Levenshtein Distance Parallelism with GPUs

Presented by Ha Hwang and Wasfi Momen

Paper Details

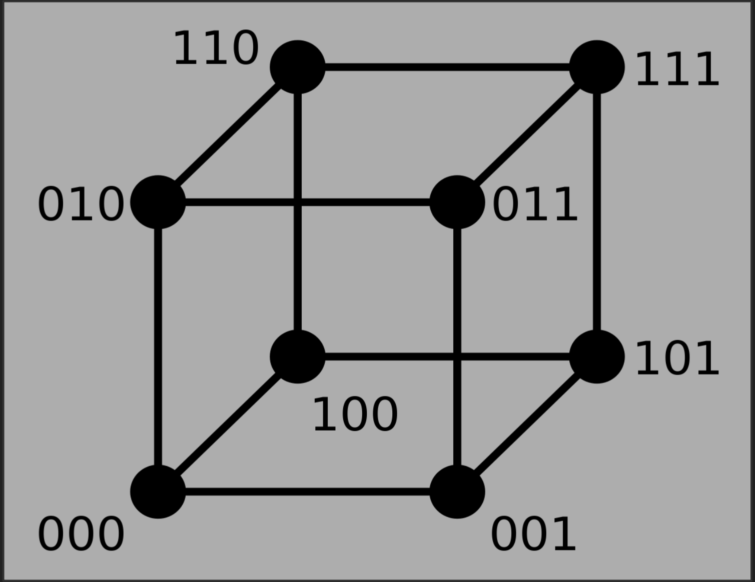
- Published in the IEEE digital, but also at the The 7th International Conference on Information and Communication Systems (ICICS) in 2016
- Authors:
  - Khaled Balhaf, Mohammed A. Shehab, Wala’a T. Al-Sarayrah, Mahmoud Al-Ayyoub, Mohammed Al-Saleh and Yaser Jararweh
- Jordan University of Science and Technology

Background and Motivation

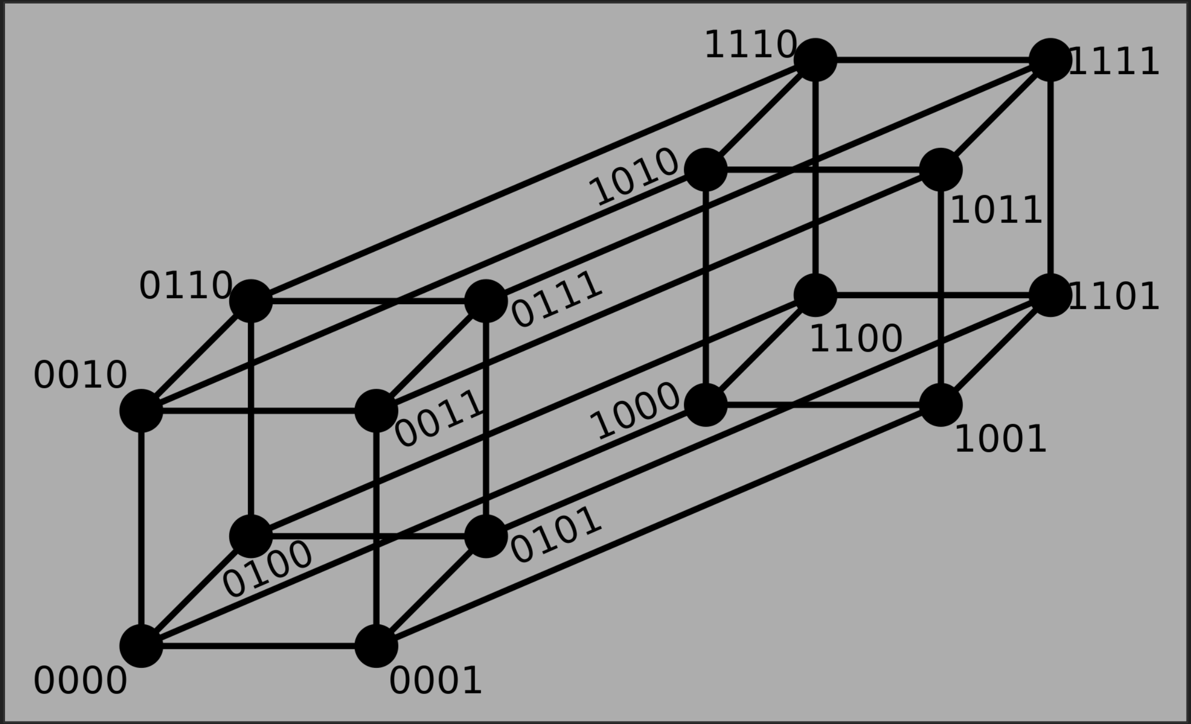
- Distance metrics give an idea of how much work we need to do to achieve a solution or how "close" we are to a solution.
    - Various types of distance metrics. EMD/Wasserstein, Acoustic, Hamming, Manhattan.
    - Can be modeled as optimization problems to iteratively reduce cost
- Wide variety of applications
    - Spelling Correction, No-Fly List Tracking, Search Engine
    - Cryptography, Privacy
    - DNA sequencing

Explanation of Problem: Hamming Distance

- Remember Hamming Distance?
- Useful for finding the distance of equivalent length strings
  - Actually may outperform Levenshtein Distance on large strings since Hamming Distance *is* the upper bound of Levenshtein Distance in the case of same length strings.
  - May outperform Gray (single-distance) codes in some situations.
- Hamming Distance is useful for finding *substitutions*.
- Can correct one-bit errors and detect (but not fix) two-bit errors.



Be Careful Who You Call Ugly In Middle School

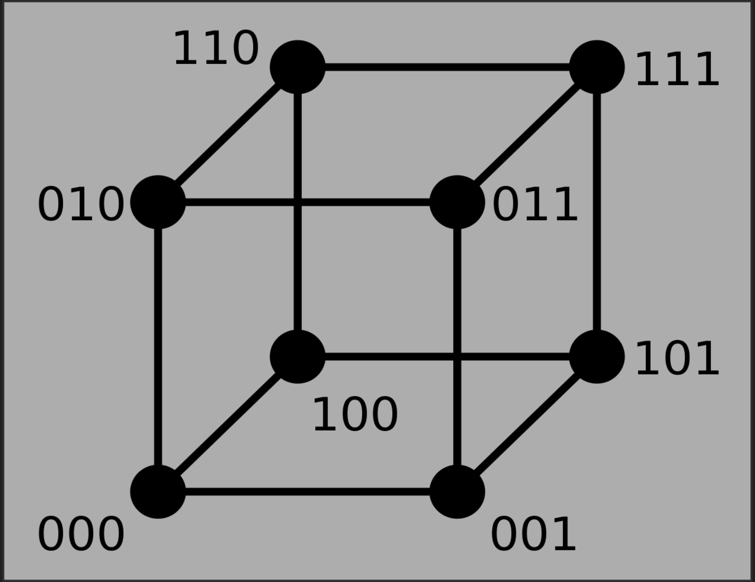


Explanation of Problem: Levenshtein Distance

- Levenshtein allows for *insertion, deletion, and substitution.*
- Levenshtein distance is better understood as the *alignment* of two strings. We try to find the minimum number of insertion, deletion, and substitution of characters in order to change string s to string t.
- Specific algorithms like Smith-Waterman or Needle-Munsch generalize Levenshtein and have properties that can be exploited to give better constraints

Explanation of Problem: Levenshtein Distance

- Which performs better Hamming or Levenshtein?
  - "01234":"12340"
  - "11234":"11324"
  - "flaw":"lawn"

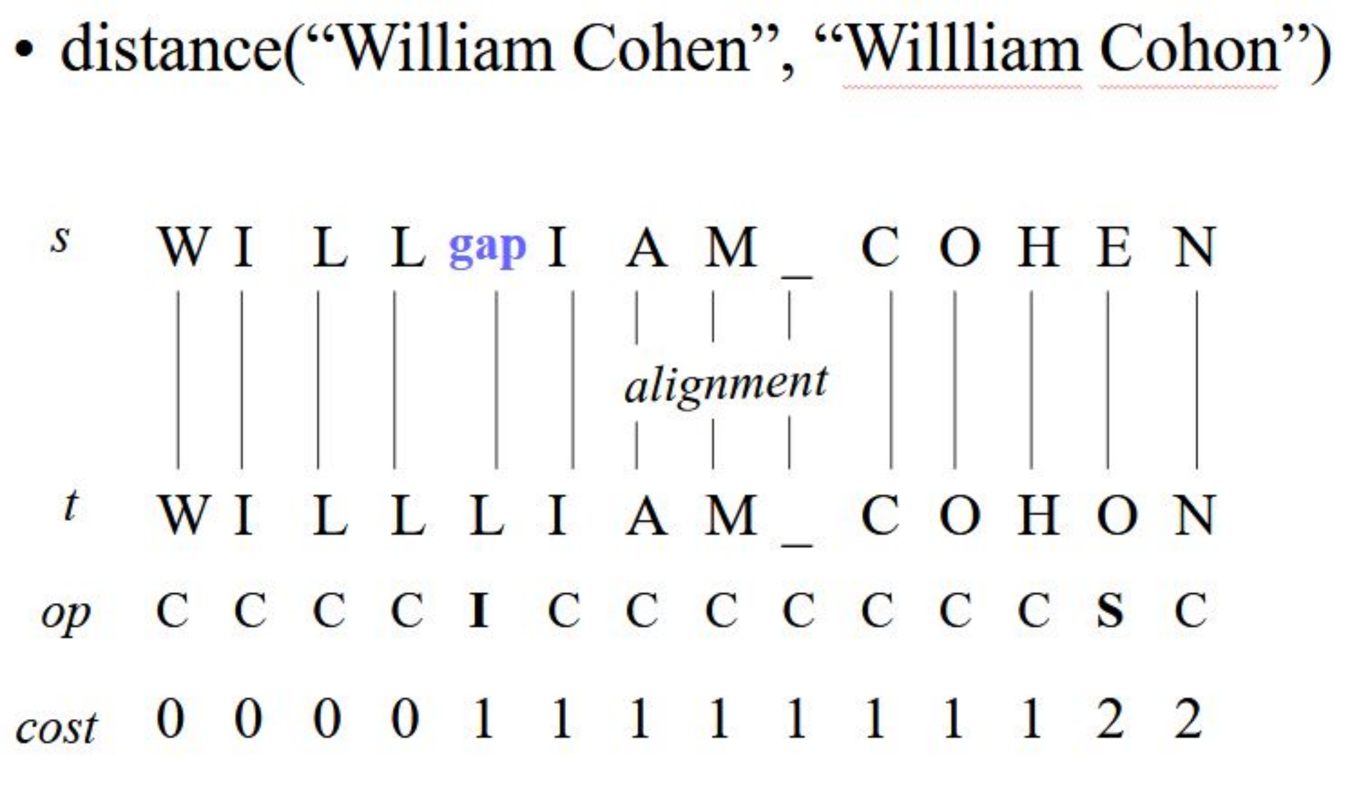


Explanation of Problem: Levenshtein Distance

- Which performs better Hamming or Levenshtein?
  - "01234":"12340"
  - "11234":"11324"
  - "flaw":"lawn"

- Answers:
  - 01234 to 12340: Hamming requires 1+1+1+1+1 = 5 substitutions, Levenshtein requires 2, insert 0 the beginning and delete 0 in end.
  - 11234 to 11324: Hamming requires 2 substitutions, Levenshtein requires 2 insertions and deletions. Represents the upper bound of Levenshtein.
  - flaw to lawn: Hamming requires substitution of all letters, distance of 4, while Levenshtein we delete f and insert n at the end, distance of 2.

Explanation of Problem: Levenshtein Distance



- distance(“William Cohen”, “Willliam Cohon”)

s	W	I	L	L	**gap**	I	A	M	_	C	O	H	E	N
t	W	I	L	L	L	I	A	M	_	C	O	H	O	N
op	C	C	C	C	**I**	C	C	C	C	C	C	C	**S**	C
cost	0	0	0	0	1	1	1	1	1	1	1	1	2	2

alignment

Explanation of Problem: Levenshtein Distance

- We can model Levenshtein in n x m matrix with one row representing string s and one column representing string t.
- Explicitly, for each cell,
    - If the string is the same, the distance is 0. Otherwise, the distance is 1. Store this value, named ***score*.**
    - Get the value of the upper cell, the left cell, and the upper left cell.
    - Find the minimum of the following:
        - (Substitution) Add *score* to the upper left cell.
        - (Insertion) Add 1 to the upper cell.
        - (Deletion) Add 1 to the left cell.
    - Store the minimum in the cell.
    - By the end of transversal of the string, cell (n, m) will contain the Levenshtein Distance.

Explanation of Problem: Levenshtein Distance

- Problem Space Guarantees:
  - One mistake (insertion, deletion, substitution) can guarantee 80% coverage of errors in English corpus ala Damerau-Levenshtein Distance.
  - Proven that Levenshtein has problem (21 Karp) and cannot improve on strings larger than n^{2-e} some e constant. Subquadratic space too large to guarantee less than subexponential time.

Initialization: Sequential Levenshtein Distance

G	T	A	G	C

A	T	A	C	C

1	0	0	1	0

	G	T	A	G	C
A					
T					
A					
C					
C					

Initialization: Sequential Levenshtein Distance

G	T	A	G	C

A	T	A	C	C

1	0	0	1	0

		G	T	A	G	C
	0	1	2	3	4	5
A	1					
T	2					
A	3					
C	4					
C	5					

Sequential Levenshtein Distance

- For each cell,
    - If the string is the same, the distance is 0. O/w the distance is 1. Store this value, named ***score.***
    - Get the value of the upper cell, the left cell, and the upper left cell.
    - Find the minimum of the following:
        - (Substitution) Add *score* to the upper left cell.
        - (Insertion) Add 1 to the upper cell.
        - (Deletion) Add 1 to the left cell.
    - Store the minimum in the cell.
    - By the end of transversal of the string, cell(n, m) will contain the Levenshtein Distance.

		G	T	A	G	C
	0	1	2	3	4	5
A	1					
T	2					
A	3					
C	4					
C	5					

Sequential Levenshtein Distance

- For each cell,
  - If the string is the same, the distance is 0. O/w the distance is 1. Store this value, named ***score*.**
  - Get the value of the upper cell, the left cell, and the upper left cell.
  - Find the minimum of the following:
    - (Substitution) Add *score* to the upper left cell.
    - (Insertion) Add 1 to the upper cell.
    - (Deletion) Add 1 to the left cell.
  - Store the minimum in the cell.
  - By the end of transversal of the string, cell(n, m) will contain the Levenshtein Distance.

		G	T	A	G	C
	0+1	1+1	2	3	4	5
A	1+1	1				
T	2					
A	3					
C	4					
C	5					

Sequential Levenshtein Distance

- For each cell,
  - If the string is the same, the distance is 0. O/w the distance is 1. Store this value, named ***score*.**
  - Get the value of the upper cell, the left cell, and the upper left cell.
  - Find the minimum of the following:
    - (Substitution) Add *score* to the upper left cell.
    - (Insertion) Add 1 to the upper cell.
    - (Deletion) Add 1 to the left cell.
  - Store the minimum in the cell.
  - By the end of transversal of the string, cell(n, m) will contain the Levenshtein Distance.

		G	T	A	G	C
	0	1+1	2+1	3	4	5
A	1	1+1	2			
T	2					
A	3					
C	4					
C	5					

Sequential Levenshtein Distance

- For each cell,
    - If the string is the same, the distance is 0. O/w the distance is 1. Store this value, named ***score*.**
    - Get the value of the upper cell, the left cell, and the upper left cell.
    - Find the minimum of the following:
        - (Substitution) Add *score* to the upper left cell.
        - (Insertion) Add 1 to the upper cell.
        - (Deletion) Add 1 to the left cell.
    - Store the minimum in the cell.
    - By the end of transversal of the string, cell(n, m) will contain the Levenshtein Distance.

		G	T	A	G	C
	0	1	2+0	3+1	4	5
A	1	1	2+1	2		
T	2					
A	3					
C	4					
C	5					

Sequential Levenshtein Distance

- For each cell,
  - If the string is the same, the distance is 0. O/w the distance is 1. Store this value, named ***score*.**
  - Get the value of the upper cell, the left cell, and the upper left cell.
  - Find the minimum of the following:
    - (Substitution) Add *score* to the upper left cell.
    - (Insertion) Add 1 to the upper cell.
    - (Deletion) Add 1 to the left cell.
  - Store the minimum in the cell.
  - By the end of transversal of the string, cell(n, m) will contain the Levenshtein Distance.

		G	T	A	G	C
	0	1	2	3+1	4+1	5
A	1	1	2	2+1	3	
T	2					
A	3					
C	4					
C	5					

Sequential Levenshtein Distance

- For each cell,
  - If the string is the same, the distance is 0. O/w the distance is 1. Store this value, named ***score*****.**
  - Get the value of the upper cell, the left cell, and the upper left cell.
  - Find the minimum of the following:
    - (Substitution) Add *score* to the upper left cell.
    - (Insertion) Add 1 to the upper cell.
    - (Deletion) Add 1 to the left cell.
  - Store the minimum in the cell.
  - By the end of transversal of the string, cell(n, m) will contain the Levenshtein Distance.

		G	T	A	G	C
	0	1	2	3	4+1	5+1
A	1	1	2	2	3+1	4
T	2					
A	3					
C	4					
C	5					

Work-Time Analysis

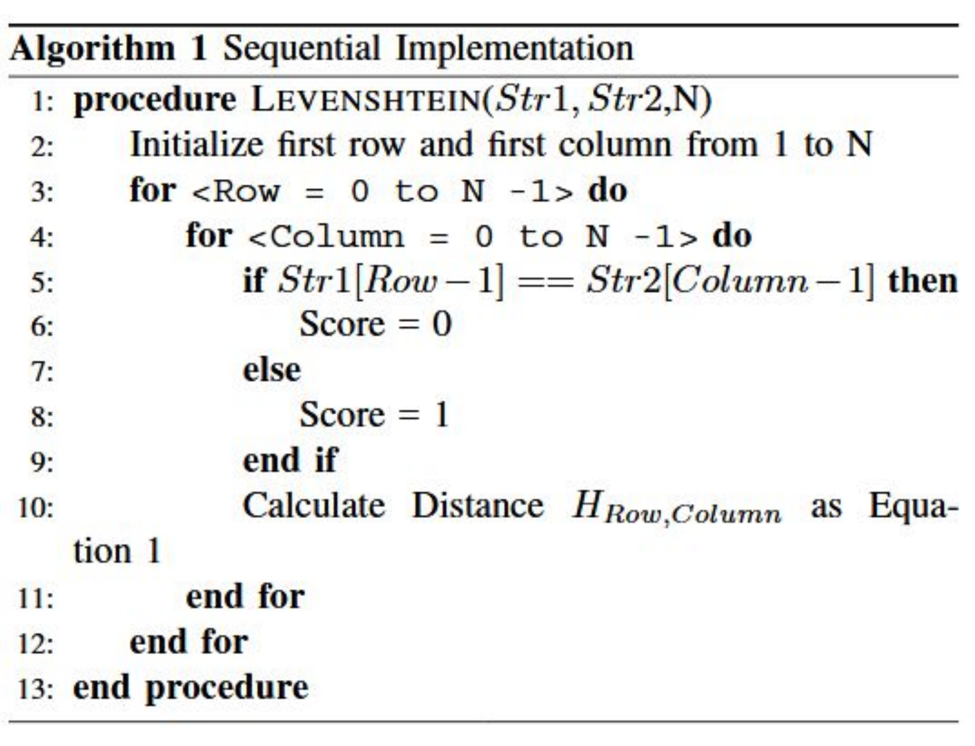
For sequential algorithm

T(n) = O(n^2) depth of calculation goes through 2D loop

W(n) = O(n^2) length of calculation also is 2D loop

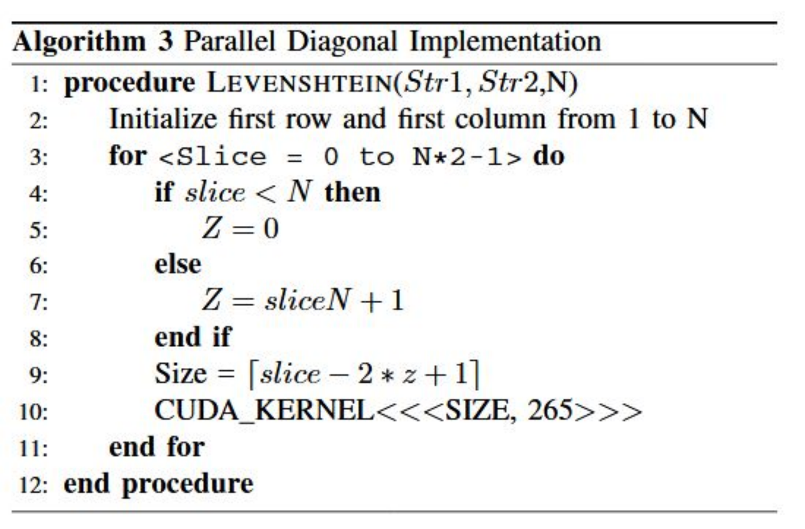
Reducing dependence of upper left, upper, left cells is the key to parallelizing Levenshtein algorithm

Sequential and Parallel Algorithms



Algorithm 1 Sequential Implementation

```
procedure LEVENSHTEIN($Str1, Str2$,N)
    Initialize first row and first column from 1 to N
    for <Row = 0 to N -1> do
        for <Column = 0 to N -1> do
            if $Str1[Row-1] == Str2[Column-1]$ then
                Score = 0
            else
                Score = 1
            end if
            Calculate Distance $H_{Row,Column}$ as Equation 1
        end for
    end for
end procedure
```

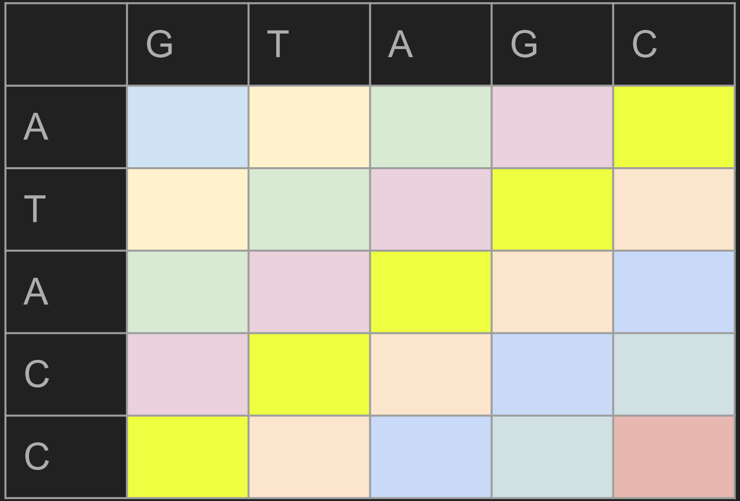


Algorithm 3 Parallel Diagonal Implementation

```
procedure LEVENSHTEIN($Str1, Str2$,N)
    Initialize first row and first column from 1 to N
    for <Slice = 0 to N*2-1> do
        if $slice < N$ then
            $Z = 0$
        else
            $Z = sliceN + 1$
        end if
        Size = $[slice - 2 * z + 1]$
        CUDA_KERNEL<<<SIZE, 265>>>
    end for
end procedure
```

Diagonal Levenshtein Distance

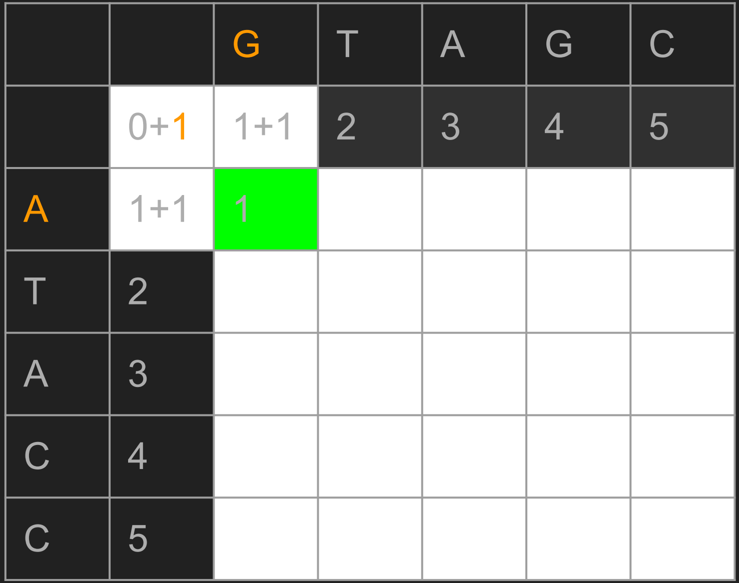
- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row



	G	T	A	G	C
A	🟦	🟨	🟩	🟪	🟡
T	🟨	🟩	🟪	🟡	🟧
A	🟩	🟪	🟡	🟧	🔷
C	🟪	🟡	🟧	🔷	⬜
C	🟡	🟧	🔷	⬜	🟥

Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 0 to 0



		G	T	A	G	C
	0+1	1+1	2	3	4	5
A	1+1	1				
T	2					
A	3					
C	4					
C	5					

Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 0 to 1

		G	T	A	G	C
	0	1+1	2	3	4	5
A	1+1	1+1	2			
T	2+1	2				
A	3					
C	4					
C	5					

Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 0 to 2

	G	T	A	G	C
A	🟦	🟨	🟩		
T	🟨	🟩			
A	🟩				
C					
C					

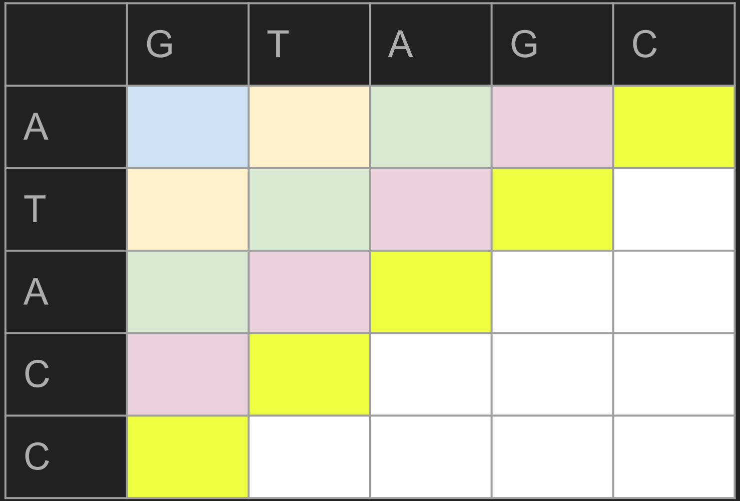
Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 0 to 3

	G	T	A	G	C
A	🟦	🟨	🟩	🟪	
T	🟨	🟩	🟪		
A	🟩	🟪			
C	🟪				
C					

Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 0 to 4



	G	T	A	G	C
A	🟦	🟫	🟩	🟪	🟨
T	🟫	🟩	🟪	🟨	
A	🟩	🟪	🟨		
C	🟪	🟨			
C	🟨				

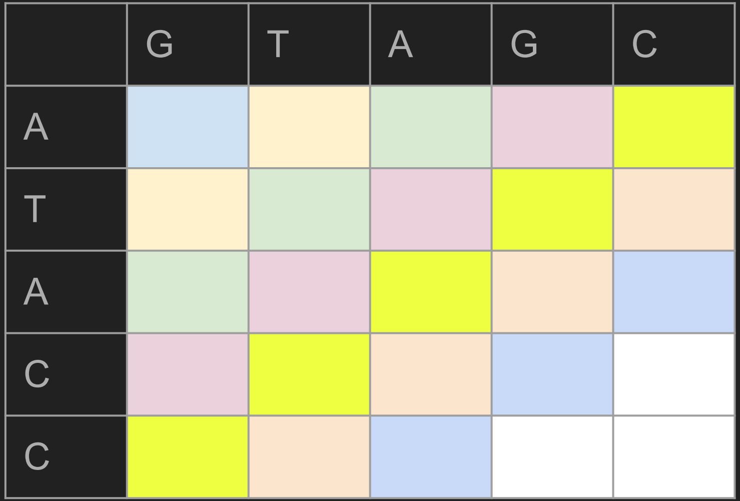
Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 1 to 5

	G	T	A	G	C
A	🟦	🟨	🟩	🟪	🟡
T	🟨	🟩	🟪	🟡	🟧
A	🟩	🟪	🟡	🟧	⬜
C	🟪	🟡	🟧	⬜	⬜
C	🟡	🟧	⬜	⬜	⬜

Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 2 to 6



	G	T	A	G	C
A					
T					
A					
C					
C					

Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 3 to 7

	G	T	A	G	C
A	🔵	🟫	🟩	🟪	🟨
T	🟫	🟩	🟪	🟨	🟧
A	🟩	🟪	🟨	🟧	🟦
C	🟪	🟨	🟧	🟦	▨
C	🟨	🟧	🟦	▨	⬜

Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- For 4 to 8

	G	T	A	G	C
A					
T					
A					
C					
C					

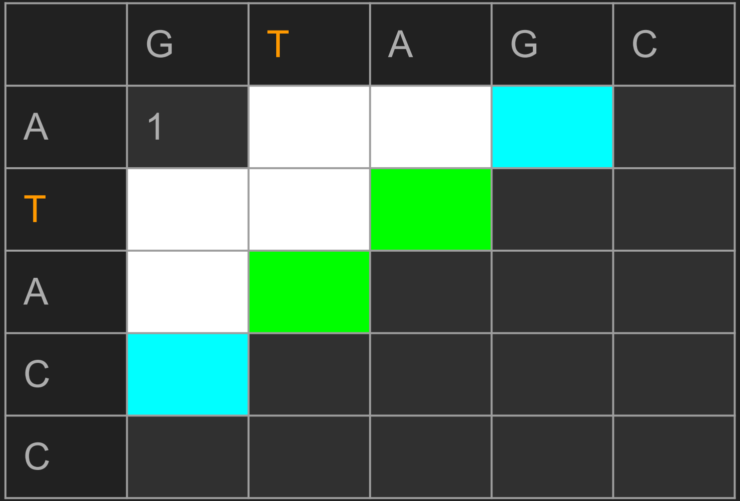
Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- Calculate minimum of upper left, upper, left cells

	G	T	A	G	C
A	⬜	⬜			
T	⬜	🟩			
A					
C					
C					

Diagonal Levenshtein Distance

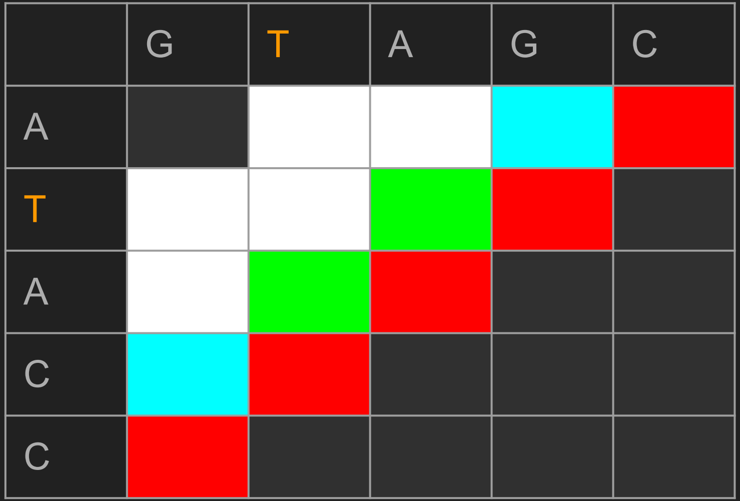
- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- Notice that the entire diagonal can be calculated
- For this step, 2 threads can run in parallel



	G	T	A	G	C
A	1	⬜	⬜	🟦	
T	⬜	⬜	🟩		
A	⬜	🟩			
C	🟦				
C					

Diagonal Levenshtein Distance

- Next diagonal depends on previous diagonal
- max(row, col) * 2 -1 gives total number of diagonal slices
- Upper left half has value in 0th row
- Bottom right half has value in +1th row
- Notice that the entire diagonal can be calculated
- For this step, 2 threads can run in parallel
- And next step, 3 threads



	G	T	A	G	C
A		⬜	⬜	🟦	🟥
T	⬜	⬜	🟩	🟥	
A	⬜	🟩	🟥		
C	🟦	🟥			
C	🟥				

Work-Time Analysis (Parallel)

For sequential algorithm

T(n) is between O(n) and O(n^2) depth of calculation is not halved, but depth is reduced

W(n) is between O(n) and O(n^2)

It is weakly optimal algorithm

Results

- Number of threads : 256
- Based on GPU utilization test

- Due to hardware limitation, 2D array converted into 1D arrays

- GPU converts index back to 2D

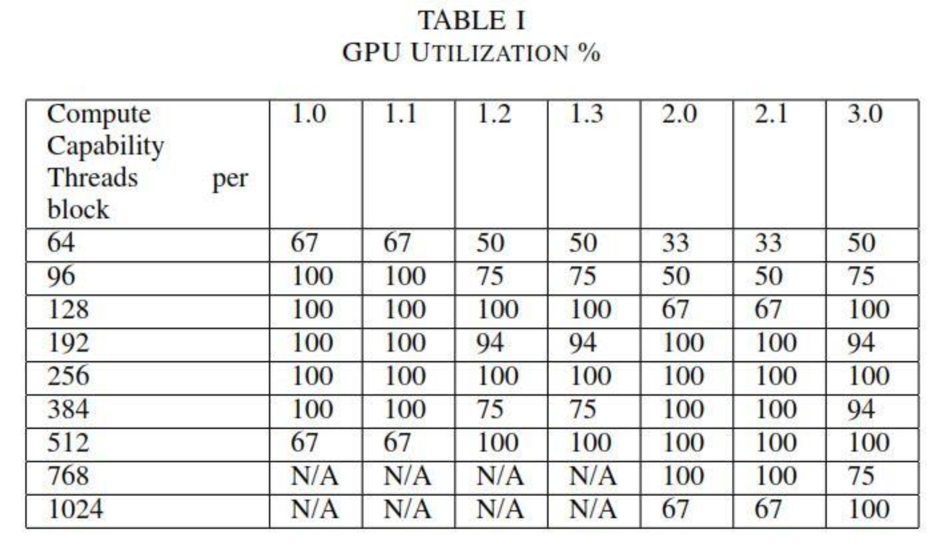


TABLE I
GPU UTILIZATION %

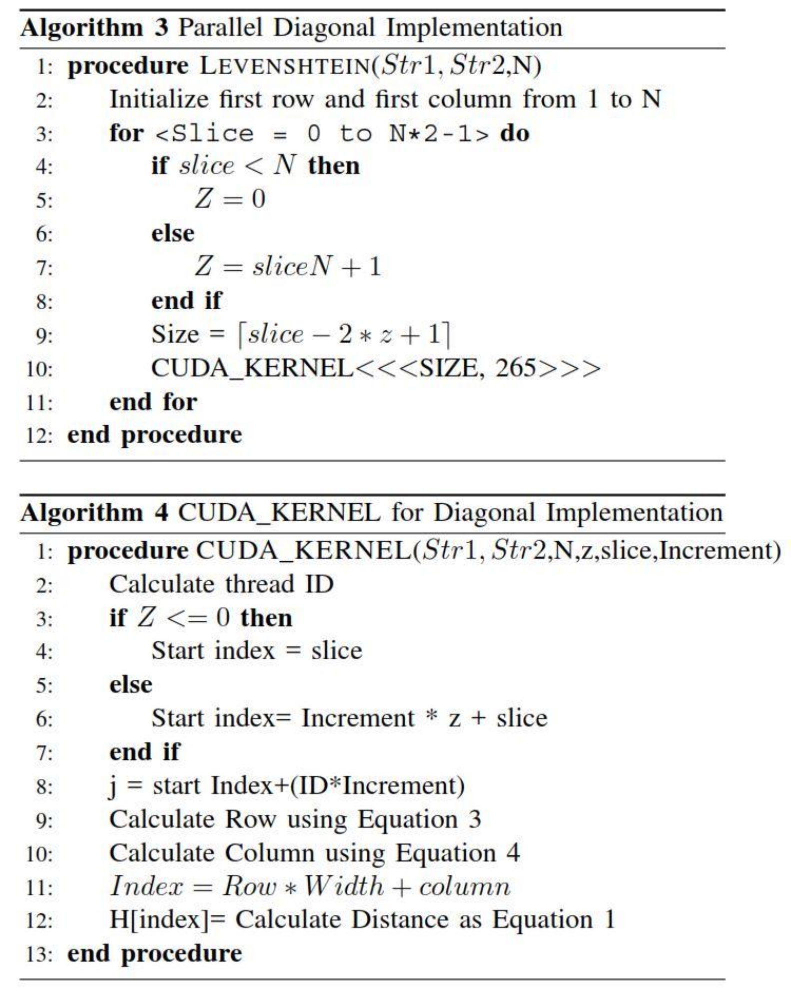
Compute Capability Threads per block	1.0	1.1	1.2	1.3	2.0	2.1	3.0
64	67	67	50	50	33	33	50
96	100	100	75	75	50	50	75
128	100	100	100	100	67	67	100
192	100	100	94	94	100	100	94
256	100	100	100	100	100	100	100
384	100	100	75	75	100	100	94
512	67	67	100	100	100	100	100
768	N/A	N/A	N/A	N/A	100	100	75
1024	N/A	N/A	N/A	N/A	67	67	100

Results

- Number of threads : 256
- Based on GPU utilization test
- GT 740M, Kepler
- Depends on CPU used

Due to hardware limitation, 2D array converted into 1D arrays

Index



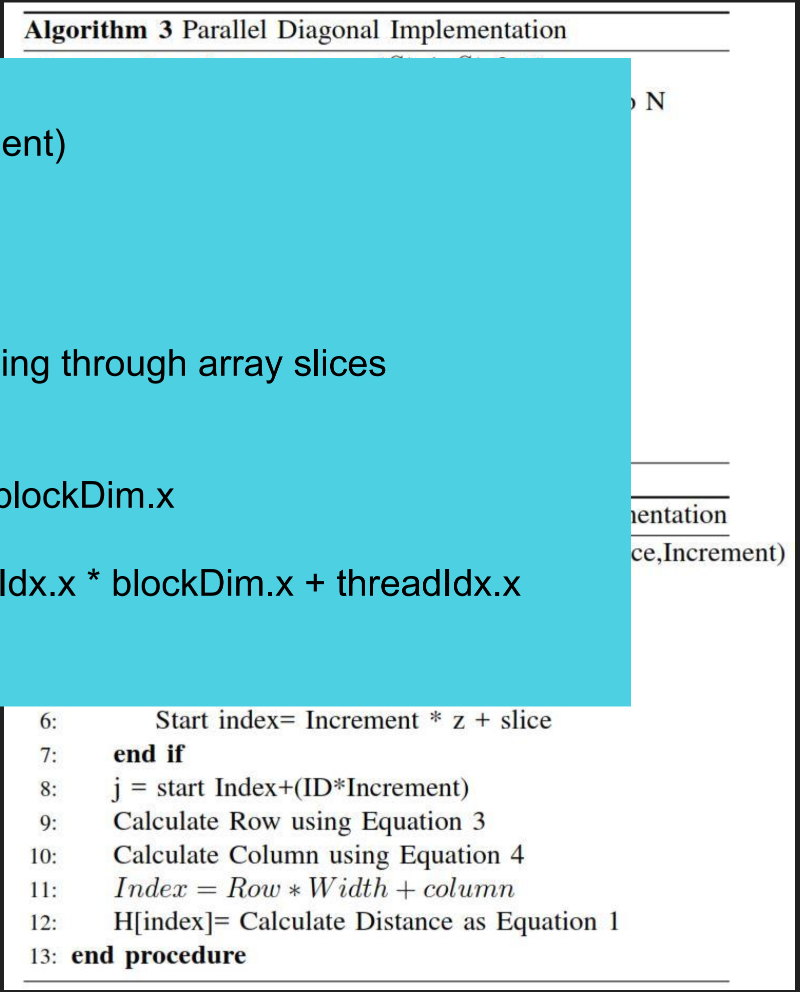
Algorithm 3 Parallel Diagonal Implementation

```
procedure LEVENSHTEIN($Str1, Str2$,N)
    Initialize first row and first column from 1 to N
    for <Slice = 0 to N*2-1> do
        if $slice < N$ then
            $Z = 0$
        else
            $Z = sliceN + 1$
        end if
        Size = $\lceil slice - 2 * z + 1 \rceil$
        CUDA_KERNEL<<<SIZE, 265>>>
    end for
end procedure
```

Algorithm 4 CUDA_KERNEL for Diagonal Implementation

```
procedure CUDA_KERNEL($Str1, Str2$,N,z,slice,Increment)
    Calculate thread ID
    if $Z <= 0$ then
        Start index = slice
    else
        Start index= Increment * z + slice
    end if
    j = start Index+(ID*Increment)
    Calculate Row using Equation 3
    Calculate Column using Equation 4
    $Index = Row * Width + column$
    H[index]= Calculate Distance as Equation 1
end procedure
```

```
// Filled equations
procedure CUDA_KERNEL(Str1, Str2, N, z, slice, Increment)
        Calculate threadID
        if Z <= 0 then
                start_index = slice  //started from beginning
        else
                start_index = Increment * z + slice  //progressing through array slices
        end if
        j = start_index + (threadID * Increment)
        Row = start_index / Width                     //Width is blockDim.x
        Column = Index % Width
        Index = Row * Width + Column                  // i = blockIdx.x * blockDim.x + threadIdx.x
        H[Index] = take the min of the 3 cells (Levenshtein)
end procedure
```



Algorithm 3 Parallel Diagonal Implementation

N

ientation

ce,Increment)

```
            Start index= Increment * z + slice
        end if
        j = start Index+(ID*Increment)
        Calculate Row using Equation 3
        Calculate Column using Equation 4
        $Index = Row * Width + column$
        H[index]= Calculate Distance as Equation 1
    end procedure
```

Conclusion and Future Work

CPU implementation (sequential) of Levenshtein algorithm takes longer time as input sequence size increases

GPU implementation (parallel) shows 11x speed improvement when input sequence size reaches about 6000

Diagonal Levenshtein algorithm solves some dependence between matrix cells

It would have been useful to include power consumption, heat generation, and noise comparing CPU (sequential) and GPU (parallel). Also, since a laptop was used, perhaps thermal limit caused system throttle down

Also, modern CPU with octa-cores or more can be tested in both sequential and parallel modes. It was not clear if authors used only one CPU thread or multiple threads.

Questions?