Comparison of execution time when running a program on the CPU vs. on the GPU

David Slatinek

March 2021

Abstract

The purpose of this paper is to demonstrate the time difference between running a python program on the CPU vs. on the GPU when calculating the result of a complex math function. We describe the algorithm implementation, along with the math function used in the program. Next, we present the results of the experiments. In the end, we define a linear regression model for determining the time taken to finish the calculation of arbitrary input size.

1 Introduction and overview

Although CPUs [1] are nowadays extremely fast, there are still some problems or algorithms that run much faster on the GPU [2] than on the CPU. Normally, these algorithms are calculating results for complex math or physics formulas. GPUs are also regularly used in the gaming industry, for crypto mining, rendering 3D models or for machine learning [1].

2 Algorithm implementation and development

As per [2], the Gaussian function is defined as:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{1}$$

where:
σ: standard deviation,
π: the number *pi*,
e: Euler's number,
μ: mean.

Equation 1 is used in the normal distribution, which is a probability function and denotes how are values distributed [3, 4].
Algorithm implementation:

1. Randomly generate function 1 input parameters, i.e. *x*, *mean* and *standard deviation*.
2. Measure elapsed time to calculate the function result when run on the CPU.
3. Measure elapsed time to calculate the function result when run on the GPU.
4. Write results to a file.

[1] Central Processing Unit.
[2] Graphics Processing Unit.

The time complexity of equation 1 is $O(1)$, therefore the time taken to calculate a result for one value of function parameter would be the same on the CPU and the GPU. On the GPU, the time taken could be even worse, because of the need to transfer data from RAM to the GPU. To prevent this, we need to have a large set of function parameters.

3 Computational results

Computer hardware:
CPU: Intel Core i7-9750H
GPU: NVIDIA GeForce GTX 1650

3.1 Experiment no. 1

Function parameter size: 1,000,000.
Number of rounds: 1.

As shown in table 1, it is clear that the calculation is much faster on the GPU than on the CPU. The calculation on the CPU took 4.61113 seconds, whereas on the GPU it took 0.25591 seconds - **18x** faster.

	CPU	GPU
Round 1	4.61113	0.25591

Table 1: The results of experiment no. 1.

3.2 Experiment no. 2

Function parameter size: 1,000,000.
Number of rounds: 10.

As shown in table 2, in round 1, the calculation on the CPU took 4.65273 seconds and on the GPU it took 0.26524 seconds - similar to 3.1. However, something strange can be seen. GPU execution time is much faster in round 2 - from 0.26524 seconds to 0.03926 seconds. That is a **6x** speedup. The reason behind this is that the Numba JIT [3] compiler compiles code on the first run, which can take some time.
So, if we were to ignore the first run, the program on the GPU runs **117**x faster than the program on the CPU.

	CPU	GPU
Round 1	4.65273	0.26524
Round 2	4.66322	0.03926
Round 3	4.68097	0.03944
Round 4	4.43533	0.03967
Round 5	4.55558	0.03965
Round 6	4.45576	0.03940
Round 7	4.39223	0.03941
Round 8	4.61270	0.03971
Round 9	4.52220	0.03955
Round 10	4.39434	0.04070

Table 2: The results of experiment no. 2.

[3] Just In Time.

3.3 Experiment no. 3

Function parameter size: 10,000,000.
Number of rounds: 1.

As shown in table 3, we can see the trend. The calculation on the GPU is much faster than on the CPU. With 10 million elements, the program on the CPU takes 47.4861 seconds and on the GPU it takes 0.63119 seconds - **75x** speedup.

	CPU	GPU
Round 1	47.4861	0.63119

Table 3: The results of experiment no. 3.

3.4 Experiment no. 4

Function parameter size: 10,000,000.
Number of rounds: 10.

The results stay similar if we increase the number of rounds. The first round on the GPU is slower compared to the second round on the GPU - 1.5x slower. Compared to the CPU, it is still much faster - **116x** faster.

	CPU	GPU
Round 1	45.54999	0.61494
Round 2	44.97076	0.40087
Round 3	45.21332	0.40623
Round 4	49.11119	0.40715
Round 5	47.85540	0.39750
Round 6	47.14230	0.40118
Round 7	46.52187	0.40250
Round 8	47.30745	0.39649
Round 9	48.05168	0.41060
Round 10	47.17812	0.40337

Table 4: The results of experiment no. 4.

3.5 Experiment no. 5

Function parameter size: 20,000,000.
Number of rounds: 2.

With the results from table 5 (92.44827 seconds in contrast to 0.80460 seconds), we can try to generalize data:

- A program execution time on the GPU is approximately **115x** faster than on the CPU.
- When we double parameters size, execution times doubles.

	CPU	GPU
Round 1	95.92450	1.03259
Round 2	92.44827	0.80460

Table 5: The results of experiment no. 5.

4 Summary and conclusions

This paper has demonstrated the time difference when running a program on the CPU vs. on the GPU. When we in front of a hard, complex computable math problem, it's advisable to consider running a program on the GPU.

Lastly, we used a linear regression model to determine the execution time for arbitrary input size. As can be seen from figure 1, CPU execution time is linear, whereas GPU execution time is close to constant. If we were to set the size of the parameters to 500 million, the CPU would take 2312 seconds $\sim$ 38 minutes, but the GPU would take 20 seconds, which is **115x** faster.
Additional notes: running a program on the GPU is only advisable if specific requirements are met [5, 6]:

- Complex calculation.
- Large input.

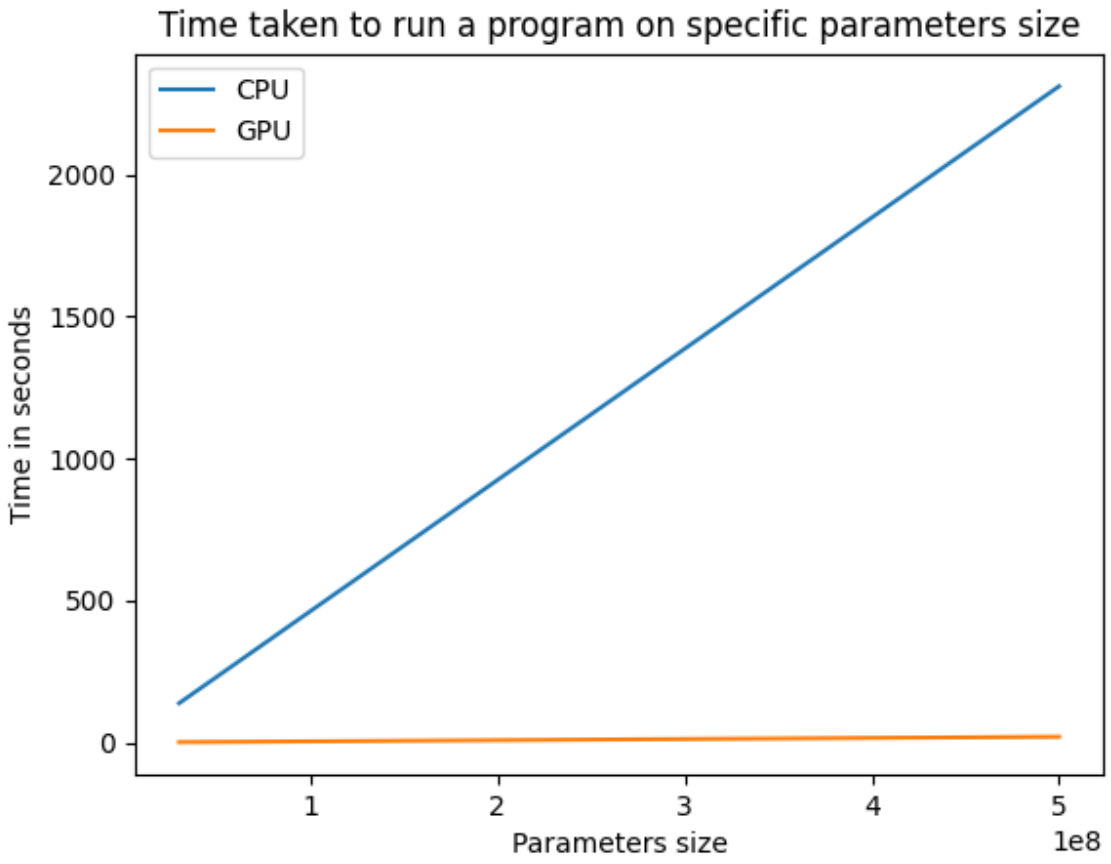

Figure 1: Linear regression model.

References

[1] Adam Shepherd. *What is a GPU?* 20th Jan. 2021. URL: `https://www.itpro.co.uk/hardware/30399/what-is-a-gpu` (visited on 27/02/2021).

[2] *Gaussian function*. URL: https://en.wikipedia.org/wiki/Gaussian_function (visited on 27/02/2021).

[3] *Normal distribution*. URL: https://en.wikipedia.org/wiki/Normal_distribution (visited on 27/02/2021).

[4] Jim Frost. *Normal Distribution in Statistics*. URL: https://statisticsbyjim.com/basics/normal-distribution/ (visited on 27/02/2021).

[5] Stan Seibert. *GTC 2017 Numba Tutorial Notebook 2: CUDA Basics*. 27th Apr. 2017. URL: https://github.com/ContinuumIO/gtc2017-numba/blob/master/2%20-%20CUDA%20Basics.ipynb (visited on 27/02/2021).

[6] *General-purpose computing on graphics processing units*. URL: https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units (visited on 27/02/2021).

A Source code

```
import json
import numpy as np
from numba import jit
from timeit import default_timer as timer

# Constant, used in the formula.
# Defined here to speed up the calculation, i.e. it's calculated only once
# and then placed in the formula.
SQRT_2PI = np.float32(np.sqrt(2 * np.pi))


# This function will run on the CPU.
def gaussian_cpu(values, mean, sigma):
    """Calculate values of the Gaussian function.

    :param values: list, function input parameters.
    :param mean: float, arithmetic mean.
    :param sigma: float, standard deviation.
    :return: list.
    """
    result = np.zeros_like(values)
    for index, item in enumerate(values):
        result[index] = (1 / (sigma * SQRT_2PI)) * (np.e ** (-0.5 * ((item - mean) / sigma) ** 2))
    return result


# This function will run on the GPU.
gaussian_gpu = jit(gaussian_cpu)


def write_to_file(name, values):
    """Write results to a file.

    :param name: string, file name, only prefix.
    :param values: dictionary, values to write.
    """
    with open(name + ".json", 'w') as f:
        json.dump(values, f, indent=4)

```

```
40
41  if __name__ == "__main__":
42      # Randomly generated values.
43      x = np.random.uniform(-3, 3, size=1000000).astype(np.float32)
44      # Randomly generated mean.
45      m = np.random.uniform(1, 10)
46      # Randomly generated standard deviation.
47      s = np.random.uniform(1, 10)
48
49      # The number of rounds.
50      n = 1
51
52      # Used to store execution time.
53      time_results = {}
54
55      for i in range(n):
56          start = timer()
57          gaussian_cpu(x, m, s)
58          end = timer() - start
59          time_results[i] = end
60
61      write_to_file("cpu", time_results)
62
63      for i in range(n):
64          start = timer()
65          gaussian_gpu(x, m, s)
66          end = timer() - start
67          time_results[i] = end
68
69      write_to_file("gpu", time_results)
```

B Plotting data

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from sklearn.linear_model import LinearRegression
4
5  if __name__ == "__main__":
6      # Set parameters.
7      input_parameters = np.array([1000000, 10000000, 20000000]).reshape((-1, 1))
8      cpu_time = np.array([4.61113, 44.97076, 92.44827])
9      gpu_time = np.array([0.03926, 0.39649, 0.80460])
10
11     # Create a linear regression model.
12     model_cpu = LinearRegression().fit(input_parameters, cpu_time)
13     model_gpu = LinearRegression().fit(input_parameters, gpu_time)
14
15     # Generate values for prediction.
16     x_predict = np.arange(30000000, 510000000, 10000000).reshape((-1, 1))
17
18     # Calculate predictions.
19     y_predict_cpu = model_cpu.predict(x_predict)
20     y_predict_gpu = model_gpu.predict(x_predict)
```

```
21
22      # Plot values.
23      plt.plot(x_predict, y_predict_cpu, label="CPU")
24      plt.plot(x_predict, y_predict_gpu, label="GPU")
25
26      # Display plot info.
27      plt.xlabel('Parameters size')
28      plt.ylabel('Time in seconds')
29      plt.title('Time taken to run a program on specific parameters size')
30
31      # Show legend and plot.
32      plt.legend()
33      plt.savefig("image.jpg")
```