages 84–87, Hillsdale, NJ, 1990. Lawrence Erlbaum.

[11] M. Manohar and J. C. Tilton. Progressive vector quantization on a massively parallel SIMD machine with application to multispectral image data. *IEEE Trans. on Image Processing*, 5(1):142–147, January 1996.

[12] G. Myklebust and J. G. Solheim. Parallel self-organizing maps for actual applications. In *Proc. ICNN'95, IEEE Int. Conf. on Neural Networks*, volume II, pages 1054–1059, Piscataway, NJ, 1995. IEEE Service Center.

[13] K. Obermayer, H. Ritter, and K. Schulten. Large-scale simulation of a self-organizing neural network: Formation of a somatotopic map. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 71–74, Amsterdam, Netherlands, 1990. North-Holland.

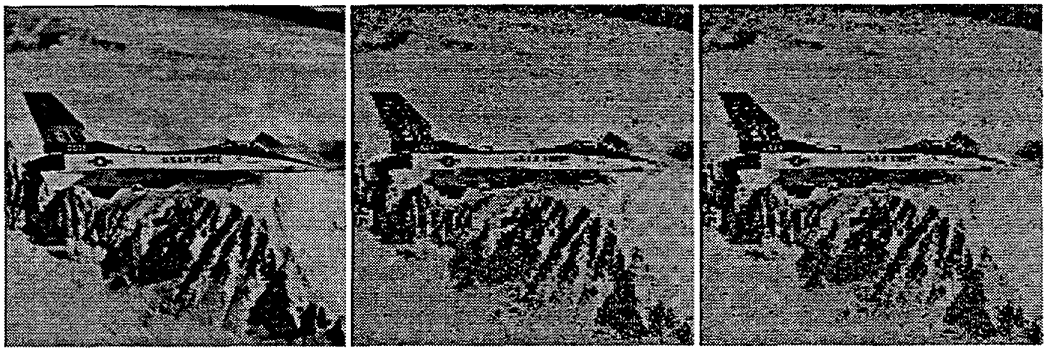
[14] S. Openshaw and I. Turton. A parallel Kohonen algorithm for the classification of large spatial datasets. *Computers & Geosciences*, 22(9):1019–26, 1996.

[15] K. K. Parhi, F. H. Wu, and K. Genesan. Sequential and parallel neural network vector quantizers. *IEEE Transactions on Computers*, 43(1):104–9, Jan 1994.

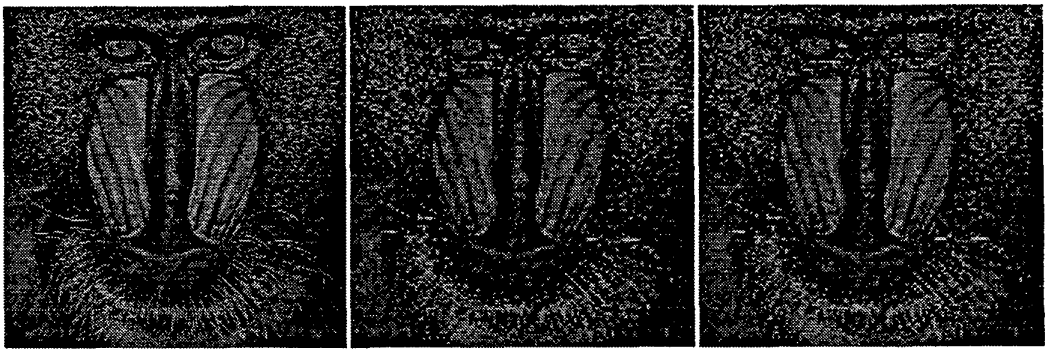
[16] J. W. Quittek. Optimizing parallel program execution by self-organizing maps. *Journal of Artificial Neural Networks*, 2(4):365–80, 1995.

[1] We tested the same algorithm on a PVM of 4 Pentium machines. However we feel that the inter process communication functions in PVM is not well implemented since it takes much time in communicating with other nodes. Thus, we only discuss the simulated results in this paper.

(a) original image: "lena" (b) decompressed image using conventional SOM for compression (c) decompressed image using parallel SOM for compression



(d) original image: "airplane" (e) decompressed image using conventional SOM for compression (f) decompressed image using parallel SOM for compression



(g) original image: "baboon" (h) decompressed image using conventional SOM for compression (i) decompressed image using parallel SOM for compression

Figure 2: Simulation results using the conventional and parallel SOM algorithms.

[17] J. Saarinen, M. Lindroos, J. Tomberg, and K. Kaski. Parallel coprocessor for Kohonen's self-organizing neural network. In V. K. Prasanna and L. H. Canter, editors, *Proceedings of the Sixth International Parallel Processing Symposium (Cat. No. 92TH0419-2)*, pages 537–42, Los Alamitos, CA, USA, 1992. IEEE Comput. Soc. Press.

[18] B. Schatz, W. Mischo, T. Cole, A. Bishop, S. Harum, E. Johnson, L. Neumann, H. Chen, and D. Ng. Federated search of scientific literature. *IEEE Computer*, 32(2):51–59, 1999.

[19] E. Schikuta and C. Weidmann. Data parallel simulation of self-organizing maps on hypercube architectures. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6*, pages 142–147. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[20] A. Singer. Implementations of artificial neural networks on the connection machine. *Parallel Computing*, 14:305–315, 1990.

[21] D. Strupl and R. Neruda. Parallelizing self-organizing maps. In F. Plasil and K. G. Jeffery, editors, *SOFSEM '97: Theory and Practice of Informatics. 24th Seminar on Current Trends in Theory and Practice of Informatics. Proceedings*, pages 563–70. Springer-Verlag, Berlin, Germany, 1997.

[22] T. Takeda, A. Tanaka, and K. Tanno. Parallel computing algorithm of neural networks on an eight-neighbor processor array. In *Twelfth Annual International Phoenix Conference on Computers and Communications (Cat. No. 93CH3249-0)*, pages 559–64, New York, NY, USA, 1993. IEEE.

[23] R. Togneri and Y. Attikiouzel. Parallel implementation of the Kohonen algorithm on transputer. In *Proc. IJCNN-91, Int. Joint Conf. on Neural Networks, Singapore*, volume II, pages 1717–1722, Los Alamitos, CA, 1991. IEEE Comput. Soc. Press.

[24] D. E. Van den Bout and T. K. Miller III. TInMANN: the integer Markovian artificial neural network for performing competitive and kohonen learning. *Journal of Parallel and Distributed Computing*, 25(2):107–14, March 1995.

[25] N. Vassilas and P. Thiran. On modifications of Kohonen's feature map algorithm for an efficient parallel implementation. In *ICNN 96. The 1996 IEEE International Conference on Neural Networks (Cat. No. 96CH35907)*, volume 2, pages 1390–1394. IEEE, New York, NY, USA, 1996.

[26] L. Vuurpijl, T. Schouten, and J. Vytopil. Performance prediction of large MIMD systems for parallel neural network simulations. *Future Generation Computer Systems*, 11(2):221–32, March 1995.

[27] C. H. Wu, R. E. Hodges, and C. J. Wang. Parallelizing the self-organizing feature map on multiprocessor systems. *Parallel Computing*, 17(6-7):821–832, September 1991.