PARALLEL LEVENSHTEIN DISTANCE

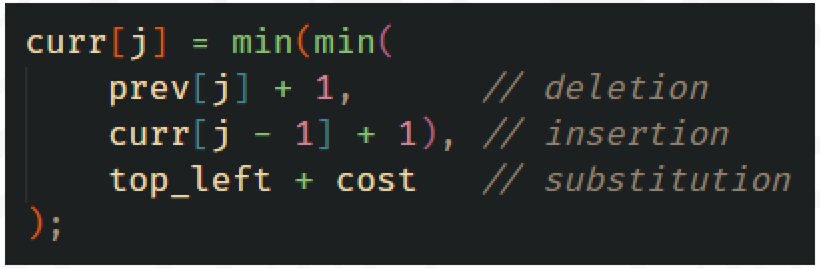
Gianna Bossoreale

CSE633: Parallel Algorithms

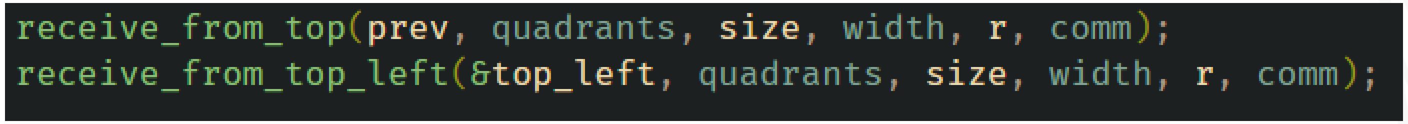
University at Buffalo The State University of New York

Levenshtein Distance Implementation

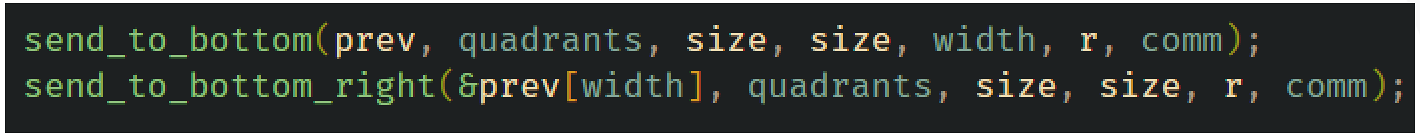
- Every element of the levenshtein distance matrix is calculated from a previously calculated distance.
- The only information that is needed is the top, top left, and left element for our calculation.
- This is what is communicated between processors



```
curr[j] = min(min(
    prev[j] + 1,        // deletion
    curr[j - 1] + 1),  // insertion
    top_left + cost    // substitution
);
```



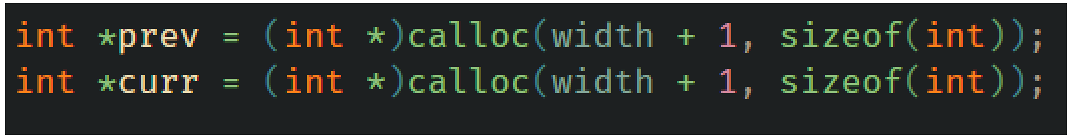
```
receive_from_top(prev, quadrants, size, width, r, comm);
receive_from_top_left(&top_left, quadrants, size, width, r, comm);
```



```
send_to_bottom(prev, quadrants, size, size, width, r, comm);
send_to_bottom_right(&prev[width], quadrants, size, size, r, comm);
```

"Matrix" vs Matrix

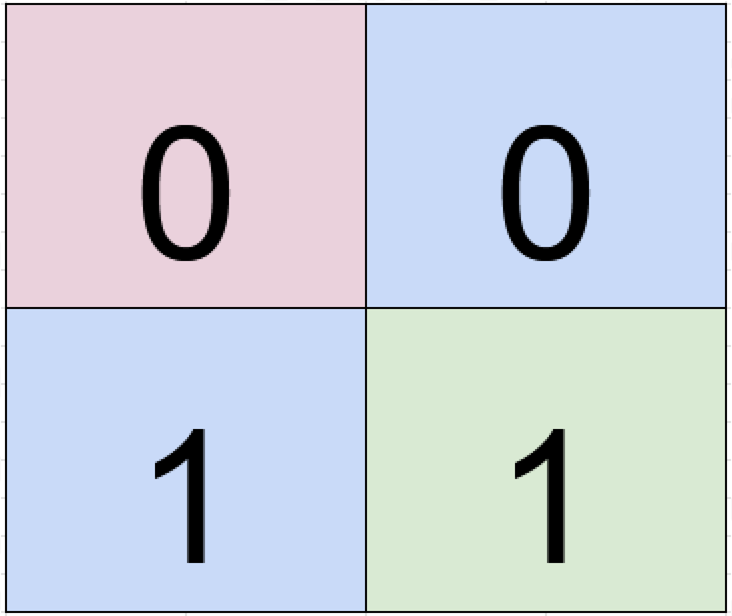
- Rather than allocating an entire matrix, we only need two rows at a time
- For example, row 1 depends on row 0
- Thus, we only have two arrays (a prev and a curr) and just swap them after every row is calculated



```
int *prev = (int *)calloc(width + 1, sizeof(int));
int *curr = (int *)calloc(width + 1, sizeof(int));
```

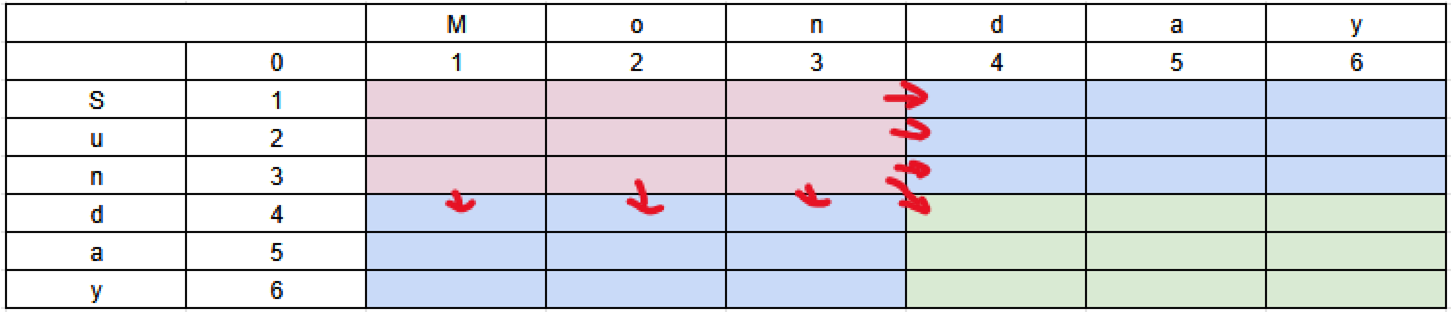
Assigning Processors

- The "matrix" is split up between the processors
- For example:
  - Processor 0 calculates the edit distance then sends its results to the bottom and bottom right processors
- Each processor has its own portion of the input strings so that we can split the work up equally between the processors
- As said before, each processor needs three things:
  - Top, top left, left values
  - The bottom right quadrant would thus need the top left value from the pink processor and both blue processors
- Information spreads through the matrix across the diagonals



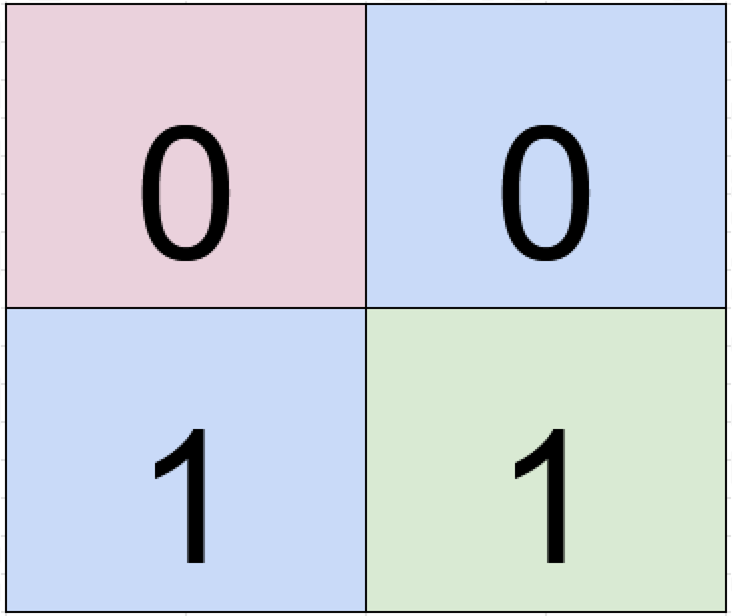
0	0
1	1

Example



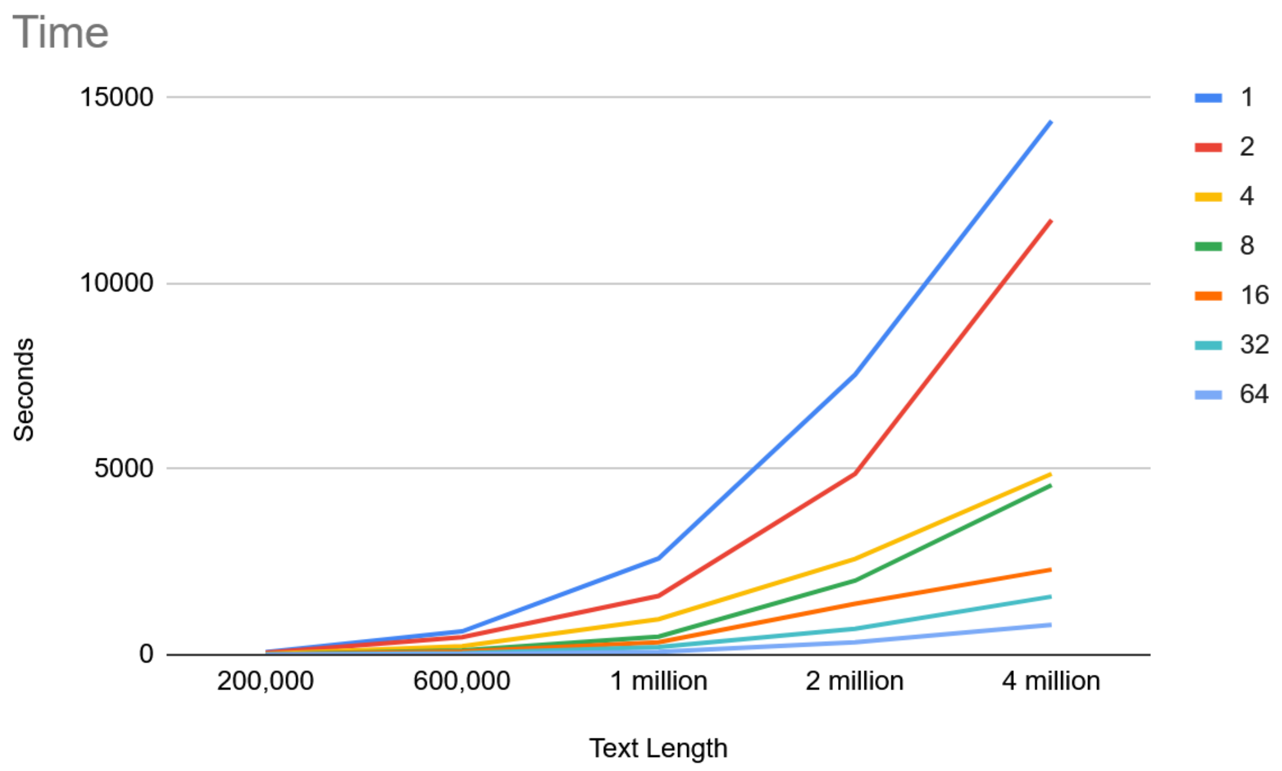
		M	o	n	d	a	y
	0	1	2	3	4	5	6
S	1	🟥	🟥	🟥	🟦	🟦	🟦
u	2	🟥	🟥	🟥	🟦	🟦	🟦
n	3	🟥	🟥	🟥	🟦	🟦	🟦
d	4	🟦	🟦	🟦	🟩	🟩	🟩
a	5	🟦	🟦	🟦	🟩	🟩	🟩
y	6	🟦	🟦	🟦	🟩	🟩	🟩

- Processor 0 calculates its edit distance, then sends the values to the bottom and bottom right processors
- It doesn't need to send to the right because it'll be the same processor as the current one
- Thus, in order for green to calculate its edit distance, it needs to wait for the blue to calculate the distance first
- We find the final edit distance at last processor at the last column of the curr array

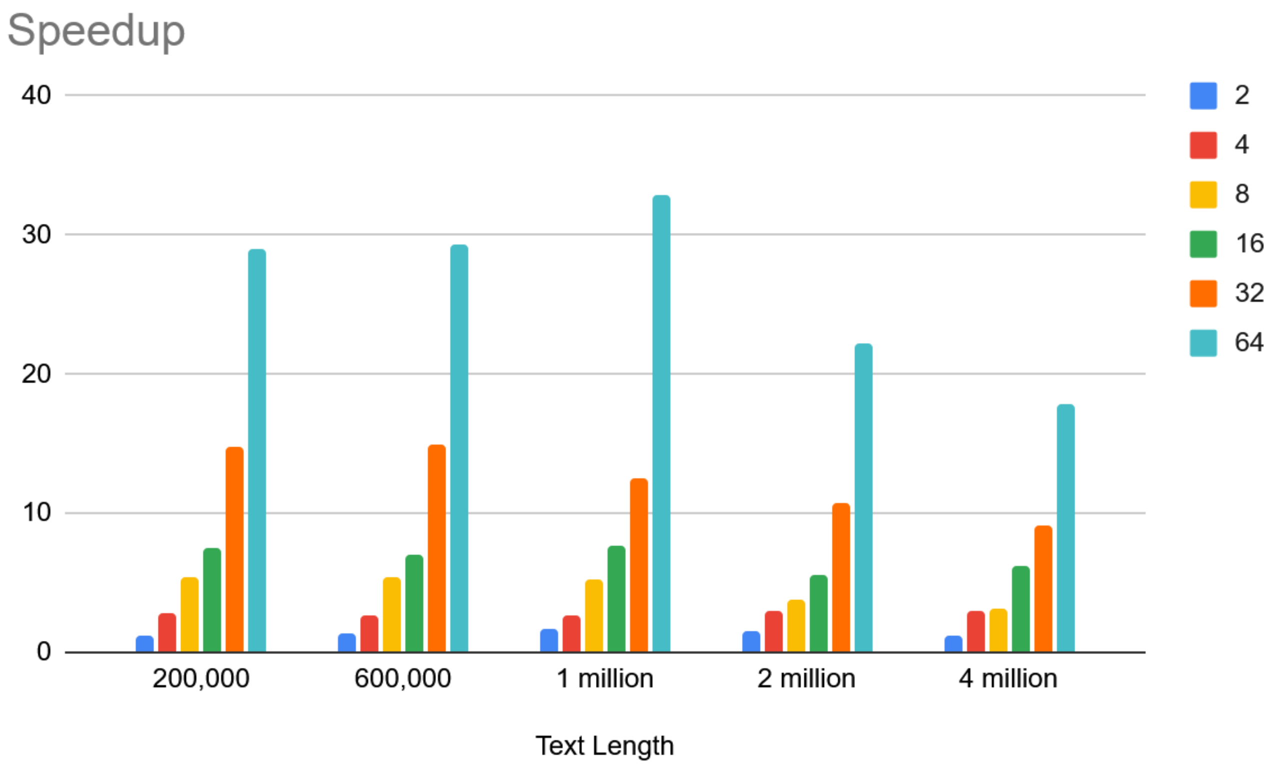


0	0
1	1

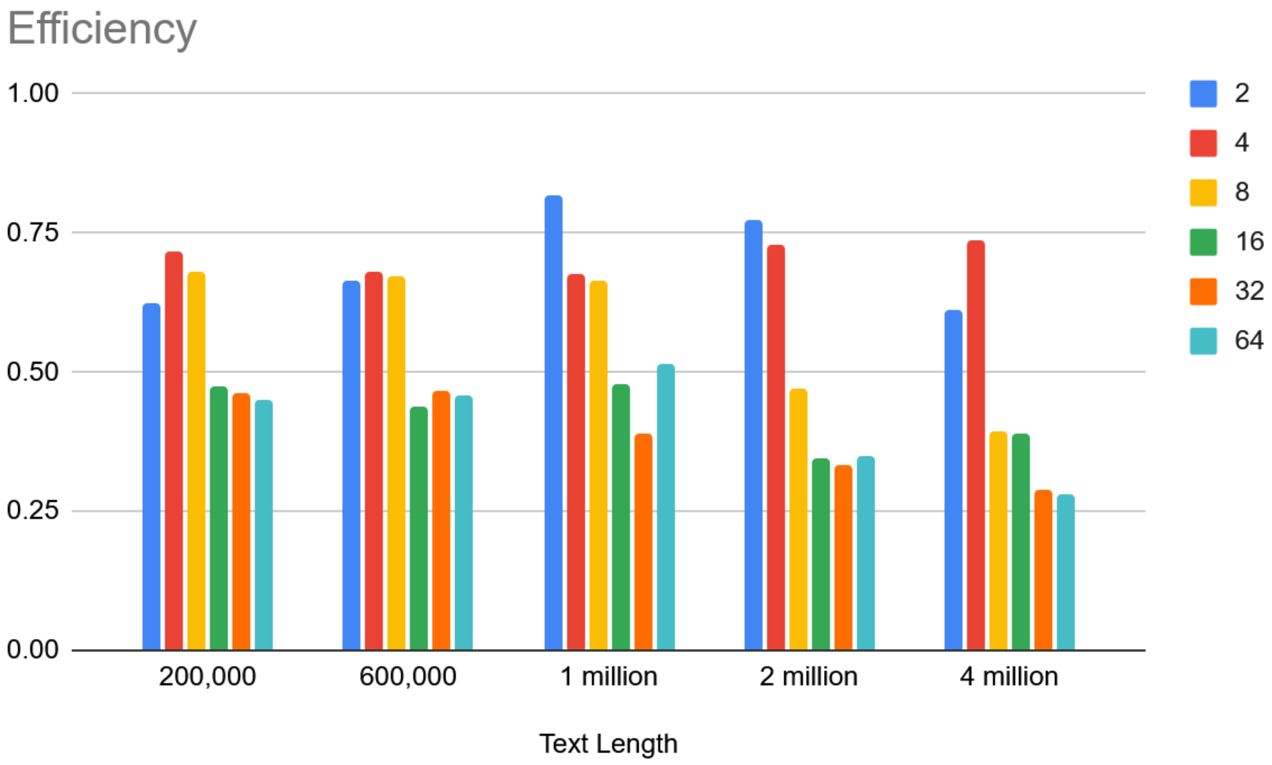
Benchmarking: Time



Benchmarking: Speedup



Benchmarking: Efficiency



Questions?