Optimal Choice of CPU/GPU/TPU Under Different Matrix Computations

Zhanhao Zhang
Columbia University
New York, NY 10027
zz2760@columbia.edu

Lixian Chen
Columbia University
New York, NY 10027
lc3359@columbia.edu

Jingbin Cao
Columbia University
New York, NY 10027
jc5514@columbia.edu

Summary

During this age of rapid changes in technology for enhanced computing power, processor types of CPU, GPU and TPU have all become more accessible for intensive computation tasks. When given the options, determining which processor to use for different computation operations is not a simple task as it seems to be, by selecting the more advanced processor for more complicated tasks. To better understand the performance of different processors for different operations, we specifically experimented with 3 types of matrix operations (addition, multiplication and inversion), and with matrix size $N \times N$ for N ranging from 10 to 1280 (namely 10, 20, 40, 80, 160, 320, 640, 1280). We had the computer record the runtime (i.e. the program execution time) and did repeated measurement of 5 trials for each setting. With 3 processor types, 3 matrix operation types and 8 matrix size levels, we ran the experiments at 72 different settings. We examined the effect of each of these 3 variables (processor, matrix operation, matrix size) on the runtime under different settings using regression analysis and analysis of variance (ANOVA). Our results show that all 3 factors have effects on the runtime, although the effects vary between different processors and matrix operation combinations. In addition, the matrix size affects the runtime more or less in different settings, but in general, larger matrix sizes have greater impact on the runtime than smaller matrix sizes. We were also able to find some rough threshold values of matrix sizes under certain circumstances, to aid the decision making of choice of processors for specific matrix operation. Finally, we also briefly mention several potential operational factors and issues that can obscure our findings and thus make it still a challenging problem to tackle.

Introduction

With the advance in technology in the 21st century, we are blessed with high computing power and innovated hardware. In recent days, the computing hardware we can access to are Central Processing Unit (CPU), Graphics Processing Unit (GPU), and Tensor Processing Unit (TPU).

CPU is the basic processor that is possessed by most of the computers, laptops, and servers. GPU [1] leverages highly parallel structure that can achieve higher efficiency than CPU. With the advent of GPU, tasks that require intensive computations, for instance, neural networks, become probable. In 2016, Google released the TPU, which is specificially designed for Google's Tensorflow framework that is mostly used in machine learning. TPU is well suited for high volume of low precision computations. It cuts down memory and runtime overhead by reducing the precision of numbers but allowing for more input and output operations.

In practice, however, it is not the case that the more advanced processor you use, the more efficient computations you can achieve. The highly parallel structure introduces a lot of overhead cost for the communications among individual units. In addition, as the majority of tasks are still performed by

[1]The CUDA device in Nvidia-GPU is largely used in machine learning.

CPU in most of the computers, we will have to copy the data from CPU into GPU or TPU device each time we want to leverage these high technologies. When we are done with these computations, we also have to copy these data back to CPU in order to proceed with further analysis or visualizations. The copying procedure also takes some time, and it is not necessarily worthwhile if the tasks can be finished by the CPU processor alone, with minimal execution time.

In this project, we aim to understand the optimal choice of CPU, GPU, and TPU processors under different settings. Since matrix computations are the most common tasks in machine learning, we will focus only on the basic computations of the matrix to make the results across different processors comparable.

Experiment

To gather the data, we conduct experiments on different computations using each of these processors. The factors we have are Processor (CPU/GPU/TPU), Matrix Operation (Addition, Multiplication, Inversion), and Matrix Size $N \times N$ (N = 10, 20, 40, 80, 160, 320, 640, 1280).

For each combination of processor, matrix operation and matrix size, we randomly generate a matrix of values following standard normal distribution, using the same random seed 5291. Then, we perform the computations corresponding to the matrix operation for 1000 times and record the program execution time. Notice that the runtime we record is actually 10000 fold of the execution time of the matrix computations. Runtime being too small can be severely impacted by the noises from the background running tasks and therefore will not be representative. For each of these experiment, we conduct 5 trials of repeated experiments so that we can further mitigate the effect from background noises.

All experiments are conducted on Google Colab, which allows users to freely select the processors in the backend. Since we have to restart a new runtime session whenever we switch to a new processor, we conduct the experiments in 3 batches, with each batch comprised of all combinations of matrix sizes and matrix operations. Before each batch of experiments, we run the experiment using a matrix size of 10 and the matrix operation addition for 100 times without recording the execution time. This is intended to warm up the processors so that we won't be having a cold start of the machine.

Results

Data Visualization

We first visualized the mean of runtime (in seconds) plotted on the log-scale from five trials for each type of Processor, Matrix Operation and Matrix Size, grouped by three matrix operations: addition, multiplication and inversion (See Fig. 1).

In Fig. 1, when doing matrix addition, for CPU only, the log runtime becomes larger when matrix size gets larger. The increasing trend of runtime as matrix size increases is quite obvious and the positive correlation is self-evident. For GPU and TPU, when doing the matrix addition, they have similar log runtimes regardless of the different matrix sizes, and TPU shows better performance than GPU. When doing matrix multiplication and inversion, both CPU and GPU have larger runtimes when matrix size gets larger.

For the larger matrix size levels, GPU runs generally faster than CPU, but for smaller matrix size levels, CPU has slightly smaller log runtime than GPU. By contrast, when doing matrix multiplication and inversion, TPU has similar log runtimes for all these different matrix sizes, and it runs faster than CPU and GPU at almost all matrix sizes. TPU's advantage in matrix inversion operation is distinct and consistent at all matrix size levels. Based on these preliminary findings, we performed further analysis (linear regression and ANOVA) in the next few sections.

Matrix Operation Effect

First, based on the interaction plot in Fig. 2, the effect of smaller matrix size (i.e. matrix size < 320) on each operation is minimal, but there's a dramatic increase of mean runtime at around matrix size 640 for both multiplication and inversion. The larger size effect is most distinct for inversion. The mean runtime for addition stays at the low level for all the matrix sizes we experimented with. These

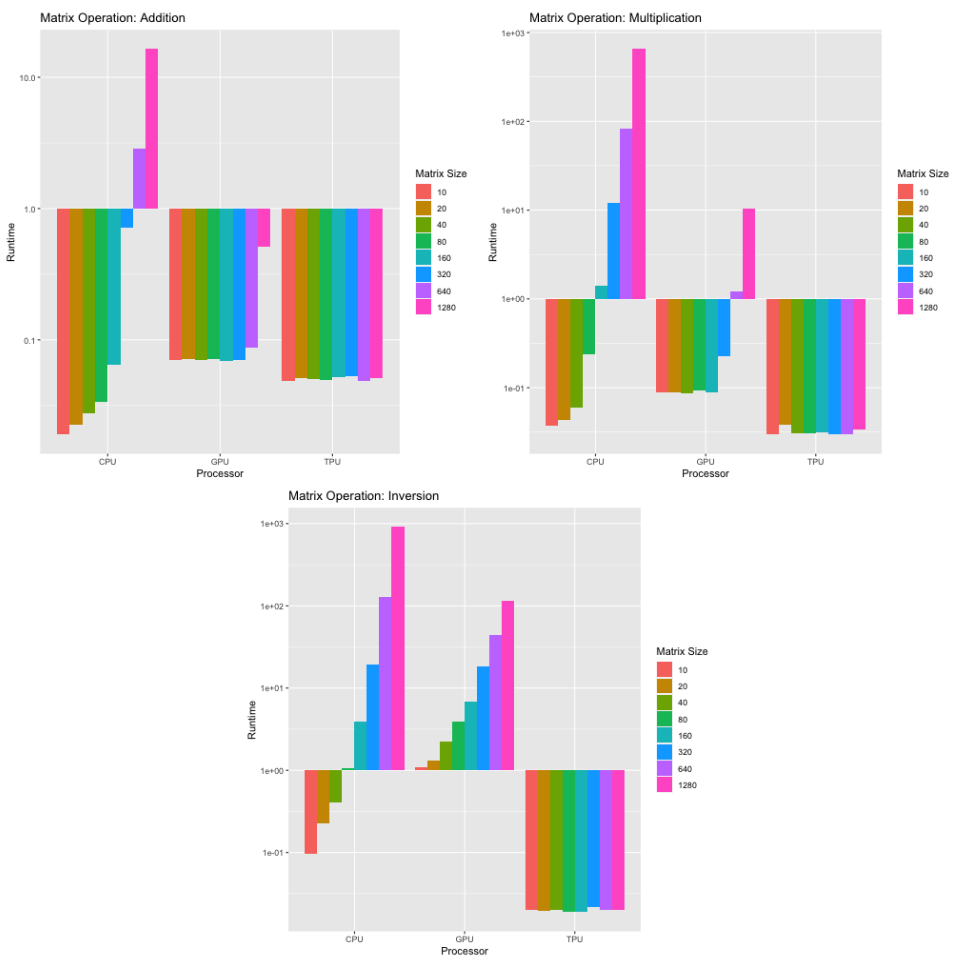

Figure 1: Data exploration for different processors, matrix operations and matrix sizes

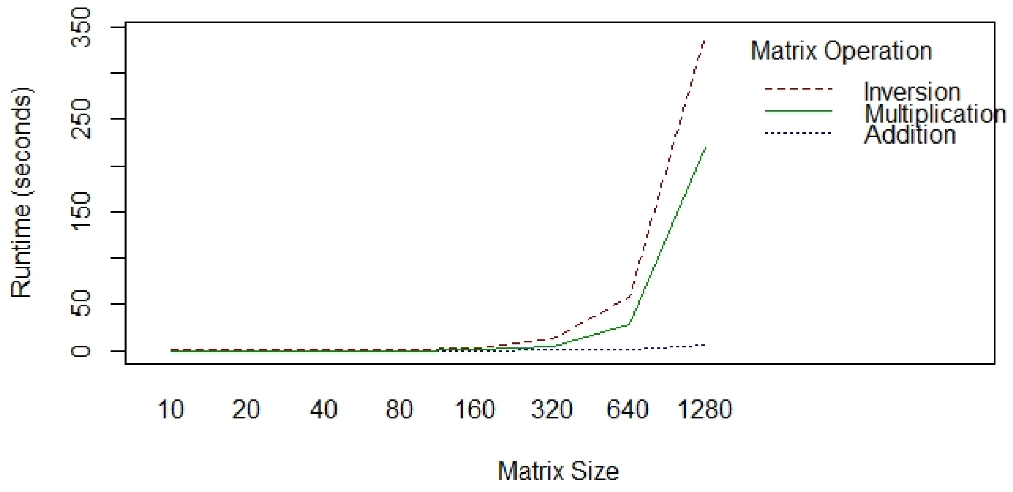

Figure 2: Interaction plot across matrix operations

findings are observed when we have pooled all 3 processors together, so the individual processor's pattern is masked. Hence, further analysis is conducted by examining the interaction plots after separating the 3 different processors (See Fig. 4). Another approach which is also adopted is that we further investigated and compared the runtime performance of different operations at matrix size level of 320, 640 and 1280, respectively, for the different processors.

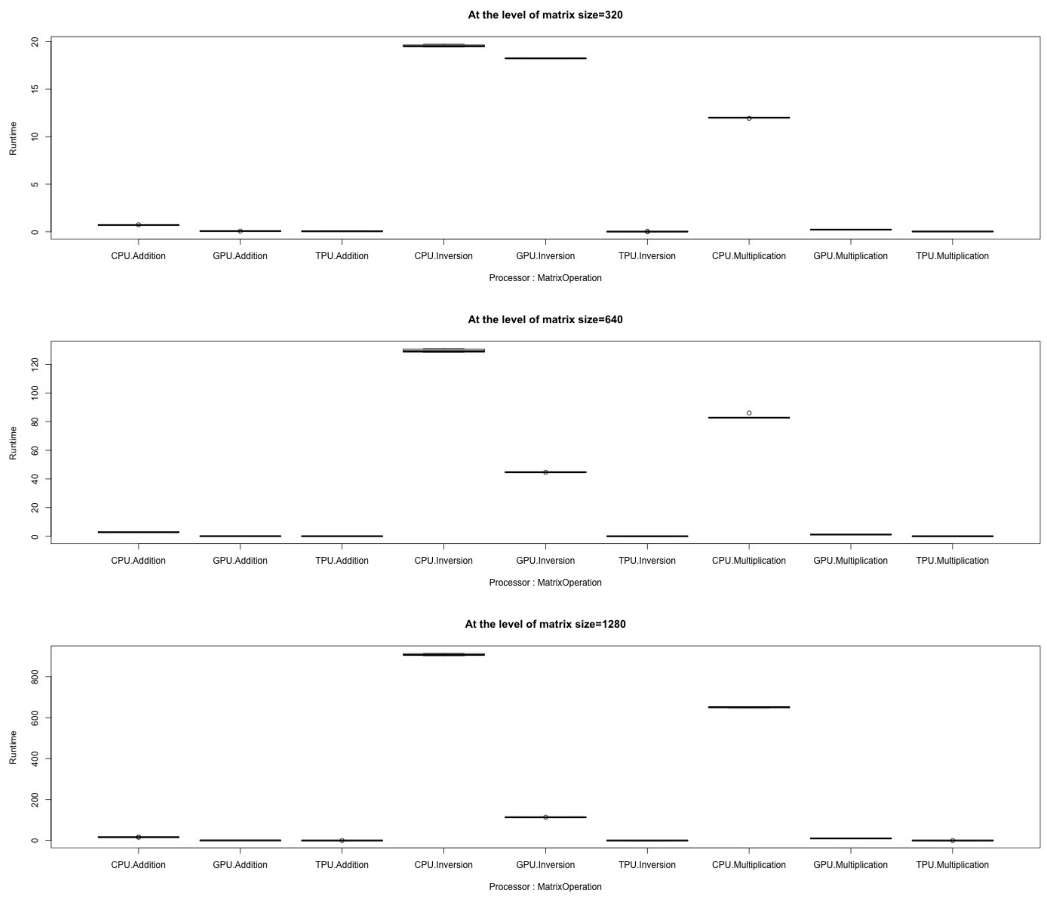

Figure 3: Boxplots at 3 matrix size levels (320, 640 and 1280)

Using boxplots at each of the 3 the matrix size levels (namely, 320, 640 and 1280), we discern the following preliminary characteristics. At matrix size level of 320, CPU for inversion is most time-consuming, and CPU for multiplication as the second most time-consuming, followed by GPU for inversion. The other operation and processor combinations have quite similar mean runtime performance. At matrix size level of 640 and 1280, the ranking for the top 3 time-consuming ones is the same as that at the matrix size level of 320. All the other operation and processor combinations have comparable mean runtime performance, i.e. they do not differ very much in runtime.

After seperating the 3 processors, the pattern of the change of mean runtime by matrix size for different processors and operations are shown in Fig. 4. CPU and GPU exhibit similar patterns of increasing runtime as matrix size increases for inversion and addition. For inversion, the rate of increase in runtime is much larger for CPU than the rate of increase in runtime for GPU. For multiplication, the larger matrix sizes have a much greater effect on the increase of runtime in the case of CPU than GPU. TPU exhibits drastically different patterns of the runtime as the matrix size increases, especially in the case of inversion and multiplication. For all runs of each type of setting using TPU, the runtime stays below 0.07 seconds. The pattern within each operation type for TPU as matrix sizes increase is relatively flat with minor fluctuations, rather than monotonically increasing. For TPU, addition is the most time-consuming operation, while inversion is the least time-consuming. Matrix sizes at the levels we have experimented do not impact the runtime dramatically. It's tentative

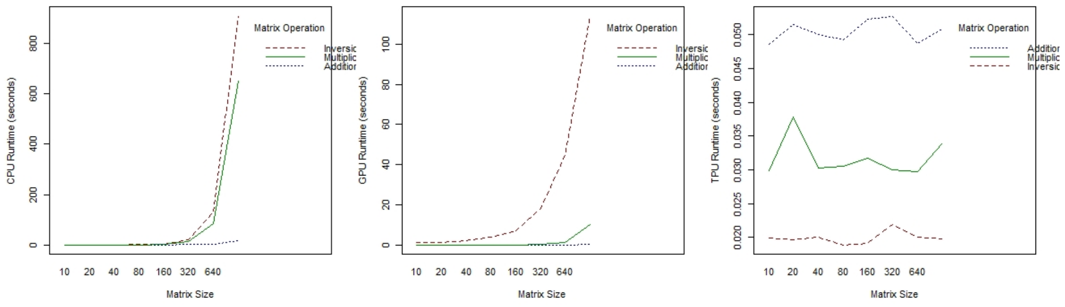

Figure 4: Interaction plots across matrix operations on different processors

to infer that TPU is highly proficient for all three types of operations at all of the matrix size levels we have experimented with.

	Addition	Inversion	Multiplication
CPU	2.5147	132.8553	93.4747
GPU	0.1280	24.0220	1.5264
TPU	0.0505	0.0199	0.0317

Table 1: Mean runtime (in seconds) by processor and matrix operation

Based on Table 1, Fig. 1 and Fig. 4, for CPU, the mean runtime of addition is much smaller than the mean time of multiplication and inversion, and its mean runtime of multiplication is close to its mean runtime of inversion. For GPU, the mean runtime of addition is also smaller than the mean time of multiplication and inversion. For TPU, the mean runtime of the 3 operations are similar to each other. For all 3 operations, the mean runtime of CPU is always the longest, and the mean runtime of TPU is always the shortest. For addition, the runtime performance do not differ very much among the 3 processors. For multiplication, the mean runtime is much higher for CPU, compared to the mean runtime of the other two processors, and TPU slightly outperforms GPU. For inversion, the difference between mean runtime of all 3 processors is quite obvious, with TPU outperforms GPU, and GPU outperforms CPU.

A 2-way ANOVA was conducted to compare the effects of processor, the operation type on the runtime, at each fixed level of the 3 matrix size (320, 640, and 1280). At each matrix size level, there was a significant main effect for processor ($p < 0.001$), and also a significant main effect for the operation type ($p < 0.001$), and a significant interaction effect between processor and operation type ($p < 0.001$).

A 3-way ANOVA was conducted to compare the effects of processor, the operation type and matrix size on the runtime. There was a significant main effect for the processor type ($p < 0.001$), a significant main effect for the operation type ($p < 0.001$), and only 1280 matrix size has a significant effect ($p < 0.001$) among all the matrix size levels. This further confirms that the larger matrix size affects the runtime more than the smaller matrix sizes. The matrix size effect is further investigated in the next section.

Matrix Size Effect

In this section, we studied the effect of Matrix Size when both Processor and Matrix Operation are held equal. Therefore, we have nine different pairs of processors and matrix operations, which are CPU and Addition, CPU and Multiplication, CPU and Inversion, GPU and Addition, GPU and Multiplication, GPU and Inversion, TPU and Addition, TPU and Multiplication, TPU and Inversion. Besides, we treated Matrix Size as both categorical and continuous, so we have two different designs for the Matrix Size effect.

Detecting Matrix Size Effect

Considering Matrix Sizes as categorical, we denote μ_1, μ_2, μ_3, μ_4 for mean of runtime at matrix size level of 160, 320, 640 and 1280. We want to test if different Matrix Sizes have different effects on nine combinations of processors (3 levels) and matrix operations (3 levels).

Therefore, the null hypothesis is:

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4$$

and the alternative hypothesis:

$$H_a : \mu_i \neq \mu_j \text{ for at least one } i \neq j$$

We used **aov** command in **R** to get ANOVA tables for all nine pairs (see Table 2). Besides, we transferred the Runtime to log(Runtime) using **log10** function in **R**.

Processor And Operation	P-Value	F-value
CPU + Addition	2×10^{-16}	76040
CPU + Multiplication	2×10^{-16}	393855
CPU + Inversion	2×10^{-16}	653718
GPU + Addition	2×10^{-16}	10205
GPU + Multiplication	2×10^{-16}	34853
GPU + Inversion	2×10^{-16}	4518902
TPU + Addition	0.849	0.487
TPU + Multiplication	0.707	0.562
TPU + Inversion	0.936	0.446

Table 2: P-values, F-values for nine pairs

In Table 2, at the significant level of $\alpha = 0.05$, we reject null hypothesis H_0 for the six pairs which include either CPU or GPU. In other words, there are matrix size effects for the following pairs, CPU and Addition, CPU and Multiplication, CPU and Inversion, GPU and Addition, GPU and Multiplication, and GPU and Inversion since the p-values for those six pairs are less than 0.05 and F-values are greater than test statistics $F_{(0.95,3,16)} = 3.2389$. By contract, there is no matrix size effect for the three pairs which include TPU, TPU and Addition, TPU and Multiplication, and TPU and Inversion.

More specifically, using CPU for matrix addition, the main effect of matrix size is significant, F(3,16) = 3.24, F = 76040, p < 0.05. Using CPU for matrix multiplication, the main effect of matrix size is significant (F = 393855, p < 0.05). Using CPU for matrix inversion, the main effect of matrix size is significant (F = 653718, p < 0.05). Using GPU for matrix addition, the main effect of matrix size is significant (F = 10205, p < 0.05). Using GPU for matrix multiplication, the main effect of matrix size is significant (F = 34853, p < 0.05). Using GPU for matrix inversion, the main effect of matrix size is significant (F = 4528902, p < 0.05). Under the following 3 settings, the matrix size effect is not significant. Using TPU for matrix addition, the main effect of matrix size is insignificant (F = 0.487, p > 0.05). Using TPU for matrix multiplication, the main effect of matrix size is insignificant (F = 0.562, p > 0.05). Using TPU for matrix inversion, the main effect of matrix size is insignificant (F = 0.446, p > 0.05).

Therefore, there is a significant effect of matrix size for CPU & Addition, CPU & Multiplication, CPU & Inversion, GPU & Addition, GPU & Multiplication, and GPU & Inversion. However, there is no significant effect of matrix size for TPU & Addition, TPU & Multiplication, and TPU & Inversion.

We also checked the validity of the nine models by generating the residual plot for each pair (See Figure 5). It shows that the zero-mean assumptions of residuals roughly hold for all models, as the residuals seem to be randomly distributed on both sides of 0, which implies that the constant variance assumption also holds.

One exception is that in the plot of GPU for addition, the data are very noisy for the smaller fitted values, compared to other fitted values. One possible explanation is that when the program execution time is very short, it can be severely impacted by the background tasks on the server, which may slow down our experiment script by a different amount, depending on how much computing power they take up during the experiment trials. In addition, as we log-transform our response variable[2] in order to satisfy the constant variance assumption, the log function is very sensitive to small values, which can also enlarge the noise.

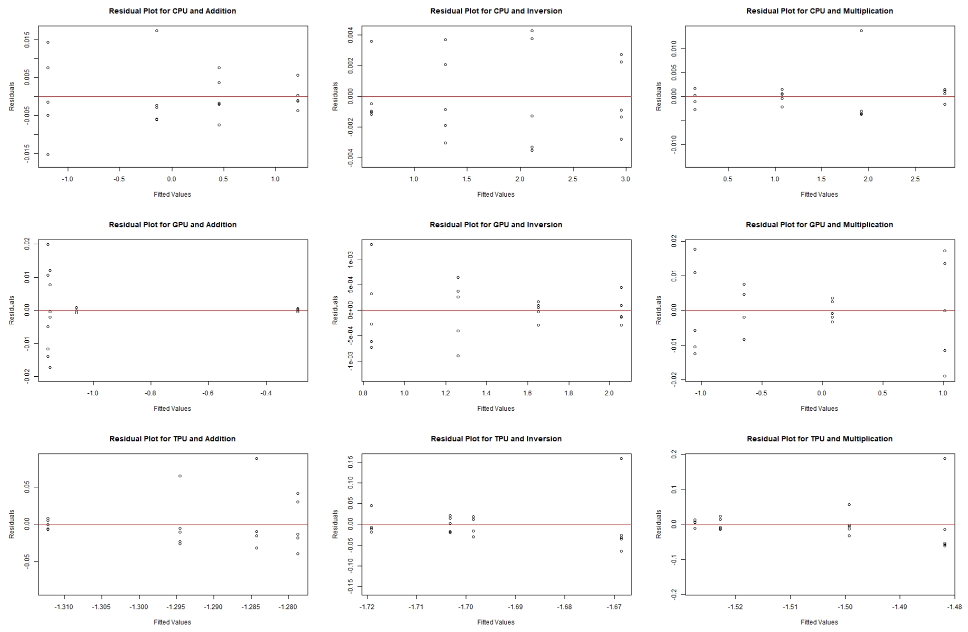

Figure 5: Residual plots for different processor & matrix operation combinations using categorical matrix size

Predict Runtime by Different Matrix Sizes

In the previous subsection, we found that different matrix sizes have different treatment effects on the six pairs as stated above. In this section, we use regression analysis to study the relationship between runtime and matrix size, and possibly to predict runtime for other matrix sizes through extrapolation, given certain combinations of processor and operation. Thus, we treated different matrix sizes (N = 10, 20, 40, 80, 160, 320, 640, 1280) as continuous numerical values, and then we ran linear regressions on the six pairs including CPU or GPU, which are CPU and Addition, CPU and Multiplication, CPU and Inversion, GPU and Addition, GPU and Multiplication, GPU and Inversion, and TPU and Addition. Besides, we transformed the Runtime to log(Runtime) using **log10** function in **R**.

Then, we used **lm** command in **R** to obtain simple linear regression models for the six pairs, and we got the following result in Table 3, including β_0, β_1, Standard Errors, and Adjusted R^2.

In Table 3, since all of the confidence intervals do not contain 0 (although some intervals are only slightly above zero), we conclude that all β_0 and β_1 for six linear regressions are significant. In addition, the adjusted R^2 for all six models are larger than 0.8 and the standard errors for all β_0 and β_1 are lower than 0.12, so we presume the validity of these models.

Moreover, when observing the values throughout the six pairs, we find that the β_1s from the pairs including GPU are generally smaller than the β_1s from the pairs including CPU for all three matrix operations. Therefore, when the matrix size gets larger, for all three matrix operations, GPU would be more efficient in terms of lower runtime.

Processor Effect

Last but not least, we assess the effect of processors by fixing the settings of matrix size and matrix operations and then conduct ANOVA tests on different levels in the processor effect. We set the

[2]The transformation power is suggested by the Box-Cox.

Processor And Operation (**Adjusted** R^2)	β_0 (Std. Error) (95 % CI)	β_1 (Std. Error) (95 % CI)
CPU + Addition (0.9093)	–1.5277230 (0.0646) (-1.6586 to -1.3969)	0.0646279 (0.0001) (0.0022 to 0.0027)
CPU + Multiplication (0.8522)	-0.9187389 (0.1196) (-1.1608 to -0.6767)	0.0033880 (0.0002) (0.0029 to 0.0039)
CPU + Inversion (0.8611)	-0.324820 (0.1003) (-0.5280 to -0.1217)	0.002947 (0.0002) (0.0026 to 0.0033)
GPU + Addition (0.8393)	-1.2307302 (0.0232) (-1.2776 to -1.1839)	0.0006241 (0.00004) (0.0005 to 0.0007)
GPU + Multiplication (0.9845)	-1.1480 (1.822×10^{-2}) (-1.1849 to -1.1111)	1.715×10^{-03} (3.487×10^{-5}) (0.0016 to 0.0018)
GPU + Inversion (0.8433)	0.3755794 (0.0557) (0.2628 to 0.4884)	0.0015252 (0.0001) (0.0013 to 0.0017)

Table 3: Values, Std. Error, and 95% CI of β_0 & β_1 and Adjusted R^2 for Six Pairs

CPU as the baseline and then compute the difference of average runtime between GPU and CPU and between TPU and CPU. We determine the significance by their corresponding p-values.

As shown in the Table 4, among all different combinations of matrix size and matrix operations, there are 3 occasions where GPU or TPU does not have significant difference with CPU, which are multiplications on matrix size 10 and 20 using TPU and additions on matrix size 160 using GPU.

Using a matrix size of 10 for addition, the effect of GPU is significant ($t = 54.56$, $p < 0.05$) and the effect of TPU is significant too ($t = 31.78$, $p < 0.05$). Using a matrix size of 10 for multiplication, the effect of GPU is significant ($t = 9.284$, $p < 0.05$), but the effect of TPU is not significant ($t = -1.36$, $p > 0.05$). Using a matrix size of 10 for inversion, the effect of GPU is significant ($t = 97.713$, $p < 0.05$), and the effect of TPU is also significant ($t = -7.653$, $p < 0.05$).

Using a matrix size of 1280 for addition, the effect of GPU is significant ($t = -334.7$, $p < 0.05$) and the effect of TPU is significant ($t = -344.5$, $p < 0.05$). Using a matrix size of 1280 for multiplication, the effect of GPU is significant ($t = -763.5$, $p < 0.05$) and the effect of TPU is also significant ($t = -775.7$, $p < 0.05$). Using a matrix size of 1280 for inversion, the effect of GPU is significant ($t = -435.8$, $p < 0.05$) and the effect of TPU is also significant ($t = -498.3$, $p < 0.05$).

For GPU, the addition operation is significantly slower than TPU if the matrix size is 80 or smaller, but the addition operation is significantly faster than CPU when the matrix size is larger than or equal to 320. Similar patterns can be observed in multiplication and inversion. GPU will be faster than CPU only if the matrix size used for multiplication is at least 80 and the size used for inversion is at least 320.

For TPU, the addition operation is significantly slower than CPU if the matrix size is 80 or smaller, but it will be faster when the matrix size is at least 160. For multiplication, TPU is significantly faster than CPU only when the matrix size is at least 40. For inversion, TPU is significantly faster than CPU for all matrix sizes.

Conclusion

All else being equal, we would suggest avoid using CPU for multiplication or inversion for large matrix sizes if it is affordable to use GPU or TPU. GPU runs faster than CPU for both multiplication and inversion when the matrix size is relatively large, and the threshold sizes, according to our experiments, are 80 and 320 for multiplication and inversion, respectively. For multiplication, other than the matrix size of 10 and 20, TPU is significantly faster than CPU. For inversion, TPU is significantly faster than CPU for all matrix sizes.

For addition, using CPU is preferred for matrices with smaller size (specifically 80 or smaller), as GPU and TPU are significantly slower in such cases. However, when matrix size is 320 or above, GPU becomes significantly faster than CPU, for addition. Likewise, when the matrix size reaches 160 or above, TPU becomes more favorable than than CPU, for addition.

	Matrix Size	Addition (Std. Error) (95 % CI)	Multiplication (Std. Error) (95 % CI)	Inversion (Std. Error) (95 % CI)
GPU v.s. CPU	10	0.051 (0.001) (0.049 to 0.053)	0.051 (0.005) (0.04 to 0.062)	0.988 (0.01) (0.969 to 1.008)
	20	0.049 (0.002) (0.045 to 0.053)	0.045 (0.006) (0.034 to 0.056)	1.092 (0.006) (1.08 to 1.103)
	40	0.043 (0.001) (0.041 to 0.045)	0.027 (0.001) (0.024 to 0.029)	1.81 (0.006) (1.799 to 1.822)
	80	0.038 (0.001) (0.035 to 0.041)	-0.144 (0.005) (-0.155 to -0.134)	2.812 (0.006) (2.8 to 2.824)
	160	0.005 (0.003) (0 to 0.01)	-1.318 (0.003) (-1.323 to -1.313)	2.97 (0.008) (2.954 to 2.986)
	320	-0.643 (0.006) (-0.655 to -0.631)	-11.747 (0.014) (-11.774 to -11.721)	-1.329 (0.047) (-1.422 to -1.237)
	640	-2.758 (0.014) (-2.785 to -2.731)	-82.208 (0.542) (-83.27 to -81.145)	-84.806 (0.412) (-85.614 to -83.998)
	1280	-15.879 (0.047) (-15.972 to -15.786)	-640.291 (0.839) (-641.935 to -638.647)	-794.204 (1.822) (-797.776 to -790.632)
TPU v.s. CPU	10	0.03 (0.001) (0.028 to 0.031)	-0.007 (0.005) (-0.018 to 0.003)	-0.077 (0.01) (-0.097 to -0.058)
	20	0.029 (0.002) (0.025 to 0.033)	-0.006 (0.006) (-0.017 to 0.005)	-0.208 (0.006) (-0.219 to -0.196)
	40	0.022 (0.001) (0.02 to 0.024)	-0.029 (0.001) (-0.032 to -0.026)	-0.382 (0.006) (-0.393 to -0.37)
	80	0.016 (0.001) (0.013 to 0.018)	-0.206 (0.005) (-0.217 to -0.196)	-1.029 (0.006) (-1.04 to -1.017)
	160	-0.012 (0.003) (-0.017 to -0.007)	-1.375 (0.003) (-1.38 to -1.37)	-3.921 (0.008) (-3.937 to -3.904)
	320	-0.661 (0.006) (-0.673 to -0.649)	-11.944 (0.014) (-11.971 to -11.917)	-19.525 (0.047) (-19.617 to -19.433)
	640	-2.796 (0.014) (-2.823 to -2.769)	-83.386 (0.542) (-84.449 to -82.324)	-129.484 (0.412) (-130.292 to -128.676)
	1280	-16.341 (0.047) (-16.434 to -16.248)	-650.591 (0.839) (-652.235 to -648.947)	-908.058 (1.822) (-911.629 to -904.486)

Table 4: Values, Std. Error, and 95% CI for GPU and TPU

TPU is highly efficient in terms of very small runtime for all three types of operations, but we have to take into consideration the overhead costs of using TPU. Among all the matrix size levels we have tried, generally speaking, 320 may be used as a threshold value, when we try to decide on the choice the processor for the same matrix operation.

However, it is still challenging to decide on the choice of the processors, with considering of the different operation types and matrix sizes, due to the following operational limitations.

Recall that each time we select a new processor in the backend, we will have to restart a new runtime session, in which all variables and packages loaded in the previous session will disappear. Therefore, there is actually a blocking effect from the different runtime session in our experiment, and this blocking effect is confounded by our Processor factor. The blocking effect might be very small compared to the effect from different processors. However, as there is no easy way to separate out the effect of different runtime sessions from our experiments, we still have to address the limitation that our Processor effect is confounded by the blocking effect from the change in runtime sessions.

In addition, the Google Colab is infamous for its unstable computational power. In fact, the processors might be simultaneously serving multiple users. Therefore, it is not too surprising that from time to time, your program execution speed suddenly drop significantly even if all of your script and your data remain constant. Nevertheless, our experiments took place in a moderately short period of time, so significant changes in the impact from the other users seldomly take place. Yet, there is still a very small chance that our processors suddenly slow down or speed up midway during our experiments due to the activities of other users.

Last but not least, it is a common practice in the machine learning to trade-off the computational time and space with precision of data. This is especially the case with TPU, which speeds up the computations by cutting down the number of bits for the representation of data. In some cases, researchers might need to compute numbers in the order of 10^{10} or 10^{-10}, so low precision in these tasks might give answers that are very far away from the ground truth. While we only investigate the execution time of programs under different settings, in future works, it is also important to assess the numerical stability of different processors before making a decision on which processor to use.

In spite of the potential confounding factors and operational limitations that may interfere with the runtime performance for our experiment, our findings can still provide some insights for the decision-making process of the optimal choice of processors for certain operations.