Rounding Error Analysis in Weak Form Kernel for 1D Equation on GPU

Huiyu Xie

Apr 9, 2024

1 Introduction

In this article, rounding error analysis is applied to the 1D weak form kernel from our project. This kernel is selected since it is concise (no any further calculations such as flux computation), and it is close to simple matrix multiplication, a widely implemented kernel on GPU. This error analysis will reveal the error bound model for this kernel (can also be referred to as an algorithm) given any input data.

Generally, this work will go through a sample error analysis based on one of our GPU kernels to further address the question of error checking in algorithm implementation.

2 Models of Arithmetic

To carry out the rounding error analysis of our algorithm, we have to adopt models about the accuracy of the basic arithmetic operations. Given a floating point number system $F \subset \mathbb{R}$, for any $x, y \in F$,

$$fl(x+y) = (x+y)(1+\delta_1), \quad |\delta_1| \leq \mu_1 \tag{1}$$

$$fl(x \times y) = (x \times y)(1+\delta_2), \quad |\delta_2| \leq \mu_2 \tag{2}$$

Note that we are using $fl(\cdot)$ here to denote the computed result of the expression, and it can be treated as the rounding process in computers. In fact, (1) and (2) always hold for IEEE standard arithmetic, and similar models also apply to division and subtraction operations [1].

3 Algorithm Error Analysis

Note that in this analysis, we care about determining with determining the model for rounding errors in the algorithm, instead of obtaining a specific error bound.

The weak form kernel algorithm can be refactored as below

Algorithm 1 Weak Form Kernel

```
1: procedure WeakFormKernel(A, B, C)
2:     Input: A (3D matrix), B (3D matrix), C (2D matrix)
3:     m ← number of rows in A
4:     n ← number of columns in A
5:     p ← number of layers in A
6:     Launch m × n × p threads in parallel
7:     for all threads (i, j, k) parallel do
8:         for ii ← 1 to n do
9:             A[i, j, k] ← A[i, j, k] + B[i, ii, k] × C[j, ii]
10:        end for
11:    end for
12: end procedure
```

Algorithm 1 involves 3D matrices, but it can be treated as a variation of simple 2D matrix multiplication, since the basis of matrix multiplication is the inner product.

So the error analysis for Algorithm 1 can start with inner product. Consider the inner product $S_n = \boldsymbol{x}^T\boldsymbol{y}$, where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. Note that we assume each element of $\boldsymbol{x}$ and $\boldsymbol{y}$ belongs to F to make analysis simple. Furthermore, we assume the evaluation $S_n = x_1y_1 + \ldots + x_ny_n$ is from left to right. Let $S_i = x_1y_1 + \ldots + x_iy_i$ be the ith partial sum. By (1) and (2), we have

$$\hat{S_1} = fl(x_1y_1) = x_1y_1(1+\delta_1) \tag{3}$$

$$\begin{aligned}\hat{S_2} &= fl(\hat{S_1} + x_2y_2) = (\hat{S_1} + x_2y_2(1+\delta_2))(1+\delta_3) \\ &= x_1y_1(1+\delta_1)(1+\delta_3) + x_2y_2(1+\delta_2)(1+\delta_3)\end{aligned} \tag{4}$$

$$\begin{aligned}\hat{S_3} &= fl(\hat{S_2} + x_3y_3) = (\hat{S_2} + x_3y_3(1+\delta_4))(1+\delta_5) \\ &= x_1y_1(1+\delta_1)(1+\delta_3)(1+\delta_5) \\ &+ x_2y_2(1+\delta_2)(1+\delta_3)(1+\delta_5) + x_3y_3(1+\delta_4)(1+\delta_5)\end{aligned} \tag{5}$$

where $|\delta_i| \leq \mu$ for $i = 1, 2, 3, 4, 5$. Note that the hat notation is used to denote the computed result. Based on (3), (4), and (5), we can get the general pattern by refactoring each δ_i appears in the evaluation. Overall, we have

$$\begin{aligned}\hat{S_n} &= x_1y_1(1+\eta_1)^n + x_2y_2(1+\eta_2)^n \\ &+ x_3y_3(1+\eta_3)^{n-1} + \cdots + x_ny_n(1+\eta_n)^2\end{aligned} \tag{6}$$

There are various ways to simplify (6), and one elegant way is by applying the following result.

Lemma 1. *If $|\eta_i| \leq \mu$ and $\rho_i = \pm 1$ for $i = 1, ..., n$, and $n\mu < 1$, then*

$$\prod_{i=1}^{n}(1+\eta_i)^{\rho_i} = 1 + \theta_n$$

where

$$|\theta_n| \leq \frac{n\mu}{1-n\mu} =: \gamma_n$$

Proof. Proof skipped. □

Note that it can be easy to verify that any η_i in (6) satisfy $|\eta_i| \leq \mu$, and the condition $n\mu < 1$ is true in nearly any circumstance that might arise with IEEE floating point number arithmetic [1].

By applying Lemma 1 to (6), we get

$$\begin{aligned}\hat{S}_n = {} & x_1y_1(1+\theta_n) + x_2y_2(1+\theta'_n) \\ & + x_3y_3(1+\theta_{n-1}) + \cdots + x_ny_n(1+\theta_2)\end{aligned} \tag{7}$$

This result can be interpreted as follows: the computed result is the exact one for a perturbed set of data $x_1, ..., x_n, y_1(1+\theta_n), y_2(1+\theta'_n), ..., y_n(1+\theta_2)$ (alternatively, we can perturb x_i and leave y_i alone), and each perturbation is bounded by γ_n.

Previously we have assumed that the evaluation of $S_n = \boldsymbol{x}^T\boldsymbol{y}$ is from left to right, and it is easy to get that for any order of evaluation, we have

$$\begin{aligned}fl(\boldsymbol{x}^T\boldsymbol{y}) = (\boldsymbol{x}+\Delta\boldsymbol{x})^T\boldsymbol{y} = \boldsymbol{x}^T(\boldsymbol{y}+\Delta\boldsymbol{y}), \\ |\Delta\boldsymbol{x}| \leq \gamma_n|\boldsymbol{x}|, \quad |\Delta\boldsymbol{y}| \leq \gamma_n|\boldsymbol{y}|\end{aligned} \tag{8}$$

where the absolute value operation ($|\boldsymbol{x}|$ and $|\boldsymbol{y}|$, etc.) is applied elementwise and the inequality is also hold elementwise (and same for matrices later).

Now we can proceed the analysis of our weak form kernel. But first, to more directly apply the result (8) from the inner product, we have to perform the variation based on Algorithm 1. It is easy to see $A, B \in \mathbb{R}^{m\times n\times p}$ and $C \in \mathbb{R}^{n\times n}$, and let $A^{(k)} = (a_{ijk})_{i=1,...,m;\, j=1,...,n}$ denote the kth layer of A (and similar for B). Then we construct two matrices B' and C' as follows

$$B' = \left(\left(B'^{(k)}\right)_{k=1,...,p}\right) = \left(\left(B^{(k)}A^{(k)}\right)_{k=1,...,p}\right) \tag{9}$$

$$C' = \begin{pmatrix} C^T \\ I_n \end{pmatrix} \tag{10}$$

where I_n is the identity matrix of size n. Now we have $A^{(k)} = B'^{(k)}C'$, where $B' \in \mathbb{R}^{m\times 2n\times p}$ and $C \in \mathbb{R}^{2n\times n}$, and the variation of Algorithm 1 can be found as below

Algorithm 2 Weak Form Kernel Variation

1: **procedure** WEAKFORMKERNELVARIATION(A, B', C')
2: **Input:** A (3D matrix), B' (3D matrix), C' (2D matrix)
3: $m \leftarrow$ number of rows in A
4: $n \leftarrow$ number of columns in A
5: $p \leftarrow$ number of layers in A
6: **Launch** $m \times n \times p$ **threads in parallel**
7: **for all threads** (i, j, k) **parallel do**
8: **for** $ii \leftarrow 1$ **to** $2n$ **do**
9: $A[i, j, k] \leftarrow B'[i, ii, k] \times C'[ii, j]$
10: **end for**
11: **end for**
12: **end procedure**

Note that the error analysis for Algorithm 1 and 2 is the same (think about it), and lines 8 and 9 in Algorithm 2 are purely performing inner products between two vectors of size $2n$ (essentially size $n + 1$, think about it again).

To make analysis easier, we assume elements of input A, B', and C' can be represented exactly in floating point number system (i.e., belong to F). First consider the analysis on the kth layer of A and B'. Let $\boldsymbol{a}_j$ is the jth column in $A^{(k)}$, $\boldsymbol{b}_i^T$ be the ith row in $B'^{(k)}$, and $\boldsymbol{c}_j$ be the jth column in C'. Based on (8) we have

$$\hat{\boldsymbol{a}}_{ji} = (\boldsymbol{b}_i + \Delta\boldsymbol{b}_i)^T \boldsymbol{c}_j, \quad |\Delta\boldsymbol{b}_i| \leq \gamma_{n+1}|\boldsymbol{b}_i| \tag{11}$$

and as $\boldsymbol{a}_j = B'\boldsymbol{c}_j$ for $j = 1, ..., n$, we can further have

$$\hat{\boldsymbol{a}}_j = (B'^{(k)} + \Delta B_j'^{(k)})\boldsymbol{c}_j, \quad |\Delta B_j'^{(k)}| \leq \gamma_{n+1}|B'^{(k)}| \tag{12}$$

where $\Delta B_j'^{(k)}$ is the perturbation matrix for computing $\boldsymbol{a}_j$ and further

$$|\boldsymbol{a}_j - \hat{\boldsymbol{a}}_j| \leq \gamma_{n+1}|B'^{(k)}||\boldsymbol{c}_j| \tag{13}$$

and thus we can easily get

$$|A^{(k)} - \hat{A}^{(k)}| \leq \gamma_{n+1}|B'^{(k)}||C'| \tag{14}$$

and the corresponding normwise bounds include

$$\|A^{(k)} - \hat{A}^{(k)}\|_p \leq \gamma_{n+1}\|B'^{(k)}\|_p\|C'\|_p, \quad p = 1, \infty, F \tag{15}$$

where F in (15) denotes the Frobenius norm. The induction from (14) to (15) is easy to check, and proofs are skipped here to save space (and a complementary document will be provided later to complete these proofs).

Note that (15) holds for any k, so the error bound model is formulated layer-wise for our weak form kernel. For those interested in obtaining an exact value

of the error bound, please refer to [2] to first obtain μ_1 and μ_2 in equations (1) and (2), respectively. These values are closely related to *machine epsilon* (but not exactly the same, remember errors will accumulate).

Therefore, the error bound model for weak form kernel is found, and this model reflects the sensitivity of the product result to componentwise relative perturbations in the input data.

4 Caveat

There are some caveats that are worthwhile to mention here. Actually, our error model is constructed based on the general assumption that the design of the GPU (specifically, the NVIDIA GPU that is applied in our project) strictly follows the IEEE standard, but this is not the case for tensor cores[3]. Thus, this factor may (or may not, as some important design details have not been published by NVIDIA, so there is no absolute conclusion) render our error model unreliable in the end.

5 Conclusion

In general, rounding error analysis serves the purpose of demonstrating the existence of bounds on the effects of rounding errors on an algorithm. Ideally, these bounds are small for any problem data, but for some algorithms (which we call unstable), there is no useful error bound [1]. Now we can address the problem that concerns us (as a group).

> Generally, the most effective way to validate an algorithm implementation on a GPU is to adopt a heuristic approach.

This means that developers should focus on ensuring each step of the algorithm on the GPU is logically equivalent to its implementation on the CPU, instead of relying solely on error bound checking. However, error bound checking serves as a good method for double checking.

Here are the supporting reasons for the above conclusion:

1. The error falling within the error bound is only a necessary condition for correct algorithm implementation.
2. The error analysis for most algorithms is complicated and time-consuming.
3. The arithmetic behaviors of GPUs are not deterministic due to their intrinsic design for acceleration.

References

[1] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 2002.

[2] D. Goldberg, *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, ACM Computing Surveys (CSUR), vol. 23, no. 1, pp. 5–48, 1991. [Online]. Available: https://dl.acm.org/doi/10.1145/103162.103163.

[3] M. Fasi, N. J. Higham, M. Mikaitis, and S. Pranesh, *Numerical behavior of NVIDIA tensor cores*, PeerJ Computer Science, 7:e330, 2021. https://doi.org/10.7717/peerj-cs.330