Une implémentation GPU de l'algorithme FlyHash: Des mouches plus rapides pour la fouille de données massives

Arthur da Cunha, Emanuele Natale, Damien Rivet, Aurora Rossi

Une implémentation GPU de l'algorithme FlyHash: Des mouches plus rapides pour la fouille de données massives

Arthur da Cunha[1], Emanuele Natale[1], Damien Rivet[1] et Aurora Rossi[1]

[1]*COATI, I3S & INRIA d'Université Côte d'Azur*

FlyHash, un algorithme de Locality Sensitive Hashing inspiré par le système nerveux des drosophiles, s'est révélé particulièrement efficace pour la recherche de similarité et ce en particulier dans le contexte fédéré, où plusieurs clients collaborent pour résoudre une tâche d'apprentissage statistique. FlyHash repose en grande partie sur l'utilisation d'un procédé de binarisation appelé winner-take-all. La façon dont cette opération winner-take-all est implémentée en pratique représente un bottleneck majeur pour l'exploitation de cet algorithme pour traiter un flux important de données. Nous proposons dans cet article un algorithme simple pour rendre cette opération winner-take-all efficace sur GPU. On implémente grâce à cela une version FlyHash déployable sur l'architecture CUDA. On évalue expérimentalement la rapidité de cette version et on présente la comparaison avec la version CPU de FlyHash.

Mots-clefs : Fouille de données, Hashing, winner-take-all, Algorithmes Distribués, Apprentissage Fédéré, GPU.

1 Introduction

Locality-sensitive hashing (LSH) is a technique in computer science introduced in [IM98] for hashing data so that similar data has a high probability of having similar hashes, while different data is likely to have different hashes. It is widely used, especially for similarity search in large databases using faster heuristics than traditional approaches, such as nearest neighbor searching. Locality-sensitive hashing is particularly effective in real-time applications, where the speed of similarity search is essential to handle a massive and continuous flow of incoming data.

In nature, animals are constantly faced with similarity recognition tasks. This is notably the case for fruit flies, which, when encountering new odors, seek to identify similarities with odors they have previously encountered in order to assess the potential quality of the available food. Observation of their olfactory nervous system revealed that some of these neural circuits bore striking similarities to well-known LSH algorithms. Based on these observations, researchers propose a new type of algorithm called FlyHash [SDN17].

FlyHash is based on the use of random projections followed by a binarization process, as is notably the case of one of the most well-known LSH heuristics SimHash [Cha02]. However, unlike SimHash, the binarization used by FlyHash is not based on thresholding, but on a process called winner-take-all (WTA), which we describe in detail below.

The adoption of winner-take-all is motivated on the one hand by its practical and computational advantages [YSRL11], and on the other hand for modelling the brain, in particular in the model of *Assemblies of Neurons* proposed by [PVM$^+$20] as well as in *Spiking Neural Networks* [Che17], artificial neural networks that are biologically closer to the real ones.

Besides, the classical winner-take-all implementation is known to represent a major bottleneck when one wants to process multiple data at once, or using large hashlengths (indeed the accuracy of such hashing algorithms improves significantly with the increase of the size of the generated hashes). In addition, the majority of data mining applications are now massively parallelized, via the use of distributed algorithms and the dominant hardware architecture is now the Graphics Processing Unit (GPU).

Our contribution with this work is to demonstrate that such WTA hashing schemes are compatible with the GPU architecture, allowing to implement them on a pipeline fully executable on GPU. Some works

have focused on GPU implementation of winner-take-all including [MVSG+09], but this direction remains relatively unexplored.

The main reason for focusing on FlyHash in this paper is that this hashing scheme is the basis of the FlyNN algorithm, introduced in [SR21], who exploited the work of [DSSN18], taking advantage once again of biological observations, to design a classification algorithm. FlyNN has been deployed in the context of federated learning in [RS22] and is currently the state of the art approximation of the k-nearest neighbor classification algorithm in the federated setting. FlyNN has in particular the advantage of being usable in the context of one-shot federated learning, where the communication among clients is restricted to a single round.

2 Description of the algorithm

2.1 FlyHash

The FlyHash algorithm takes as input a vector in $\mathbb{R}^d$ and returns its hash, which is a binary vector of length N (the *hashlength* parameter). The algorithm has two main parts, a projection and a winner-take-all binarization part.

The projection matrix is a random binary matrix M of size $N \times d$ with a fixed number s (the *projection* parameter) of zeros in each row. The first part of the algorithm is the multiplication between M and the input vector.

The winner-take-all binarization is then applied to the outcome of the previous step, which is in $\mathbb{R}^N$, and transformed into a $\{0,1\}^N$ vector by setting the k highest entries to one and the others to zero. The parameter k is also called *number of winners* parameter.

Algorithm 1 contains the FlyHash pseudocode. The WTA function is explained in detail in the next section.

Algorithm 1 FlyHash

Input: $X \in \mathbb{R}^{d \times b}$, $M \in \{S \in \{0,1\}^{N \times d} :$ each row of S contains s ones$\}$, $k \in [1, N]$
Output: $X \in \{0,1\}^{N \times b}$
  $A = M \times X$
  return $\mathrm{WTA}(A, k)$

2.2 A parallelized winner-take-all

We implement the FlyHash algorithm on the GPU to process large amounts of data. Our main contribution is a parallelized winner-take-all binarization algorithm that, rather than taking as input a single vector as mentioned before, processes a batch of vectors in a matrix X of size $N \times b$, where b is the batch size. The binarization step is thus applied to each column simultaneously. More specifically, we perform a parallel binary search for the values that, when used to threshold the respective columns, give the desired number of ones k.

Algorithm 2 contains the winner-take-all pseudocode. It starts by computing, for each column, the lower bound lb and the upper bound ub of the search interval by taking, respectively, the minimum and the maximum with a small margin $\varepsilon > 0$ to allow for strict inequalities. It then calculates the middle value mid and updates the extremes according to the current number of ones tot (corresponding to the number of values greater than mid) : if they are greater than k, we increase the lower bound of the interval by setting it equal to the middle value ; instead, if they are less than k, we decrease the upper bound to be equal to the middle value. The process is repeated a given number of times, which is at most 278 for single-precision floats, but in practice it can be set to 64 if a small fraction of erroneous entries can be tolerated (for example, the average fraction of erroneous entries caused by such a limitation is around 2.095×10^{-7} when the output is of size 20000×5000).

Algorithm 2 Winner-take-all (WTA). Functions preceded or followed by a dot (Julia's broadcasting operator) are applied element-wise.

Input: $X \in \mathbb{R}^{N \times b}$, $k \in [1, N]$
Output: $X \in \{0,1\}^{N \times b}$
  $lb = \texttt{minimum}(X,\ \text{dims} = 1). - \varepsilon$
  $ub = \texttt{maximum}(X,\ \text{dims} = 1). + \varepsilon$
  $mid = (lb. + ub)./2$
  for _ in 1 : 64 **do**
    $tot = \texttt{count}(X. > mid,\ \text{dims} = 1)$
    $lb = \textbf{ifelse}.(tot. > k, mid, lb)$
    $ub = \textbf{ifelse}.(tot. < k, mid, ub)$
    $mid = (lb. + ub)./2$
  end for
  return $X. > mid$

3 Experiments

In our experiments, we compare the performance of the FlyHash algorithm on an `Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz` CPU and a `NVIDIA Quadro RTX 8000` GPU with `CUDA version: 11.8`, focusing on the processing of large amounts of data. We implemented the algorithms in the Julia programming language, which is now one of the most popular languages for scientific computing, and we relied on CUDA.jl package for the GPU part. On the algorithm engineering side, some optimizations have been made to speed up the process, such as pre-allocating variables for the GPU version. As for the CPU implementation, it uses efficient partial sorting algorithms to select the k winners [RS22]. Our code is available in the following Github repository : `https://github.com/AInnervate/flyhash.jl`, along with the code to replicate the experiments explained in more detail below.

First, we test the speed of the two versions of the algorithm on well-known datasets taken from the Machine Learning Datasets Julia library MLDatasets.jl. One of the two datasets we examine is the FashionMNIST dataset, which is a collection of 60000 greyscale images with a size of 28×28. We therefore transform each image into a vector of dimension $d = 28 \times 28 = 784$ and collect them in a matrix of dimension 784×60000. Then we pass it as input to the FlyHash algorithm, choosing the following parameters according to previous work [SDN17] : the hash length N is equal to the hash factor $h = 32$ multiplied by the input dimension d, the projection parameter s is set to 5% of the input dimension d and the number of winners k is set to 5% of the hash length N. The same configuration is used to compute the time to process the CIFAR10 dataset, which is a collection of 50000 coloured images with a size of 32×32, so the input dimension of the matrix is $d \times b$, where $d = 32 \times 32 \times 3 = 3072$ (the factor 3 comes from the fact that the images are coloured) and the batchsize is $b = 50000$.

In addition to those datasets, we perform experiments on synthetic data, where each entry is uniformly sampled in [0,1]. To understand how the two architectures behave when dealing with large amounts of data, we run the code varying the batchsize b in the range $\{2^1, ..., 2^{15}\}$ and fixing the other parameters : the projection parameter and number of winners are as above, the hash factor is set to $h = 128$ and the input dimension is $d = 1024$.

The results are shown in Figure 1 and Figure 2. They are obtained by averaging 10 independent runs after first running the code without taking the time to avoid Julia's just-in-time compilation overhead. Both plots show low standard deviation values, in the first case as bars and in the second as shadows.

4 Conclusions

In this work, we proposed a GPU implementation of the famous FlyHash locality sensitive algorithm. Experiments show that our GPU version can run one order of magnitude faster than the best CPU version, thus allowing to speed up the use of the FlyHash algorithm in settings where GPUs are more easily available than dozens of CPU cores. In future work, we expect that the GPU code can be further improved in terms

of performance by writing a lower-level CUDA kernel.

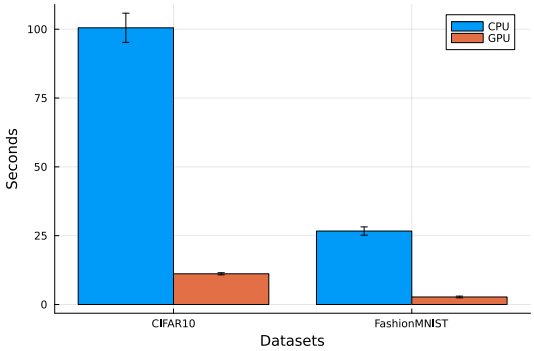

FIGURE 1: Comparison on two popular datasets. Hash factor parameter is set to $h = 32$.

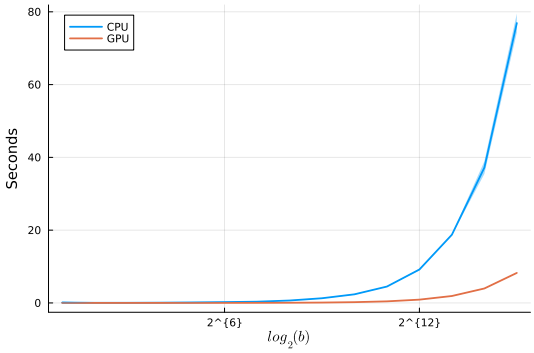

FIGURE 2: Comparison as batchsize increases with fixed hash factor $h = 128$ and input dimension $d = 1024$.

Références

[Cha02] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*. ACM, 2002.

[Che17] Yanqing Chen. Mechanisms of winner-take-all and group selection in neuronal spiking networks. *Frontiers in computational neuroscience*, 2017.

[DSSN18] Sanjoy Dasgupta, Timothy C. Sheehan, Charles F. Stevens, and Saket Navlakha. A neural data structure for novelty detection. *Proceedings of the National Academy of Sciences*, 2018.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors : Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98. Association for Computing Machinery, 1998.

[MVSG+09] O. Moslah, A. Valles-Such, V. Guitteny, S. Couvet, and S. Philipp-Foliguet. Accelerated multi-view stereo using parallel processing capababilities of the gpus. In *2009 3DTV Conference*, 2009.

[PVM+20] Christos Papadimitriou, Santosh Vempala, Daniel Mitropolsky, Michael Collins, and Wolfgang Maass. Brain computation by assemblies of neurons. *Proceedings of the National Academy of Sciences of the United States of America*, 2020.

[RS22] Parikshit Ram and Kaushik Sinha. Federated nearest neighbor classification with a colony of fruit-flies. In *Thirty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press, 2022.

[SDN17] Charles F. Stevens Sanjoy Dasgupta and Saket Navlakha. A neural algorithm for a fundamental computing problem. *Science*, 2017.

[SR21] Kaushik Sinha and Parikshit Ram. Fruit-fly inspired neighborhood encoding for classification. In *KDD '21 : The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2021.

[YSRL11] Jay Yagnik, Dennis Strelow, David A. Ross, and Ruei-sung Lin. The power of comparative reasoning. In *2011 International Conference on Computer Vision*, 2011.