Haute École d'Ingénierie et de Gestion du Canton de Vaud

Acceleration of an AI application

Thursday, 02 February 2023

Professor: Marina Zapater

Kevin Jordil & Olivier D'Ancona

STAGE 1 – CHOOSING AN APPLICATION

1 Multi-Layer Perceptron

Multi-Layer Perceptron (MLP) is a type of artificial neural network commonly used for supervised learning tasks such as classification. It consists of one input layer, some hidden layer and one output layer. In each layer, each neuron is connected to all neurons in the next layer. The architecture that we built have 3 layers:

Architecture our neural network implementation is as follows:

- Input layer of size 784 (28x28 images)
- Hidden layer of size 1000 (number of hidden neurons)
- Output layer of size 10 (number of class of the dataset)

We used a c implementation who consists of a single file with all the functions needed to train and test the neural network.

Dataset We used the fashion dataset which is a commonly used dataset in machine learning and computer vision task. He consists of a collection of Zalando shopping items such as shirts, pants and shoes. There are in total 10 categories. he class are T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

2 Grayscale Conversion Program

Definition We ran into troubles with the cuda neural network acceleration so we choosed an auxiliary program to be accelerated with cuda. The program converts a color image to a grayscale image which is a good idea to implement with our multi layer perceptron. Because our mlp necessitates a grayscale image as input we can convert rgb images to grayscale images with this program. Furthermore this task is a good example to show the acceleration of a program with cuda.

Principle For each pixel of the image, we calculate 0.56 * red + 0.33 * green + 0.11 * blue. The result is the grayscale value of the pixel. We then set the red, green and blue values of the pixel to the grayscale value.

STAGE 2 – ANALYSING APPLICATION BOTTLENECKS

1 Execution time

MLP baseline The baseline execution time is showed on table 1 with a total time of 575.676547s.

time	accuracy	error	samples/sec	gflop/s
67.270 s	40.99%	0.045	743.28	2.20
134.516 s	61.63%	0.022	743.53	2.20
201.754 s	74.50%	0.019	743.64	2.20
268.999 s	82.60%	0.016	743.54	2.20
336.225 s	87.68%	0.015	743.77	2.20
403.459 s	90.91%	0.013	743.67	2.20
470.688 s	93.01%	0.013	743.73	2.20
537.911 s	94.43%	0.012	743.79	2.20

Table 1: MLP Baseline execution time

MLP accelerated with OpenMP The accelerated execution time is showed on table 2 with a total time of 393.993599s.

time	accuracy	error	samples/sec	gflop/s
46.092 s	40.99%	0.045	1084.79	3.21
92.241 s	61.63%	0.022	1083.43	3.20
138.231 s	74.50%	0.019	1087.20	3.22
184.198 s	82.59%	0.016	1087.76	3.22
230.153 s	87.68%	0.015	1088.01	3.22
276.135 s	90.91%	0.013	1087.40	3.22
322.124 s	93.01%	0.013	1087.23	3.22
368.109 s	94.43%	0.012	1087.32	3.22

Table 2: MLP OpenMP accelerated execution time

MLP accelerated with OpenMP and CUDA The accelerated execution time is showed on table 3 with a total time of 326.487s.

Analysis as we can see, the OpenMP implementation is the fastest then the baseline and finally the CUDA and OpenMP is the slowest. The CUDA profiler results are showed on table 4. The CUDA profiler shows that the CUDA memcpy is the most time consuming operation, followed by the backprop_kernel and the CUDA memset. So our kernel function is too short to benefit from the GPU acceleration. The cost to move the data to memory are too high.

time	accuracy	error	samples/sec	gflop/s
64.577 s	36.72%	0.061	774.27	2.29
130.425 s	57.61%	0.034	759.32	2.25
195.328 s	70.85%	0.029	770.39	2.28
261.072 s	79.38%	0.025	760.52	2.25
326.487 s	84.74%	0.024	764.35	2.26
391.392 s	88.17%	0.021	770.37	2.28
455.994 s	90.38%	0.021	773.96	2.29
519.951 s	91.87%	0.019	781.79	2.31
584.651 s	92.86%	0.020	772.79	2.29
649.924 s	93.56%	0.018	766.02	2.27
714.872 s	94.03%	0.018	769.84	2.28
780.575 s	94.48%	0.017	761.00	2.25
845.612 s	94.74%	0.016	768.80	2.27

Table 3: MLP OpenMP and CUDA accelerated execution time

2 Complexity Analysis

3 Accelerated part

4 Theoretical acceleration

STAGE 3 – ACCELERATION

STAGE 4 – ANALYSIS OF RESULTS

Type	**Time perc.**	**Time**	**Calls**	**Avg**	**Name**
GPU :	47.10%	15.4163s	410754	37.531us	CUDA memcpy HtoH
	35.72%	11.6917s	68459	170.78us	backprop_kernel
	17.18%	5.62353s	136918	41.072us	CUDA memset
API calls:	74.08%	145.662s	410754	354.62us	cudaMemcpy
	13.21%	25.9788s	136918	189.74us	cudaMemset
	9.01%	17.7185s	68459	258.82us	cudaDeviceSynchronize
	3.49%	6.86282s	68459	100.25us	cudaLaunchKernel
	0.18%	344.80ms	7	49.257ms	cudaMallocManaged
	0.04%	72.091ms	68459	1.0530us	cudaGetLastError
	0.00%	953.77us	7	136.25us	cudaFree
	0.00%	108.55us	97	1.1190us	cuDeviceGetAttribute
	0.00%	10.521us	1	10.521us	cuDeviceTotalMem
	0.00%	7.3430us	3	2.4470us	cuDeviceGetCount
	0.00%	3.8540us	2	1.9270us	cuDeviceGet
	0.00%	1.5100us	1	1.5100us	cuDeviceGetName
	0.00%	938ns	1	938ns	cuDeviceGetUuid

Table 4: MLP CUDA accelerated GPU profiling results