Accelerating Levenshtein Edit Distance Using GPUs

Zhizhuo Yang
Rochester Institute of Technology
04/26/2018

Problem Definition

Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.[1]

For example, the Levenshtein distance between "kitten" and "sitting" is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

1. **k**itten → **s**itten (substitution of "s" for "k")
2. sitt**e**n → sitt**i**n (substitution of "i" for "e")
3. sittin → sittin**g** (insertion of "g" at the end).

[1] https://en.wikipedia.org/wiki/Levenshtein_distance#cite_note-4

		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Problem Definition

Levenshtein Distance Formula

$$H_{i,j} = min \begin{cases} H_{i-1,j-1} + Score \\ H_{i,j-1} + 1 \\ H_{i-1,j} + 1 \end{cases} \quad (1)$$

Score = 1 while $H_{i,j}$!= $H_{i-1,j-1}$

Score = 0 while $H_{i,j}$ == $H_{i-1,j-1}$

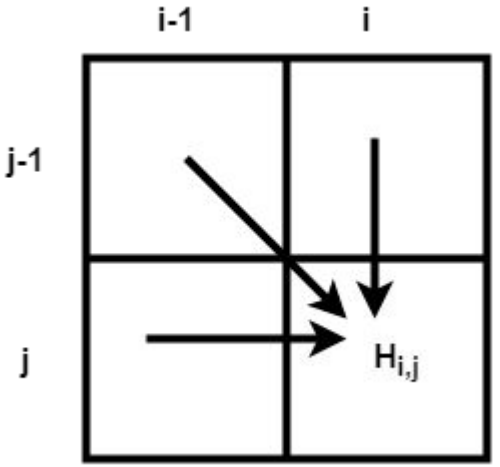
		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Problem Definition

Levenshtein Distance Formula

$$H_{i,j} = min \begin{cases} H_{i-1,j-1} + Score \\ H_{i,j-1} + 1 \\ H_{i-1,j} + 1 \end{cases} \quad (1)$$

Score = 1 while $H_{i,j}$!= $H_{i-1,j-1}$, Score = 0 while $H_{i,j}$ == $H_{i-1,j-1}$



Dependency !!!

Two main calculation methods:

1. Recursive
2. Iterative with full matrix (Dynamic Programming and divide-and-conquer)

   Complexity: O(nm) time and O(n+m) space

Solution

1. **Diagonal Technique in matrix**

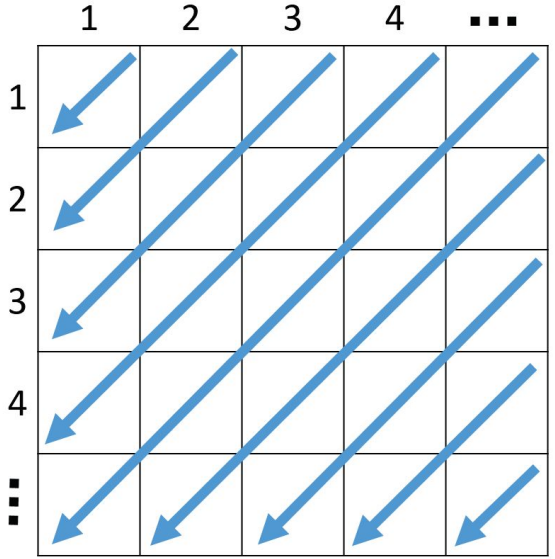
Index = row * width + column (2)

Row = index / width (3)

Column = index % width (4)

2. **Convert 2D array to 1D array**

Make the data transfer from the CPU side to the GPU



Solution

1. **Diagonal Technique in matrix**

Index = row * width + column (2)

Row = index / width (3)

Column = index % width (4)

2. **Convert 2D array to 1D array**

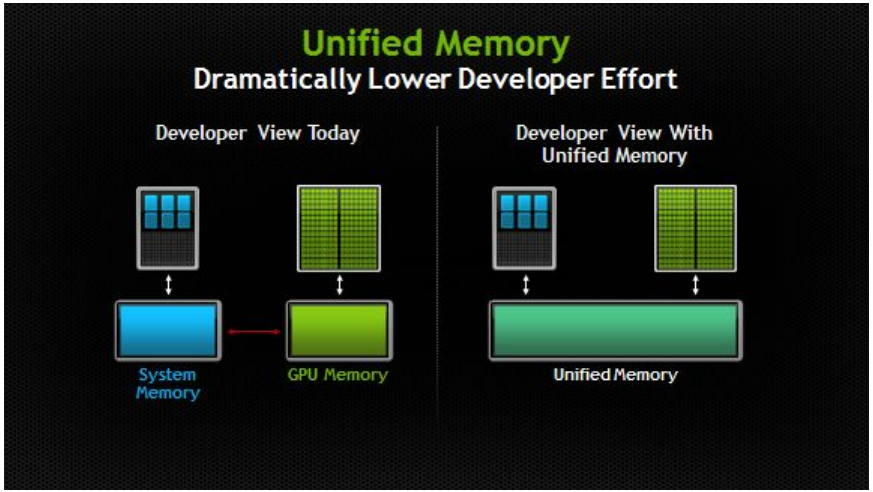
Make the data transfer from the CPU side to the GPU

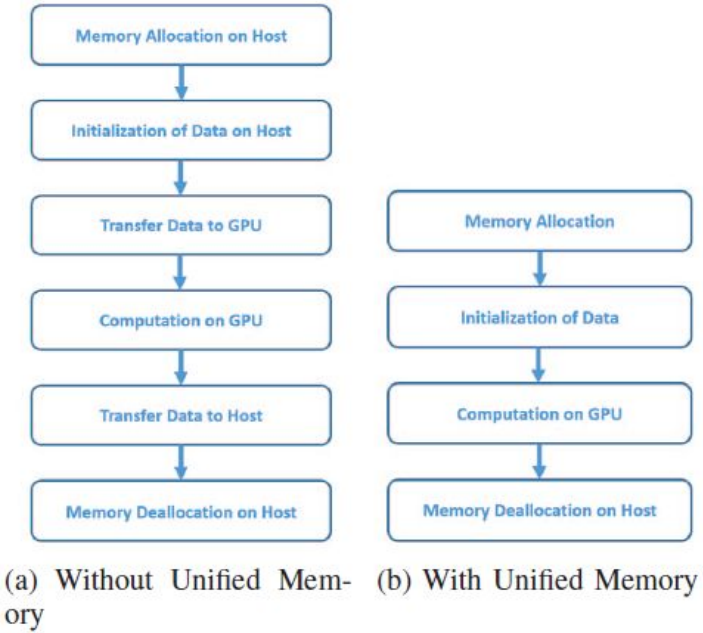
Algorithm 1 Sequential Diagonal implementation

```
Require: str1, str2, N
    Allocate the matrix H of size (N+1) × (N+1)
    Initialize the first row and the first column in H
    for slice = 1 to N*2-1 do
        if slice <N then
            z = 1;
        else
            z = slice - N + 1;
        end if
        for j = z to slice - z +1 do
            row= j;
            column=slice-j +1;
            if row == 0 OR column == 0 then
                continue;
            end if
            if str1[row-1] == str2[column-1] then
                score = 0;
            else
                score = 1;
            end if
            H[row,column]= Calculate Distance using Equation 1
        end for
    end for
```

How to make it Faster?

Unified Memory in CUDA





(a) Without Unified Memory (b) With Unified Memory

cudaMallocManaged()

Image courtesy of https://devblogs.nvidia.com/unified-memory-in-cuda-6/

Design Detail

Management part runs on CPU

It access the matrix H by following a right to left diagonal fashion, calculates the size of each diagonal, and send these information to the GPU.

Algorithm 2 Parallel Diagonal Implementation

```
Require: str1, str2, N
    Allocate the matrix H of size (N+1) × (N+1) in the unified memory
    Initialize the first row and the first column in H
    Transform H into H_T according to Equation 4
    for slice = 1 to N*2-1 do
        if slice <N then
            z =1
            sliceSize = slice;
        else
            z = slice-N+1;
            sliceSize = slice - (2 * z) + 2;
        end if
        //The following statement will create sliceSize threads on
        //the GPU. Each thread executes the code in Algorithm 3
        CUDA kernel(str1,str2,N+1,z,slice,sliceSize);
    end for
    Transform H_T to H using Equations 3 and 4
```

Design Detail

Computation part runs on GPU

The GPU runs Algorithm 3 which creates a thread for each element in the diagonal and calculates the value of the element using Equation 1 for Levenshtein algorithm.

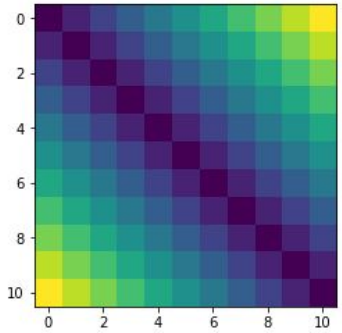
Algorithm 3 CUDA kernel

```
Require: str1, str2, N, z, slice
        Calculate thread ID
        if z == 1 then
                startIndex = slice+1;
        else
                startIndex = N * z
        end if
        index = startIndex + (ID+1) *(N-1)
        row = index / N
        column=index % N
        if row ==0 or column ==0 then
                return
        end if
        if str1[row-1] == str2[column-1] then
                score=0
        else
                score=1
        end if
        H[index] = Calculate Distance as Equation 1
```

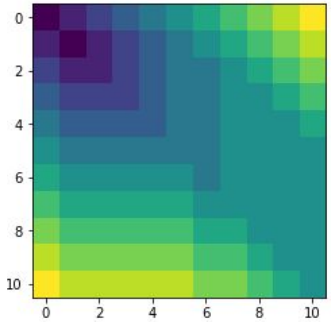
Live Demo

Results

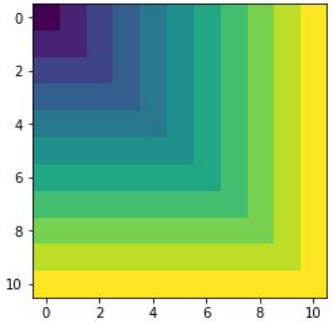
Comparison of two strings with length of 10 for each



Two identical strings



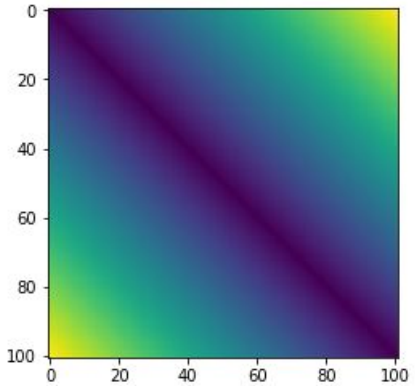
Two partially different strings



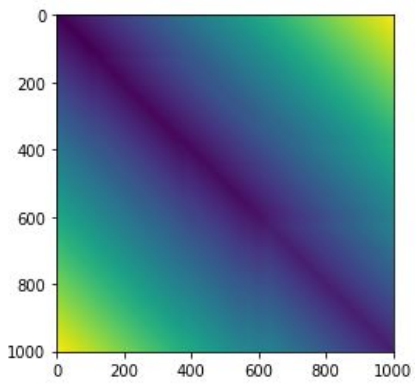
Two completely different strings

Results

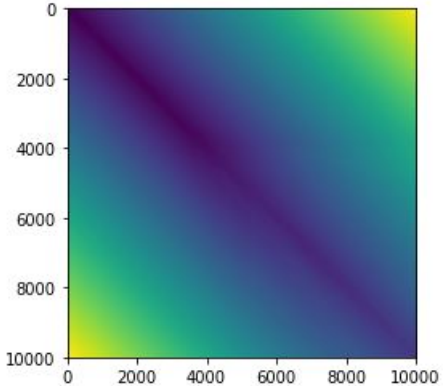
Comparison between CPU vs GPU



Score: 0
CPU time: 0.01562s
GPU time: 0.01799s
Speed up: 0.86826



Score: 85
CPU time: 1.74244s
GPU time: 0.49430s
Speed up: 3.52506



Score: 1621
CPU time: 181.41621s
GPU time: 37.29088s
Speed up: 4.86489

Type conversion

How do I specify the correct types when calling and preparing PyCUDA functions?

- (unsigned) char = numpy.(u)int8
- (unsigned) short = numpy.(u)int16
- (unsigned) int = numpy.(u)int32
- (unsigned) long = numpy.(u)int64 (only 64-bit)
- floats = numpy.float32
- double = numpy.float64
- all pointers (e.g int *, float ***, anything at all) should be numpy.intp.

Results

In length=10000 case, exclude the H and Ht matrices memory allocation and initialization time:

Pure CPU computation time: 182.38547s

Pure GPU computation time: 4.44953s

Speed up: 40.98983

Reference

1. Balhaf, K., Shehab, M.A., Wala'a, T., Al-Ayyoub, M., Al-Saleh, M. and Jararweh, Y., 2016, April. Using gpus to speed-up levenshtein edit distance computation. In Information and Communication Systems (ICICS), 2016 7th International Conference on (pp. 80-84). IEEE.

2. Balhaf, K., Alsmirat, M.A., Al-Ayyoub, M., Jararweh, Y. and Shehab, M.A., 2017, April. Accelerating Levenshtein and Damerau edit distance algorithms using GPU with unified memory. In Information and Communication Systems (ICICS), 2017 8th International Conference on (pp. 7-11). IEEE.

3. Fakirah, M., Shehab, M.A., Jararweh, Y. and Al-Ayyoub, M., 2015, November. Accelerating needleman-wunsch global alignment algorithm with gpus. In Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of (pp. 1-5). IEEE.

Q & A

Please feel free to ask any questions~

Thank you for listening!

Evaluation Method

Run each algorithm on different size of strings for 10 times and calculate the average execution time of ten runs.

$$improvement = \frac{CPU\ time}{GPU\ time}$$