Graphics Characterization Analysis Report: Part II

Jason Mak and Yangzihao Wang

February 7, 2012

Modern programmable GPUs have become a more and more significant part of various applications with heavy computation tasks. However, the results of some of these applications still suffer from the loss of floating-point arithmetic operations precision. In this experiment we explore the numerical precision of some of the native arithmetic operations on a NVIDIA GTX560 GPU. We limit our experiment on single-precision floating point numbers. There are two purporses of this experiment: 1) Try to quantify and measure the difference between the results of the same arithmetic functions generated on GPU and on CPU. 2) Try to find some patterns of these errors.

The following table shows the setting of this experiment, including type of functions, parameters, domains, etc.

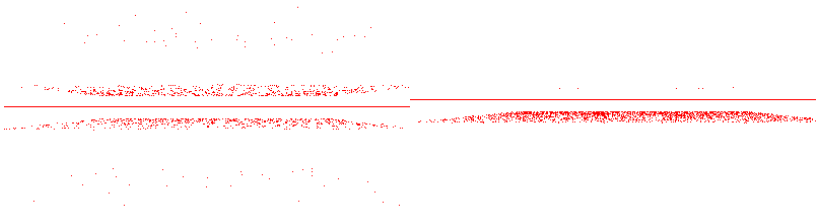
Function	Parameters	Domain
sinf	i*degree	[0, 65535]
cosf	i*degree	[0, 65536]
logf	i+1	[1, 65536]
expf	i/max_i*10.0	[0, 10]
powf	i, 1.01	[0, 65535]

After the appearance of CUDA, processors in GPU become unified. In our experiment, we use CUDA to exploit the numerical precision of the native arithmetic operations on processors inside the GPU. Here we define the precision to be the comparative precision. We consider it as the difference of the result of the same arithmetic operations between CPU and GPU. The basic idea is letting GPU and CPU do the same arithmetic operation and compare the differences between their results. We record these differences and level of error (in 10 to the power of n) in the file to analyze later. We also visualize the result using a VBO with different height value y as position value for each vertex. The (x, z) coordinates of vertices in the VBO will be between 0 and 256, the height value y represent the level of error, the further away one point from the plane $y = 0$, the more severe the error is.

First we analysis the proportion of the occurance of errors during our 65536 calls to each function. The result is shown in the following table:

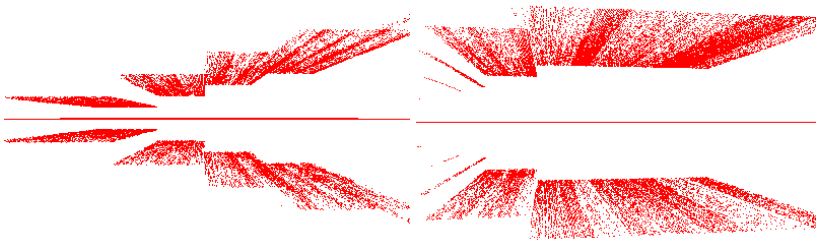
Function	Number of Errors	Proportion of the Errors
sinf	10211	15.581%
cosf	10197	15.559%
logf	1948	2.972%
expf	19730	30.106%
powf	26143	39.891%

From the result we can tell that on our data set, logarithm operation has the lowest error rate, while exponent and power operations have relatively higher error rate. This reveals that the implementation of logarithm operation might be less complex than the implementation of exponent and power operation so that it will preserve a higher level of precision.
Next we will look at the visualization result and try to find the patterns of the occurance of errors in each type of arithmetic operation. The camera is looking from the side of the unit cube where all the results are possible to fall in. Points on the left side has a smaller value as the input of the function. We define error level to be n so that the difference between the result generated on GPU and the result generated on CPU starts to appear on the 10 to the power of minus n bit in decimal.



(a) sinf function

(b) logf function



(c) expf function

(d) powf function

Figure 1: Visualization result of errors in each type of arithmetic operation.

From the graph we can find out some interesting patterns of the occurences of errors in different type of functions. In sinf(cosf is similar), the errors distribute evenly across the data set. In logf function, all the error values are smaller than the original values. Both sinf and logf have smaller errors compare to expf and powf. Both expf and powf function have more error values and the level of errors are also higher. The error level in expf function is proportional to the input of the function and is near linear. However, the increasing of error level in powf function is non-linear and becomes large quite fast. From these four graphs we can conclude powf function has the worst precision on our data set.

Looking at powf and expf functions, we can conclude that the precisions of these two functions decrease according to the increasing of the input value of the functions. The following table will show this conclusion:

Function	Result Value	Change of Error Level
expf	16	from 7 to 6
expf	128	from 6 to 5
expf	751	from 5 to 4
expf	13758	from 4 to 3
powf	16	from 6 to 5
powf	106	from 5 to 4
powf	737	from 4 to 3
powf	11388	from 3 to 2
powf	73056	from 2 to 1

Our experiment is far from complete, however, we can draw the following conclusions from this experiment on a NVIDIA GTX560 GPU: 1) All native arithmetic operations have precision problems with single-precision floating number. 2) Among the five functions we test, logf seems to be the function with highest precision and powf seems to be the function with lowest precision. 3) When the result of the function gets large, the precision of the function will decrease.