National Technical University of Athens
School of electrical and computer engineering
Master program in Data Science and Machine Learning

ParML semester exercise - A techno-economic analysis

Team ParML03

Alexandros Vythoulkas
alvythoulkas@protonmail.com
03400045

Dimitrios Zerkelidis
dimzerkes@gmail.com
03400049

Maria Kaiktzoglou
makaiktzoglou@gmail.com
03400052

Thanos Pantos
pantos.thn@gmail.com
03400026

Sotiris Pelekhs
pelekhs@gmail.com
03400069

Spyros Pouros
spyros.poulos@gmail.com
03400071

2020
July-September

Contents

1. Exercise 1

The first exercise deals with the computation of the total system speedup for two programs based on Amdahl's law. We used the results obtained from the two lab exercises in which we experimented with parallelization, tuning some parameters in various ways. The two programs regard a classification task with (a) a Naive Bayes and (b) a logistic regression classifier. The total speedup was measured exclusively on the training part for logistic regression, as it is a part that can be parallelized. We accepted the assumption that the portion that can be parallelized is 80%. Although, on Naive Bayes acceleration, the speedup was not on training but in inference stage, because the classification of non-labeled samples using MLE can be very data-intensive. That said, we assume that on Naive Bayes the portion that can be parallelized is also 80%.

We are calculating the total speedup based on the following formula:

$$S_{latency} = \frac{1}{1 - p + \frac{p}{s}}$$

where p is the portion of the program that can be parallelized and s the respective speedup achieved in the parallelized portion

Regarding the Naive Bayes, the maximum speedup that was obtained was 62.76. Thus, the respective speedup was $S_{latency} = 4.7$. Logistic regression's maximum speedup was 4.39 and consequently $S_{latency} = 2.616$

2. Exercise 2

The second exercise concerns a Techno-economic analysis on different accelerators such as Xeon, FPGAs, GPUs and TPUs. This analysis is doubtlessly valuable as it guides the choice of the optimal accelerator with respect to the task to be performed. Our research is based on articles and product briefs.

First, we compare the speed ups from the lab Jupyter notebooks with those speed ups we found from our research. Then we present a techno-economic evaluation for each platform to identify the fastest and the most cost-efficient ones. For this purpose we used the tables provided by Amazon. We were asked to consider the reference design the processor that we used in Jupyter notebooks, which was "Xeon-S 4210 10-Core (2.20GHz 14MB) 32GB" and consider the reference speedup to be 1. Finally, after conducting some market research and computations, we are suggesting a platform for best performance and a platform for the best performance/cost, supposing we have to do do Logistic regression and Naïve Bayes and use the Xeon processor which takes 100 hours to finish.

2.1 Speedup comparison on different accelerators

In this section we are going to provide the speedup results from our research about logistic regression and Naive Bayes executing in different accelerators. On this article [1], we found a comparison among three different platforms, GPU, CPU and FPGA. The performance evaluation is in terms of total execution time, accuracy and cost and the algorithms used for classification are logistic regression, Naive Bayes and KMeans. The implementations used included Python's Scikit-learn package (alongside Intel's Math Kernel Library), Rapids cuml library and InAccel Sklearn-like package. As the article states, Intel's MKL is a CPU math processing accelerated framework, while the others are newer solutions built on top of GPU and FPGA accelerators respectively. The comparison took place among 4 different platforms:

- Reference: Intel Xeon Skylake SP (r5.2xlarge with original package)
- MKL: Intel Xeon Skylake SP (r5.2x large using MKL)
- GPU: NVIDIA® V100 Tensor Core (p3.2x large RAPIDS library)
- FPGA: FPGA (f1.2x using InAccel ML suite)

2.1.1 Logistic Regression

The article we were based on [2] provides notebooks and results, which are obtained from the sklearn implementation, just like the lab's implementation. The dataset used is also the same, namely, the MNIST dataset with 10 classes. The article has used (i) letters dataset: 26 classes, 125,000 train samples and 20,000 test samples, and (ii) digits dataset: 10 classes, 1 and 2 million train samples and 200,000 test samples. We have used the digits dataset with 70,000 samples and 3fold cv. The article 's implementation regards the use of the original code and the Intel MKL fo optimization of ML kernels for the case of CPUs. As far as GPUs are concerned the RAPIDS framework is used and regarding FPGAs, the authors have used their own ML suite for logistic regression which is available from InAccel.

The optimal speedup achieved on the Jupyter notebooks was 4.39. The article's authors have repeatedly run their notebooks and extracted the mean running times, using them to compute the speedup, which was 4 for the FPGA. We have copied part of the results from the following article [2].

Our results were the following:

Thus, we can notice that in the digits dataset our speedup result basically coincides with that of the article although the size of the dataset varies.

	Reference	MKL	Rapids	FPGA(InAccel)	FPGA SpeedUp
1m digits dataset time	152s	81s	30s	38s	4x
2m digits dataset time	317	165s	62s	72s	4.4x

Table 2.1: Logistic Regression Comparison - Digits Dataset

	Reference	MKL	Rapids	FPGA(InAccel)	FPGA SpeedUp
letters dataset time	45s	17s	4.5s	7s	6.43x

Table 2.2: Logistic Regression Comparison - Letters Dataset

	CPU(Reference)	FPGA(InAccel)	FPGA SpeedUp
digits dataset time	191.62s	43.65s	4.39x

Table 2.3: Logistic Regression - Lab Jupyter Notebooks

The following figures 2.1 2.2, taken from [1] depicts the total execution times of each platform as well as the accuracy achienved. We can see that although GPUs have the lower running time, they achieve inferior accuracy. Thus, it is evident that FPGAs outperform when it comes to the combination of performance and running time.

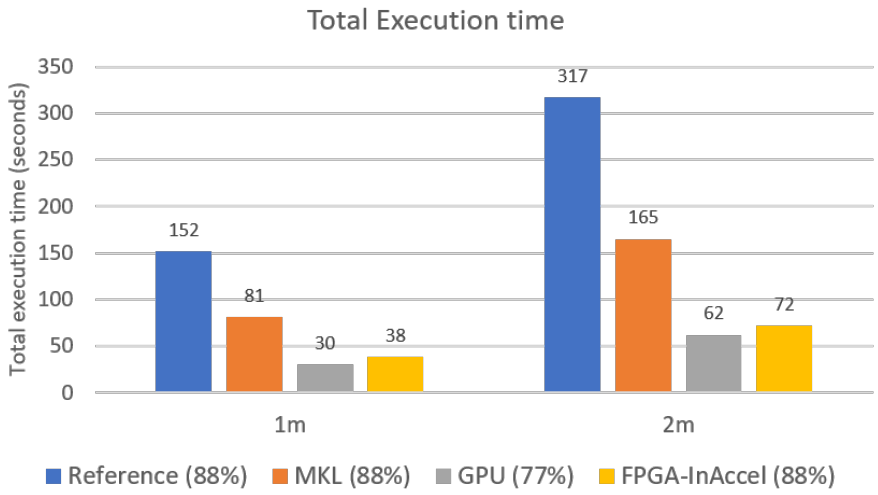

Figure 2.1: Accelerators comparison on Time and Accuracy

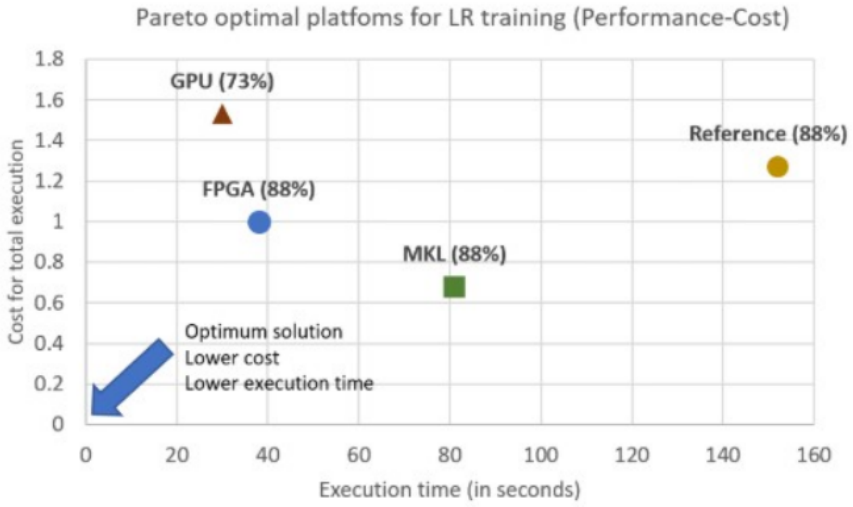

Figure 2.2: Performance-Cost plot

A different research based on H20.ai[3], they refer to a new generation of GPU algorithms for Machine Learning problems, called H204GPU, that achieve with the use of 8 GPUs NVIDIA Tesla p100 speedup equal to × 35.7 over 2 Xeon CPUs. In figure 2.3, we can see the execusion times as we described before.

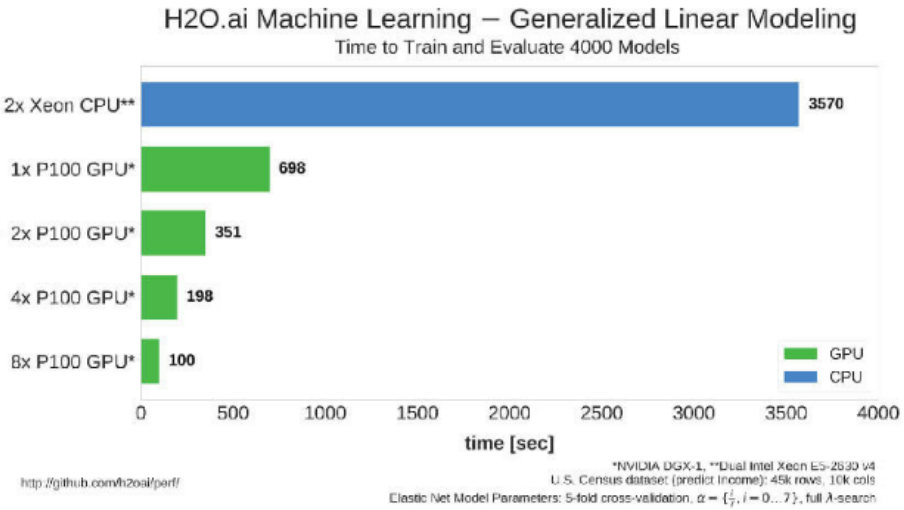

Figure 2.3: Plot of execution time based on H20.ai

According to GPU and CPU comparison, we would like to refer to a new framework of IBM called Snap Machine Learning [4], which achieves × 10 speedup in logistic regression with a GPU NVIDIA Tesla V100 over a CPU Intel x86-based machine (Xeon Gold 6150 CPU @ 2.70 GHz). A speedup of × 46 can be achieved in a huge dataset if 16 GPUs NVIDIA Tesla V100 and 4 Powere 9 CPUs are used [5]. This speedup has as baseline the second best performance of Tensorflow(×3.2 over 1 core scikit-learn implementation). These results are visualized in the next figures 2.4 2.5 on the facing page.

Figure 2.4: Time and Log loss of Snap ML

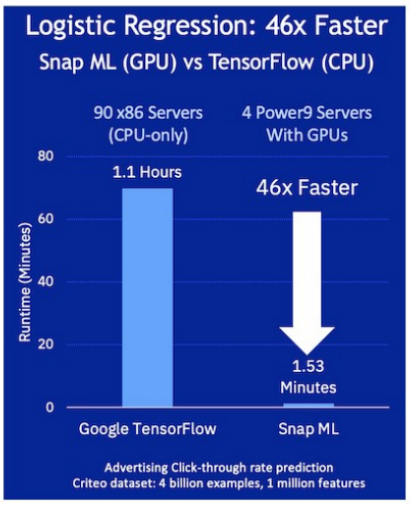

Figure 2.5: Max speedup of Snap ML

Another paper [6] provides further evidence about the potentiality of FPGAs in terms of speedup improvement. This paper presents the framework Spynq, a framework for the efficient development of data analytics on embedded systems that are based on the heterogeneous MPSoC (Multi Processors Systems on Chip) FPGA called Pynq, which is mapped on Spark. Pynq is based on the Zynq all-programmable SoC. As we read in the paper, Zynq FPGA incorporates two RISC Crotex A9 ARM cores and a programmable logic unit in a single chip. Each of these cores has 32 KB level 1 4-way set-associative instruction and data cache and they share a 512 KB Level 2cavhe. The processors are clocked at 667 Mhz and they have coherent multiprocessor support. This framework is evaluated in a machine learning application that employes logistic regression on a handwritten digits recognition problem. The dataset consists of 40,000 training samples with 784 features and 10 labels. The results show that heterogeneous FPGA based MPSoc can achieve up to 11x speedup compared to the execution time in the ARM cores. Evaluation took place performance in terms of latency and execution time. Namely, it is shown that 11x acceleration could be achieved when the framework was used with logistic regression.

In addition to the FPGA implementations for logistic regression we have found the followings in the research literature. According to Elias Koromilas, Ioannis Stamelos, Christoforos Kachris, and Dimitrios Soudris [7] it is possible to get a speedup over $\times$ 11 using Pyng framework that is consisted from Zynq FPGA with 2 RISC Cortex A9 ARM cores. Inaccel has very close results to what we mentioned before and can get until $\times$ 12 speedup over logistic regression with 8-core CPU, $\times$ 16 speedup with 16-core CPU and $\times$ 70 for single thread execution [8]. Inaccel offers accelerators for SparkML in Amazon AWS with the use of f1 instance, that offers a very good ratio of performance and cost.

The research literature over TPUs is very poor on machine learning algorithms. And that is discussed briefly later in the Conclusions Chapter, since they are prefered on Deep Learning models.

Finally, about ASICs combined with FPGAs we have found the following research [9]. There, the authors use a full computing stack constituting language, compiler, system software, template architecture and circuit generators, called CoSMIC (consisted of Xilinx UltraScale VU9P and 2 P-Asics). Using as a baseline of 4 NODE-CPU over spark, 4-FPGA, 8-FPGA, 16-FPGA CoSMIC configurations apply respectively $\times$ 12.6 ,$\times$ 23.1, $\times$ 33.8 speedups.

2.1.2 Naive Bayes

Based on the same article [2], we found the execution times on Reference, MKL and FPGA for Naive Bayes in the respective section. The acceleration is calculated on the prediction and not on the training stage, as the classification of non-labeled samples using maximum likelihood can be significantly data-intensive. This can be useful there are time constraints to train a model on small real-world batches and classify large batches of data. RAPIDS is not included on the third benchmark as there does not exist a Naive Bayes classifier for it cuML. We tested the execution time for predictions on various datasets. The first dataset is the handwritten letters dataset that consists of 26 clusters, 20,000 training samples and 125,000 test samples. The second dataset is handwritten-digit dataset with 10 clusters, 200,000 training samples and 1 and 2 million test samples.

	Reference	MKL	FPGA(inaccel)	SpeedUp
Time on 1st_Dataset - 125k	17s	17s	0.51s	33.4x
Time on 2nd_Dataset - 1m	51.9s	51.9s	3.95 s	13.1x
Time on 2nd_Dataset - 2m	106.7s	106.7s	7.71s	13.8x

Table 2.4: Naive Bayes Comparison

The table 2.4 reveals when applying Gaussian Naive Bayes with FPGA(inaccel) accelerator the best SpeedUp of the three benchmarks is achieved, reaching even 33x compared to the software-only execution. Also, we should mention that the accuracy in all cases is the same as the Reference/MKL one. Consequently, we can say that FPGAs improved the performance over the execution time on test sampling prediction without reducing the accuracy.

As we mentioned before, Naive Bayes [2] could not be implemented using GPUs. To do a comparison with GPUs we searched other research papers with different parameters and datasets. That said, a line is drawn regarding the comparison of our results with the findings of our research.

This paper in proceedings [10] makes a comparison between GPU and CPU accelerators on a Naive Bayes implementation. The GPU implementation was based on CUDA framework. The authors refer to Naive Bayes as the most widely used algorithm in document classification, collocating it with the advent of WEB 2.0 and the generation of huge amount of document data lent for analyzing. Thus, accelerating Naive Bayes comes in hand with improving this computational challenge.

The dataset that the researchers used consists of six real digital libraries that contained medical documents and a set of news from Reuters. It also includes a collection of documents from newsgroups, and a collection related to webpages collected by the World Wide Knowledge Base Project of the CMU text learning group. For the CPU implementation the researchers **inproceedings** used C++ with shared memory parallelization using OpenMP so that they can get a fair comparison over the GPU-CUDA implementation.

Collection	Classes	Number of Attributes	Number of Documents	Density
MedLine	7	268,783	861,454	30.805
ACM	11	56,449	24,897	27.687
20ng	20	61,050	18,805	129.511
Reuters_ny	8	24,985	8,184	42.2302
Webkb	7	40,195	8,277	139.275
acl_bin	2	1,110,351	27,677	181.509

Figure 2.6: general information on the data collections

All experiments were performed on a computer with Intel Core 17-2600 CPU at 3.40 GHz with HyperThreading, 16GB of main memory and equipped with a GeForce GT520 2GB graphics card. All results presented are averages of 10 independent executions of the algorithms. In the case of CPU implementation the average execution time of 1(sequential), 2 and 4 threads was measured, as this is the number of cores in the processor. Also, they used hyperthreading to try 8 thread version and observe the results. The graph with the speedup comparison can be seen below.

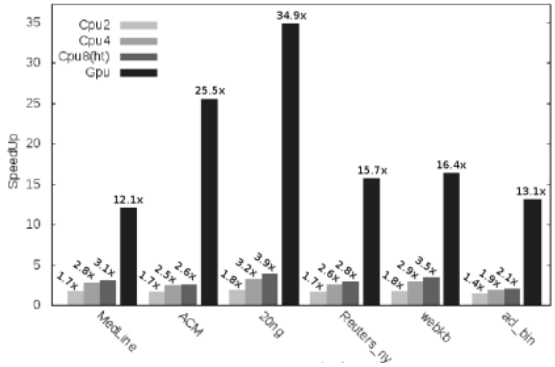

Figure 2.7: Speedup of Parallel Implementations for Various Collections

Dataset	ARM Cortex-A9		NBC Accelerator	
	CC	RT (ns)	CC	RT (ns)
MNIST	2.7+e10	6.8+e10	8.0+e6	8.0+e7
Car	3.3+e7	8.3+e7	3.7+e3	3.7+e4
Connect	2.0+e11	5.0+e11	5.4+e5	5.4+e6
Voting	1.2+e7	3.0+e7	1.8+e3	1.8+e4
Monk	4.5+e6	1.1+e7	1.1+e3	1.1+e4

Figure 2.8: Training cost time

Dataset	ARM Cortex-A9		NBC Accelerator	
	CC	RT (ns)	CC	RT (ns)
MNIST	3.9+e7	9.8+e7	800	8.0+e3
Car	6.2+e4	1.6+e5	13	1.3+e2
Connect	1.7+e7	4.2+e7	48	4.8+e2
Voting	1.5+e5	3.8+e5	21	2.1+e2
Monk	3.9+e4	9.8+e4	13	1.3+e2

Figure 2.9: Inference cost time

Results show that for parallelizations in CPU, the speedup obtained by utilizing 4 threads (effectively 4 cores) reaches a maximum of 3.2x (for the 20ng collection). At the same time, evaluation on GPU-NB reveals speedups of up to 34.9x (also for the 20ng collection), meaning that the proposed algorithm on GPU accelerator leads to significant reduction on the execution time. Based on these two results, one can claim that NaiveBayes-CUDA is 11x faster on the GP(GeForce GT520 2GB).

Another research [11] implemented Naive Bayes Classifier Real-Time on FPGA accelerator, using logarithm transformation based look-up table and float-to-pixed point process to simplify the calculations of the method. Accuracies and performance/cost were compared over an ARM Cortex-A9 processor. The proposed hardware accelerator was implemented on a Xilinx ZYNQ 7020, using Verilog HDL and Vivado 2018.1. The experiments used MNIST dataset and some UCI datasets, namely, Car Evaluation Dataset (Car), Connect-4 opening dataset (Connect), 1984 United States Congressional Voting Records Dataset (Voting) and The Monk's Problems DataSet (Monk).

Figures 2.8 and 2.9 depict the time cost of the training and inference part over all the datasets for each accelerator used. The time cost of the training and the inference part of the NBC accelerator are both far less than their counterparts of the ARM Cortex-A9 processor. As to clock cycles (CC), the design improves the efficiency of the training part up to 3.7+e5 times and 7.9+e4 times on average. Moreover, the NBC accelerator improves the performance of the inference part 8.3+e4 times on average, up to 3.5+e5 times. Similarly, the running time (RT) of the training and the inference part are reduced 2.0+e4 times and 2.1+e4 times on average, respectively. It is also important to mention that this acceleration was achieved without accuracy being significantly reduced as we can see on figure 2.10.

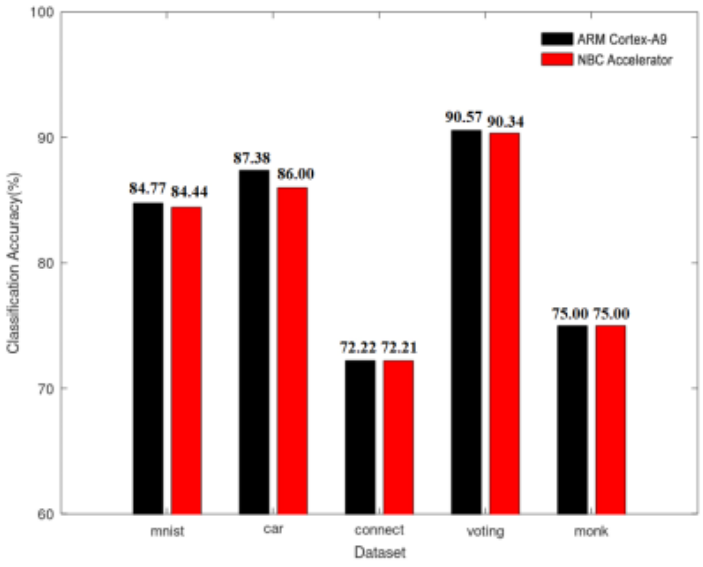

Figure 2.10: Accuracies of the two accelerators over the datasets

Parallel heap array was built to accelerate the process of feature quantization. Distance table lookup is built to speed up the process of feature search. Comparative experiments on UIUC-Sport dataset demonstrate that paper's integrated solution outperforms other implementations significantly on Core-quad Intel Core i7 950 CPU and GPU of NVIDIA Geforce GTX460.

Scalability experiment on 80 million tiny images database shows that their approach still performs well when large-scale

Implementation	Description
CPU1T_proposed	Product quantization + 1-thread CPU sequential implementation
CPU8T_proposed	Product quantization + 8-thread multi-core parallelization
GPU_brute	GPU based *brute force* feature search
GPU_proposed	Product quantization + GPU parallelization

Figure 2.11: Different implementations of NBNN used for comparison

image database is explored. The NBNN-based image classification algorithm was implemented on GPU using CUDA framework.

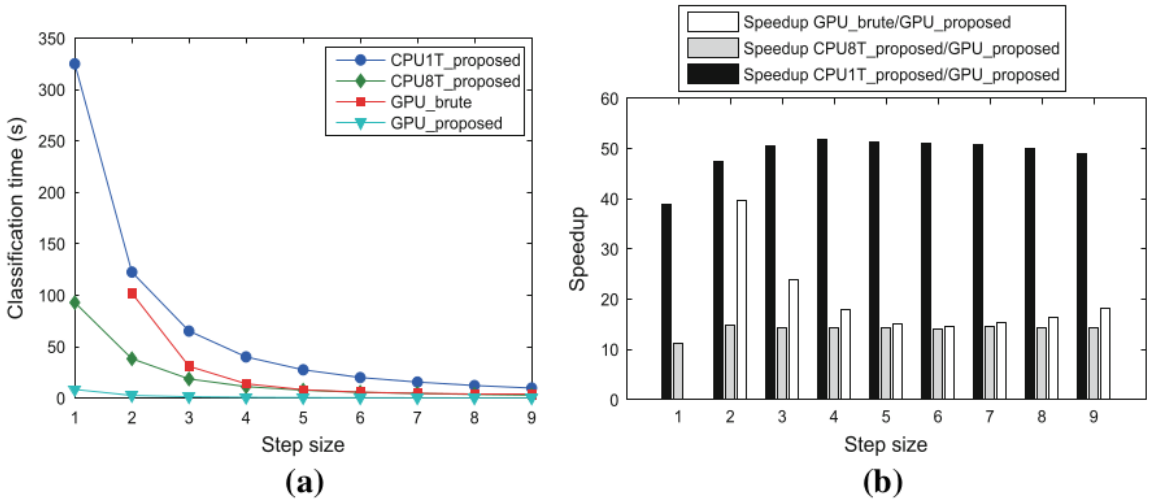

Figure 2.12: Performance of different implementations. a) Describes the classification time variation with feature extraction step size. b) Describes the speedup variation with feature extraction step size

Figure 2.12 illustrates the actual classification time for different methods to answer the question-how fast different systems can response classification request via an empeirical experiment. Figure 2.12 (a) reports the classification time variations of different implementations while (b) reports the corresponding speedups. Among all NBNN implementations, paper's approach using GPU to accelerate the NB algorithm, consumes less classification time than any other approaches. In addition, it is worth noting that compared with GPU_brute,CPU8T_proposed consumes less time.

The approach proposed by the paper seems from figure 2.12 that it outperforms other methods significantly. For example, When the step size is 2, their approach can get $\times$ 47.5 speedup compared with 1-thread CPU implementation, 14.9x speedup compared with multi-core implementation, and $\times$39.5 speedup compared with GPU_brute implementation.

2.1.3 Conclusions

Currently, data scientists have a plethora of choices when it comes to the processing platform to use for training machine learning models. Cloud providers offer platforms such as general purpose CPUs, compute-optimized CPUs, memory-optimized CPUs, GPUs, TPUs and FPGAs, are AWS, Alibaba cloud, Azure and Huawei. Choosing the best platform is a difficult task and should take into account many parameters. An article from InAccel [12] addresses this problem and tries to provide some insights on when to choose a specific accelerator. We attempt to connect the ideas of this article over the results we found on the previous scientific research we read.

First of all, it is important to highlight the advantages of each platform. When it comes to CPUs, their main advantage is that they are very easy to program and they can support any programming framework. However, when it comes to

machine learning training a CPU is suited for simple models that do not take long to train, and for small models with small and effective batch sizes. If one wants to run large models over large datasets, then the total execution time for machine learning training would prevent the use of CPU.

GPUs are specialized processing units that were mainly designed to handle large amounts of data in parallel. They have many more processing units than CPUs that are simpler in structure. This trait makes them ideal for applications in which data needs to be processed in parallel, something that explains why the research results we found on literature found GPU -which is used for parallelization- to be faster than CPU.

The use of TPUs (Tensorflow Processing Units) basically regards deep learning models. TPUs are significantly fast at performing dense vector and matrix computations and they are specialized on running programs written with Tensorflow. This is the reason why we did not attempt to search implementations of TPUs, as we are dealing with logistic regression and Naive Bayes algorithms.

As far as FPGAs are concerned, this inAccel article [12] suggests that a great way to use FPGAs to train a model be through the use of pre-configured architectures specialized for the applications of interest. This can provide importantly higher performance compared to CPUs and GPUs and at the same time there is no need to edit the code. The pre-configured accelerated architectures provide all the required APIs and libraries for programming frameworks (Python, Scala, Java, R and Apache Spark) and allow to overload the most computationally intensive tasks, as well as offload them in the FPGAs. This method suggested by InAccel was show to lead to up 3x faster training with Logistic Regression and inference with Naive Bayes.

Accelerators	Speedups(x)	Cost/hour(in €)	Total Cost(in €)
Snap ML (1 GPU NVIDIA Tesla V100)	10	4.18	41.86
Snap ML (16 GPU NVIDIA Tesla V100)	46	43.41	130.217
InAccel (MKL Intel Xeon Skylake SP r5.2xlarge)	1.92	0.42	22.67
InAccel (NVIDIA Tesla V100 TCore p3.2xlarge)	5.11	2.59	51.96
InAccel (FPGA ML suite f1.2xlarge)	4.4	1.40	100.87
InAccel (FPGA ML suite f1.16xlarge)	16	11.59	78.45
H2O.ai H2O4GPU (8 GPU NVIDIA P100)	35.7	9.91	29.75
CoSMIC 4 FPGAs + 8 P-ASICs	12.6	4.91	39.33
CoSMIC 8 FPGAs + 16 P-ASICs	23.1	9.83	49.16
CoSMIC 16 FPGAs + 32 P-ASICs	33.8	19.7	59.35

Table 2.5: Performance, Cost per Hour and Total cost for logistic regression

2.2 Techno-Economic Analysis

For the Techno-Economic analysis we have made a research over the pricings of Amazon's EC2[13], Google Cloud GPU[14] and SNAP ML IBM[15]. We have created tables that denote the Performance (or speedup), Cost/Hour(in euro) and total cost(in euro) of each algorithm. We use as baseline the Processor used in our jupyter notebooks "Xeon-S 4210 10-Core (2.20GHz 14MB) 32GB".

2.2.1 Logistic Regression

On table 2.5 we can see the techno-economic analysis for the logistic regression algorithm. As we mentioned before we used as references [13], [15] and [14].

In terms of total cost the best choice would be Inaccel (MKL Intel Xeon Skylake SP r5.2xlarge). But in case of performance the most optimal choice would be SnapML of IBM with 4 GPU NVIDIA Tesla v100 which speedups our program $\times$ 46. So instead of being executed in 100 hours, it will need 2.17.

Although to find the balance between performance and cost we should use a pareto diagram. This diagram can be seen in the next figure 2.13.

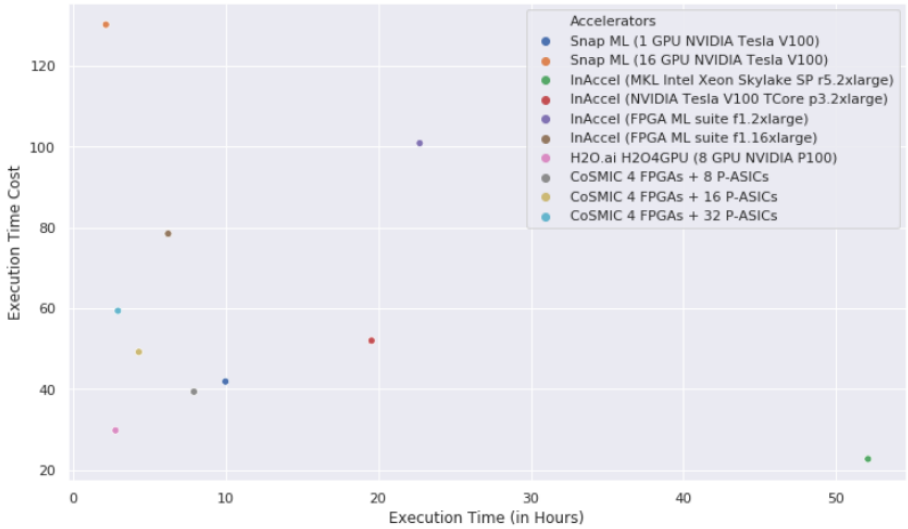

Figure 2.13: Performance-Cost Pareto Diagram for Logistic Regression algorithm

The best choice when we want to find the balance between Performance and Cost is the point that is closest to the origin. After observing the diagram 2.13 we conclude to H204GPU(8 GPU NVIDIA P100) as the most optimal choice for logistic regression.

Finally, we should make our final choice depending on what is our goal(performance,cost or both).

Accelerators	Speedups(x)	Cost/hour(in €)	Total Cost(in €)
GPU-NB	34.9	0.76	2.29
PQ on GPU	52	0.76	1.52
NBC Accelerator	850	1.40	1.40
InAccel (FPGA ML suite f1.2xlarge)	90.9	1.51	3.023

Table 2.6: Performance, Cost per Hour and Total cost for Naive Bayes

2.2.2 Naive Bayes

On table 2.6 we are able to see the techno-economic analysis for Naive Bayes algorithm. Again, to create the table we used the following sources [13], [15] and [14]. By observing the table we can see that the best choice for both performance and cost separately and combined is the NBC accelerator.

In the case of Naive Bayes we do not need to plot a pareto diagram, since we already know our optimal choice even if we want to balance the trade-off between performance and cost.

Bibliography

[1] Inaccel, *CPU, GPU or FPGA: A use case on Logistic regression training in cloud computing platforms*, 2019. [Online]. Available: https://inaccel.com/cpu-gpu-or-fpga-performance-evaluation-of-cloud-computing-platforms-for-machine-learning-training/.

[2] ——, *Performance evaluation of Machine Learning Libraries on CPU, GPU and FPGA*, 2020. [Online]. Available: https://bitbucket.org/inaccel/notebooks/src/master/.

[3] H2O.ai, *H2O.ai Releases H2O4GPU, the Fastest Collection of GPU Algorithms on the Market, to Expedite Machine Learning in Python*, 2019. [Online]. Available: https://www.h2o.ai/blog/h2o-ai-releases-h2o4gpu-the-fastest-collection-of-gpu-algorithms-on-the-market-to-expedite-machine-learning-in-python/.

[4] C. Dünner, T. Parnell, D. Sarigiannis, N. Ioannou, and H. Pozidis, "Snap Machine Learning", Mar. 2018.

[5] S. Gupta, *IBM Research Cracks Code on Accelerating Key Machine Learning Algorithms*, 2018. [Online]. Available: https://medium.com/@sumitg%2016893/ibm-research-cracks-code-on-%20accelerating-key-machine-learning-algorithms-647b5031b420.

[6] E. Koromilas, I. Stamelos, C. Kachris, and D. Soudris, "Spark acceleration on FPGAs: A use case on machine learning in Pynq", May 2017, pp. 1–4. DOI: 10.1109/MOCAST.2017.7937637.

[7] Koromilas, *Spark ac- celeration on FPGAs: A use case on machine learning i Pynq, Elias Koromilas, Ioannis Stamelos, Christoforos Kachris, and Dimitrios Soudris*, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7937637.

[8] Inaccel, *FPGA Accelerated Machine Learning*, 2019. [Online]. Available: https://bigdataconference.lt/2018/%20wp-content/uploads/2018/12/Machine-Learning-Acceleration-using-FPGAs-in-the-%20Cloud-by-Christophoros-Kachris-min.pdf.

[9] J. Park, H. Sharma, D. Mahajan, J. K. Kim, P. Olds, and H. Esmaeilzadeh, "Scale-Out Acceleration for Machine Learning", in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 367–381.

[10] G. Andrade, F. Viegas, G. Ramos, J. Almeida, L. Rocha, M. Gonçalves, and R. Ferreira, "GPU-NB: A fast CUDA-based implementation of Näive Bayes", Oct. 2013, pp. 168–175, ISBN: 978-1-4799-2927-6. DOI: 10.1109/SBAC-PAD.2013.16.

[11] Z. Xue, J. Wei, and W. Guo, "A Real-time Naive Bayes Classifier Accelerator on FPGA", *IEEE Access*, vol. PP, pp. 1–1, Feb. 2020. DOI: 10.1109/ACCESS.2020.2976879.

[12] Inaccel, *CPU, GPU, FPGA or TPU: Which one to choose for my Machine learning training?*, 2018. [Online]. Available: https://medium.com/@inaccel/cpu-gpu-fpga-or-tpu-which-one-to-choose-for-my-machine-learning-training-948902f058e0.

[13] Amazon, *Amazon EC2 Pricing*, 2020. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/.

[14] Google, *Google Cloud GPUs Pricing*, 2020. [Online]. Available: https://cloud.google.com/compute/gpus-pricing.

[15] IBM, *Snap IBM Pricing*, 2020. [Online]. Available: https://cirrascale.com/pricing%20power8BM.php.