Massively parallel implementation of sparse message retrieval algorithms in Clustered Clique Networks

Philippe Tigréat, Pierre-Henri Horrein, Vincent Gripon

Massively Parallel Implementation of Sparse Message Retrieval Algorithms in Clustered Clique Networks

Philippe Tigréat, Pierre-Henri Horrein, Vincent Gripon
Electronics Department, Telecom Bretagne
Brest, France
Email: {philippe.tigreat, ph.horrein, vincent.gripon}@telecom-bretagne.eu

***Abstract*—Auto-associative memories are a family of algorithms designed for pattern completion. Many of them are based on neural networks, as is the case for Clustered Clique Networks which display competitive pattern retrieval abilities. A sparse variant of these networks was formerly introduced which enables further improved performances. Specific pattern retrieval algorithms have been proposed for this model, such as the Global-Winners-Take-All and the Global-Losers-Kicked-Out. We hereby present accelerated implementations of these strategies on graphical processing units (GPU). These schemes reach interesting factors of acceleration while preserving the retrieval performance.**

Keywords*—*Clustered Clique Networks, Parallel processing, CUDA, GPGPU.

I. Introduction

Associative memories are a type of computer memories that are accessed by content instead of address. Hetero-associative memory link different patterns together by pairs, and when requested with one pattern they return the associated one. Auto-associative memories link together the subparts of a pattern, and perform pattern completion when submitted a subset of a stored memory element.

Among the latter category are models such as Hopfield [1] and Willshaw [2]. Both models are based on neural networks. Hopfield neural networks come with a fully-interconnected set of neurons, and the connection weights are adapted to the set of stored messsages so that each message becomes a local minimum of the network's energy function. Willshaw networks differ from this scheme in that their connectivity is very sparse, and only when two neurons are part of a same message do they get connected. This characteristic grants Willshaw networks with a greater storage capacity than Hopfield, meaning they allow a better message retrieval success rate on average for given numbers of neurons and stored messages.

Clustered Clique Networks are another model of associative memories originally developed by Gripon and Berrou [3]–[6]. They leverage further benefits from sparseness than Willshaw, as a result of a clustering of the neurons set. Neurons are parted in equal sized clusters and intra-cluster connections are prohibited. In the first published version of these networks, a stored message is supported by a set of neurons with one picked in each cluster of the network. This fully-interconnected pattern is called a clique. Aliabadi *et al* [7] has investigated the storage of sparse messages in these networks, meaning messages composed by neurons located in a subset of the network's clusters. This specific case can benefit from an adaptation of the retrieval procedure used in classic CCNs. Aboudib *et al* [8] introduces two such variants of the Winner-Take-All algorithm. These are the Global Winners-Take-All (GWsTA) and the Global Losers-Kicked-Out (GLsKO), and come with a significant improvement in retrieval success rate.

The present work focuses on the acceleration of these algorithm variants by means of Graphical Processing Units (GPUs). These chipsets allow the massively parallel computation of operations on data, and are currently widely used to accelerate neural network implementations. To achieve this, one must ensure a sufficient amount of operations to execute can actually be performed in parallel without altering the accuracy of the whole computation. The main requirement for this is the independence of the parallelly processed data. We aim at making clear here how we adapt the former serial implementation under this constraint. We use the Compute Unified Device Architecture (CUDA) which is a coding framework specifically designed to program GPUs, supported by C/C++. Yao [9], [10] formerly achieved acceleration of the classic CCN algorithm using CUDA.

The paper is organized as follows. Section II summarizes the Clustered Clique Network model and algorithm, and its adaptation to sparse messages. Section III describes the strategies we use to adapt the specific retrieval algorithms for sparse messages on a massively parallel architecture. Section IV presents the results we achieve in terms of acceleration and message retrieval performance depending on different parameters of the network.

II. CCNs

A. Structure

Clustered Clique Networks (CCN) are composed of a set of N neurons, divided in clusters all containing the same number of neurons. A connection can be drawn between two of these neurons only if they belong to different clusters. A memory element is then supported by a choice of neurons all located in different clusters. The storage of such a message in the network consists in connecting all of its neurons together. The pattern of connections so created is called a clique.

By the separation of the network into clusters, it makes use of sparseness to a higher extent than former states-of-the-art like Willshaw networks, thus lessening the amount of overlapping between stored messages and easing the search of the unknown elements of patterns to complete. Hence, CCN display a high storage capacity and the number of messages they are able to store and retrieve successfully grows quadratically as a function of the number of neurons in the network.

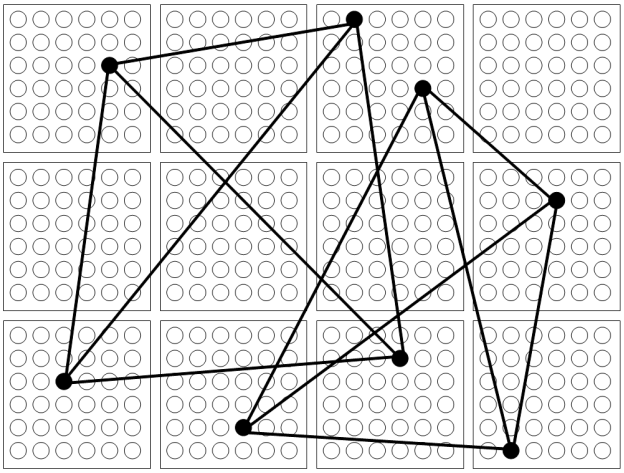

Figure 1. Example of a sparse Clustered Clique Network with 12 clusters and messages of order 4.

During retrieval, the commonly used procedure consists in propagating activation from the neurons of a request through all their existing connections to other neurons, followed by a selection of neurons based on the resulting activity scores. A local Winner-Takes-All rule is generally used to perform the selection. It consists in keeping active only the neurons possessing the highest activity score in each cluster. This is an iterative process, and it is repeated as long as the stopping criterion is not reached. Activation is computed again with the selected neurons, and the rule is applied once again.

B. Sparse Messages

Sparse Clustered Clique Networks as introduced by Aliabadi *et al* [7] have the same principle as classic CCNs. The main difference is that stored messages no longer need to possess a neuron in every cluster of the network, but only in a selected subset of the clusters. This characteristic raises the diversity of messages, that is the total number of different messages one can store in the network. However, it brings the added difficulty of not knowing which clusters support the searched message. The high level of sparsity usually makes up for this, and the retrieval error rates are much lower in general as compared to full messages CCNs. Figure 1 shows an example of such a network.

C. Specific Retrieval Algorithms for Sparse Messages

The retrieval of sparse messages can be performed more efficiently by using strategies specifically tailored for this case. These will typically make use of the set of activity scores observed after stimulation instead of solely focusing on local winners. Two such strategies can be found in [8] namely the Global Winners-Take-All and the Global Losers-Kicked-Out.

In Global Winners-Take-All, one wants to retain a number of neurons equal or superior to a predefined minimum. The minimal threshold score for a fanal to be retained is adapted in order to fulfill this requirement. After the first activity computation step, we iteratively select a number of fanals which are then used to compute activity for next iteration. For instance, if after a stimulation of the network, there are 7 activated fanals with scores {2,3,1,4,3,4,4} and we know at least 4 neurons must be retained, a lower threshold of 3 will be applied on the scores of neurons to remain active. This way the 5 neurons with an activity of 3 or more are kept as output for the ongoing iteration, while the 2 neurons with scores 1 and 2 are discarded.

In Global Losers-Kicked-Out, focus is put on active neurons with relatively low scores. After the initial activity computation, the aim is to discard a number of fanals iteratively, instead of selecting them. Among the non-zero scores observed after stimulation, a fixed number of lower score values is set so that neurons with these activity levels are put to zero. In the previous example, if we know 3 lower score values must be discarded, this will designate the 4 neurons in the list with scores {2,1,3,3} and only the 3 neurons with an activity of 4 will remain active. Nonetheless, it has been shown that banning only the one minimal score in the list gives the best results.

III. Accelerations

A. GPU usage

Graphics Processing Units (GPU) are highly parallel processors. They are based on a high number of processing units, which can be used to process data using a Single Instruction Multiple Data (SIMD) paradigm. In this paradigm, the same instruction can be applied simultaneously on all elements of a vector. As a result, GPU can be very efficient when processing algorithm without data dependency. For example, loops in which each iteration can be performed regardless of previous iterations can be very efficiently accelerated. In order to fully benefit from the GPU, the selection algorithms have been adapted to follow this paradigm.

B. GWsTA

Global Winners-Take-All aims at retaining a number of active fanals superior or equal to a predefined minimum. In order to do this, we compute for each possible score value the number of fanals that reach it. In practice, scores counts are first computed in parallel for the different clusters, with one thread computing the counts for the different scores inside of one cluster, as shown on Figure 2. In a second step, the scores counts are accumulated by reduction to get their totals over the whole network. A number of threads is launched, equal to half the number of clusters. Each thread sums the vectors of scores counts of two clusters located in different halves of the network. The operation is repeated on the new set of counts lists and so forth, each time halving the number of computed values until one single list remains with the global sums. A final thread then runs a loop over the possible score values in descending order, accumulating scores counts until the sum equals or exceeds the minimum expected number of active neurons. As soon as this criterion is met, looping stops and the last considered score value is kept as the minimum score for nodes to remain active. A thresholding with binarization is then applied on nodes activities, with scores above the

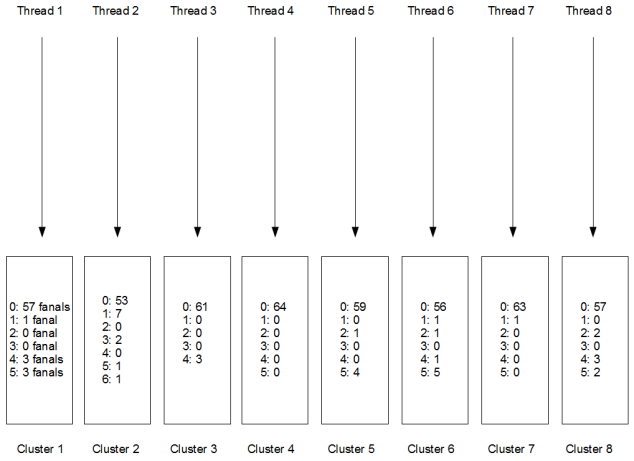

Figure 2. Example of the parallel computation of local scores counts in the different clusters. Each thread computes a vector of the numbers of fanals possessing the different possible score values in one cluster.

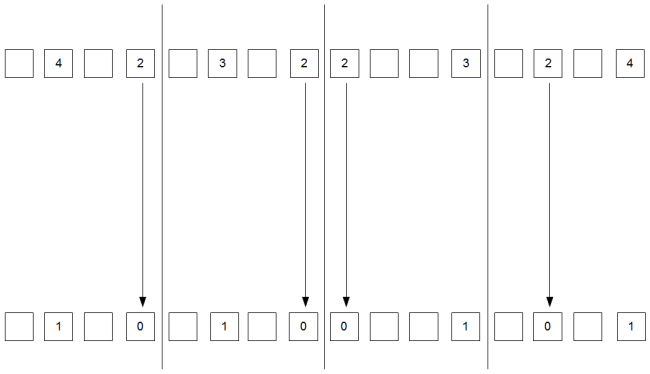

Figure 3. Example of the threshold-binarize operation as performed on GPU. Non-zero scores below the threshold are put to 0 while those above it are set to 1. Input and output values are in one-to-one relationships, hence computation can be performed in parallel.

threshold being put to one while other neurons are all at zero. This operation is performed in parallel, where each thread computes the output values for one neuron as illustrated by Figure 3. The resulting set of active fanals is then returned as output for the current iteration.

C. GLsKO

For our implementation of the Global Losers-Kicked-Out we set to 1 the number of low scores to be discarded. This is known to give the best results, and it reduces the complexity of the algorithm, since we only need to find the minimum score and discard neurons with this score. Hence we focus on finding the global minimum among all non-zero scores present after stimulation. To this end the minimum positive scores inside of the different clusters are first computed in parallel.

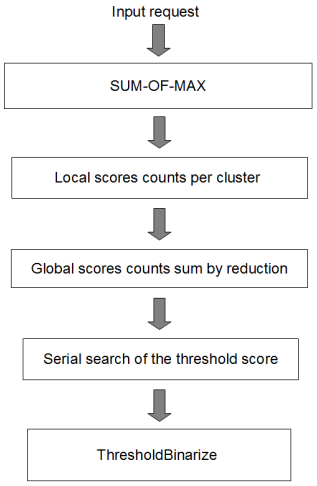

Figure 4. Illustration of the GWsTA procedure after the first iteration of retrieval. All operations in rectangular boxes are performed on GPU.

A reduction strategy is employed on the resulting set of local minimums to compute the global one. At every step of the reduction, each thread compares the minimums obtained for two clusters, keeping the lowest of the two values. The set of values is thus halved iteratively to result in a single minimum. Along with the strictly positive minimum, a global boolean value is computed indicating the presence of several different non-zero scores, as described in Figure 5. Here the fact that several threads modify the output boolean does not alter its accuracy, as it indicates whether two different non-zero scores are seen at least once. If this boolean is true at the end of the reduction procedure, the ThresholdBinarize operation is applied to compute the iteration output, setting off all nodes with the non-zero minimum score. Otherwise, only one strictly positive score is present and all its representatives are kept active. This case where all stimulated fanals have equal scores is also a good stopping criterion for the GLsKO.

IV. Experiments

Our GPU experiments were performed with an NVIDIA GeForce 780 Ti, with a frequency of 900MHz and 3GB of memory. The CPU used is a 3.3GHz Intel Core i7. Each message is generated randomly by first picking a fixed size subset of the available clusters, and then choosing one fanal in each selected cluster. In the retrieval phase, requests are formed from stored messages by dropping out a fix number of their nodes.

Figure 7 shows the execution times in seconds for the retrieval of 500 stored messages as a function of the number of erasures per queries, in networks with 16 clusters and 32 fanals per cluster. Each message is made of 12 neurons and the corresponding query is a subsample from the message with a size going from 11 down to 1.

The execution times are very close at the beginning and differ when the number of erasures exceeds about half the message size, due to the different stopping criteria employed. For higher number of erasures the GLsKO takes more and

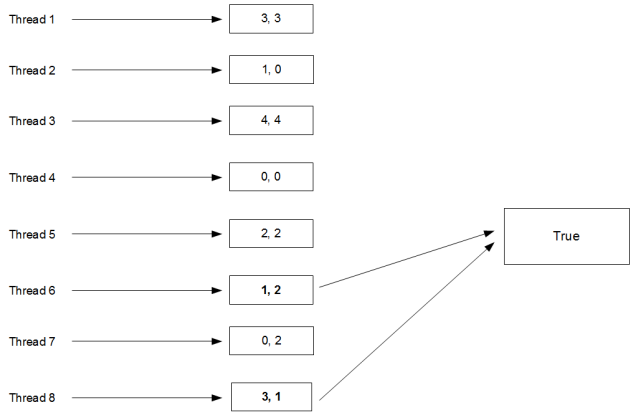

Figure 5. Illustration of the operation SeveralScoresPresent performed in the parallel implementation of GLsKO. This operation is performed in the same threads as the search of the non-zero minimum by reduction among local minimums. Each thread compares the minimum scores of two clusters to assess whether they are different and strictly positive. Several threads may access and modify the output boolean without affecting the result's accuracy.

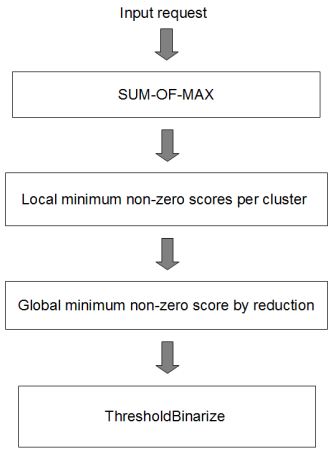

Figure 6. Illustration of the GLsKO procedure after the first iteration of retrieval. The global minimum non-zero score is computed by reduction, along with a boolean indicating if several non-zero scores are present. The output is computed using the parallel ThresholdBinarize operation, excluding the lowest non-zero score in the latter case.

more time, with a maximum when the request is made of a single neuron. The stopping criterion used is the equality of scores of all neurons active after stimulation. The GWsTA scheme takes less time for higher numbers of erasures. Its stopping criteria are convergence, *i.e.* equality between input and output at a given iteration, and a maximum number of iterations, hereby 8.

Figure 8 shows the execution time of the two schemes for networks of 64 clusters of 724 fanals each and 500 stored messages. Here messages are made of 56 nodes and the number of erasures applied to create the corresponding queries varies from 2 to 54. The two schemes make no retrieval error

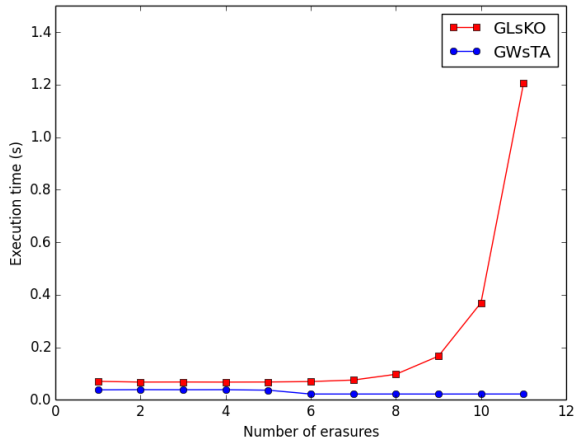

Figure 7. Execution time of the parallel versions of GWsTA and GLsKO for the retrieval of 500 sparse messages as a function of the number of erasures per query, in a network of 16 clusters and 32 fanals per cluster. Messages are made of 12 nodes and the size of sub-sampled queries goes from 11 nodes to 1.

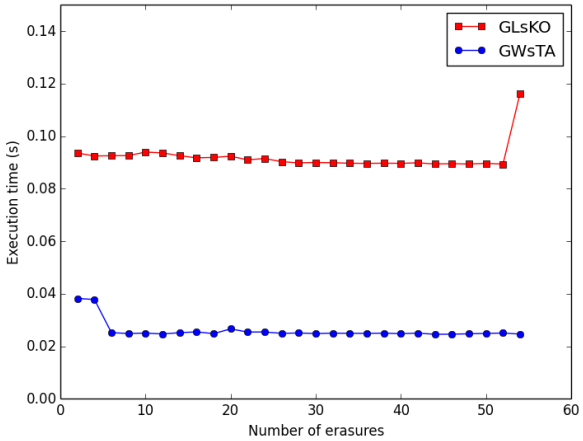

Figure 8. Parallel execution times of the two schemes for the retrieval of 500 messages of 56 nodes in a network with 64 clusters and 724 fanals per cluster. Queries size goes from 54 down to 2.

except for the GLsKO which fails to retrieve one pattern out of 500 when provided with 2-nodes requests. The number of iterations is impacted and alters the speed of retrieval.

With 2 erasures the GLsKO scheme takes 0.093573 seconds to complete retrieval of the whole set of messages, when a serial implementation of the same algorithm takes 4.74341 seconds running on CPU. This brings an acceleration factor of 50.69. Parallel and serial versions of the GWsTA use respectively 0.0246 seconds and 4.16881 seconds in the case with 2-nodes requests, giving an acceleration of 169.319. To be fair, it has to be stated that our serial and parallel implementations are not tailored to leverage a high sparseness of the connectivity graph, which could bring further improvements. To this end, one would want to represent the set of connections as lists of indexes instead of a boolean matrix.

V. Summary

We present parallel implementations of two specific strategies for the retrieval of sparse messages in Clustered Clique Networks. The accelerations we obtain are interesting and bring prospects for the use of these schemes in real-world applications, such as machine learning. Future work may indeed focus on parallel implementations for the retrieval of sequences in Clustered Clique Networks, with practical trials on high-scale datasets. Using Clustered Clique Networks in conjunction with Deep Learning for image classification is also one of our main directions of research.

Acknowledgements

This work was supported by the European Research Council under Grant ERC-AdG2011 290901 NEUCOD.

The authors would like to thank NVIDIA for providing us with a free graphics card allowing to speed up computations for the experiments performed during this work.

References

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Proceedings of the national academy of sciences, vol. 79, no. 8, 1982, pp. 2554–2558.

[2] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," Nature, 1969, pp. 960–962.

[3] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," Neural Networks, IEEE Transactions on, vol. 22, no. 7, 2011, pp. 1087–1096.

[4] B. Larras, C.Lahuec, M. Arzel, and F. Seguin, "Analog implementation of encoded neural networks," In Circuits and Systems (ISCAS), 2013 IEEE International Symposium on, 2013, pp. 1612-1615.

[5] ——, "A simple and efficient way to store many messages using neural cliques," Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2011 IEEE Symposium on, 2011, pp. 1–5.

[6] ——, "Nearly-optimal associative memories based on distributed constant weight codes," Information Theory and Applications Workshop (ITA), 2012, pp. 269–273.

[7] B. K. Aliabadi, C. Berrou, and V. Gripon, "Storing sparse messages in networks of neural cliques," Proceedings of the national academy of sciences, vol. 25, no. 5, 2014, pp. 461–482.

[8] A. Aboudib, V. Gripon, and X. Jiang "A study of retrieval algorithms of sparse messages in networks of neural cliques," arXiv preprint arXiv:1308.4506., 2013.

[9] Z. Yao, V. Gripon, and M. Rabbat, "A GPU-based associative memory using sparse Neural Networks," High Performance Computing and Simulation (HPCS), 2014 International Conference on, 2014, pp. 688–692.

[10] Z. Yao, V. Gripon, and M. Rabbat, "A massively parallel associative memory based on sparse neural networks," arXiv preprint arXiv:1303.7032., 2013