Hashtag Post Counting

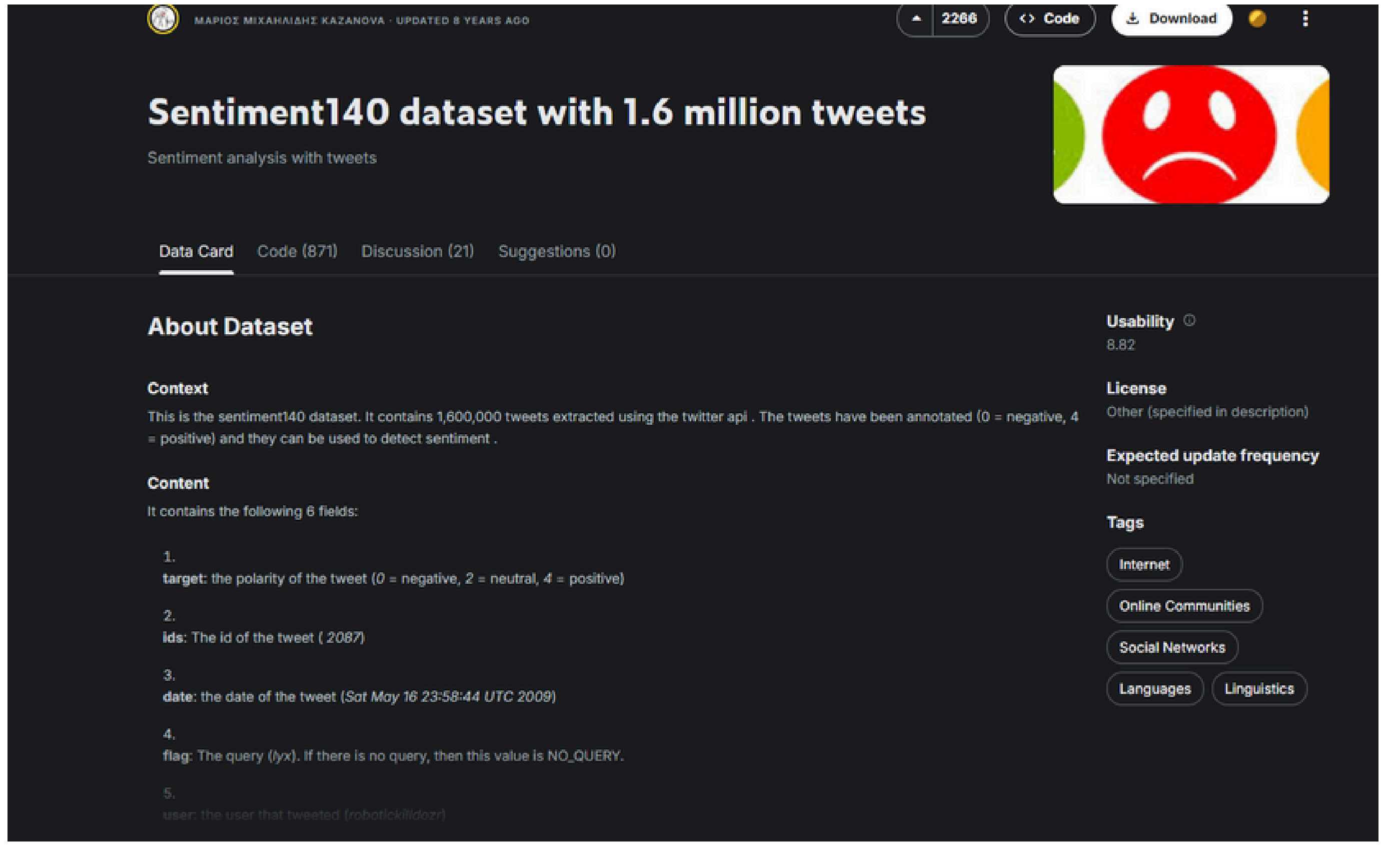
01418343 Parallel Computing with CUDA

6610402141 Patiput Ukham

About the project

Parallel Hashtag Counter using CUDA - เปรียบเทียบ
ประสิทธิภาพระหว่าง Sequential และ GPU Parallel Processing

Dataset



Sentiment140 dataset with 1.6 million tweets

Sentiment analysis with tweets

About Dataset

Context

This is the sentiment140 dataset. It contains 1,600,000 tweets extracted using the twitter api . The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment .

Content

It contains the following 6 fields:

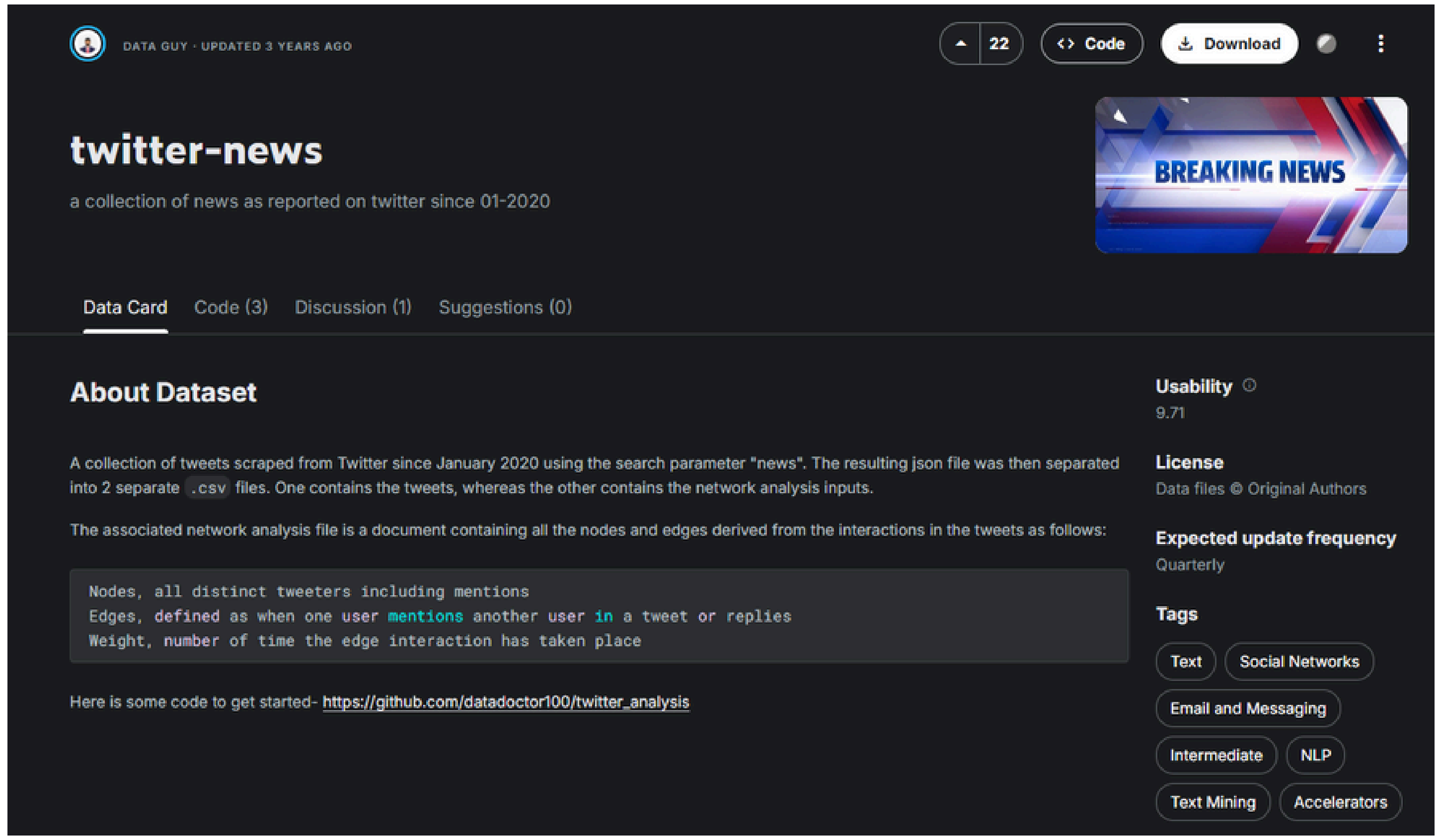
1. **target:** the polarity of the tweet (*0* = negative, *2* = neutral, *4* = positive)
2. **ids**: The id of the tweet (*2087*)
3. **date**: the date of the tweet (*Sat May 16 23:58:44 UTC 2009*)
4. **flag**: The query (*lyx*). If there is no query, then this value is NO_QUERY.
5. **user**: the user that tweeted (*robotickilldozr*)

Usability
8.82

License
Other (specified in description)

Expected update frequency
Not specified

Tags
Internet
Online Communities
Social Networks
Languages
Linguistics



twitter-news

a collection of news as reported on twitter since 01-2020

About Dataset

A collection of tweets scraped from Twitter since January 2020 using the search parameter "news". The resulting json file was then separated into 2 separate `.csv` files. One contains the tweets, whereas the other contains the network analysis inputs.

The associated network analysis file is a document containing all the nodes and edges derived from the interactions in the tweets as follows:

```
Nodes, all distinct tweeters including mentions
Edges, defined as when one user mentions another user in a tweet or replies
Weight, number of time the edge interaction has taken place
```

Here is some code to get started- https://github.com/datadoctor100/twitter_analysis

Usability
9.71

License
Data files © Original Authors

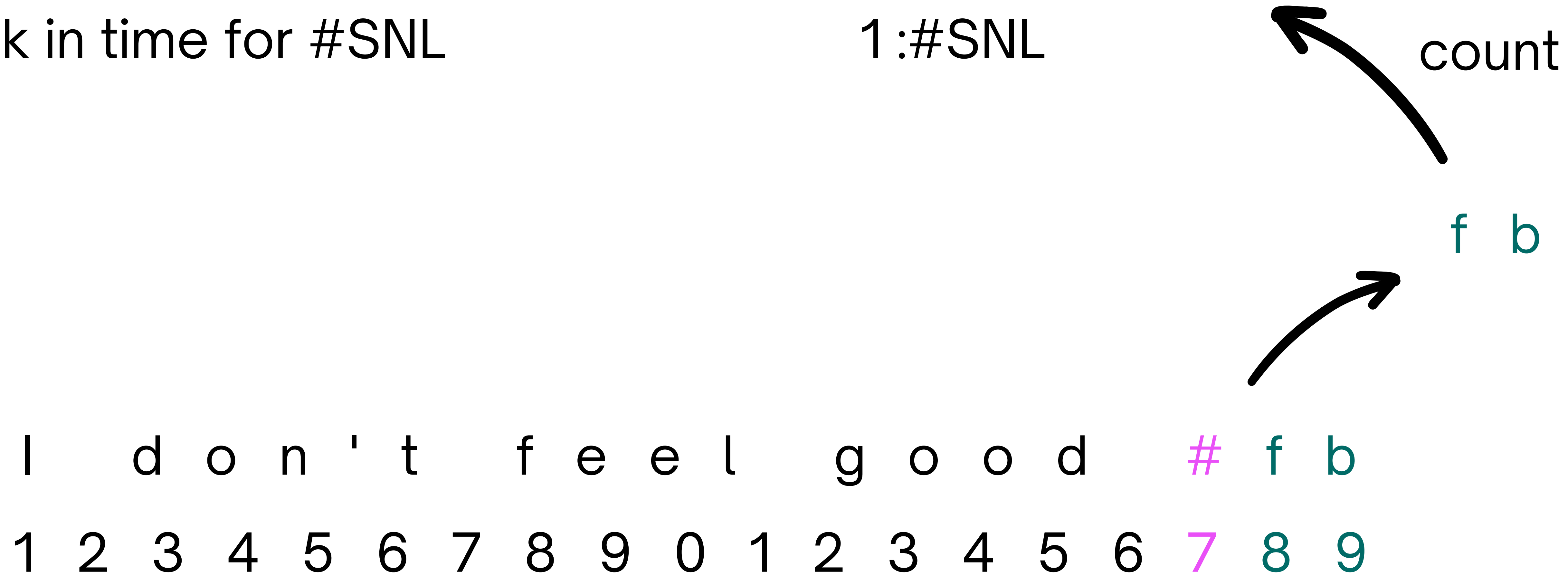
Expected update frequency
Quarterly

Tags
Text
Social Networks
Email and Messaging
Intermediate
NLP
Text Mining
Accelerators

Idea

I don't feel good #fb
Sad end to the game #Canucks
I can get back in time for #SNL

1 :#fb
1 :#Canucks
1 :#SNL



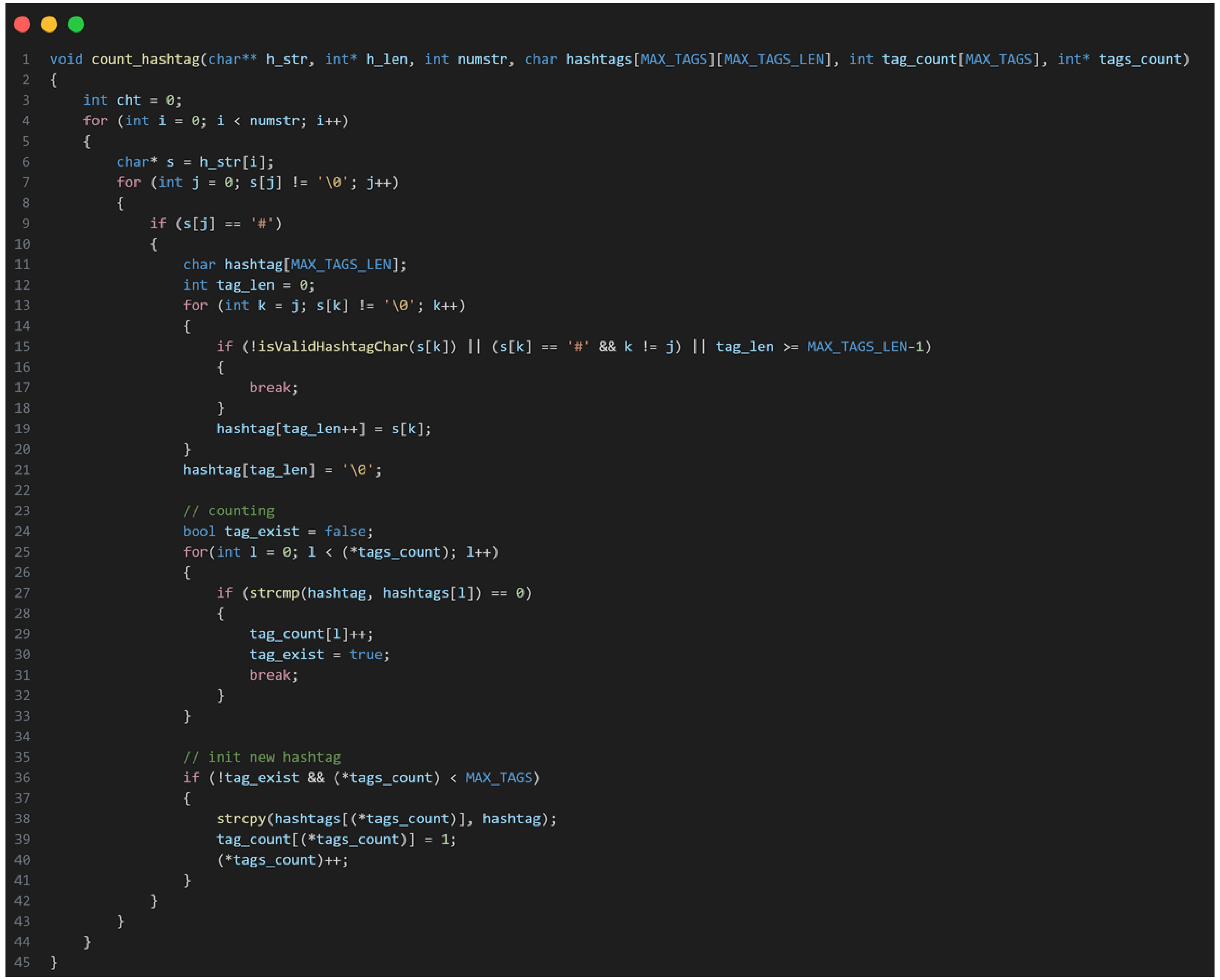
count

f b

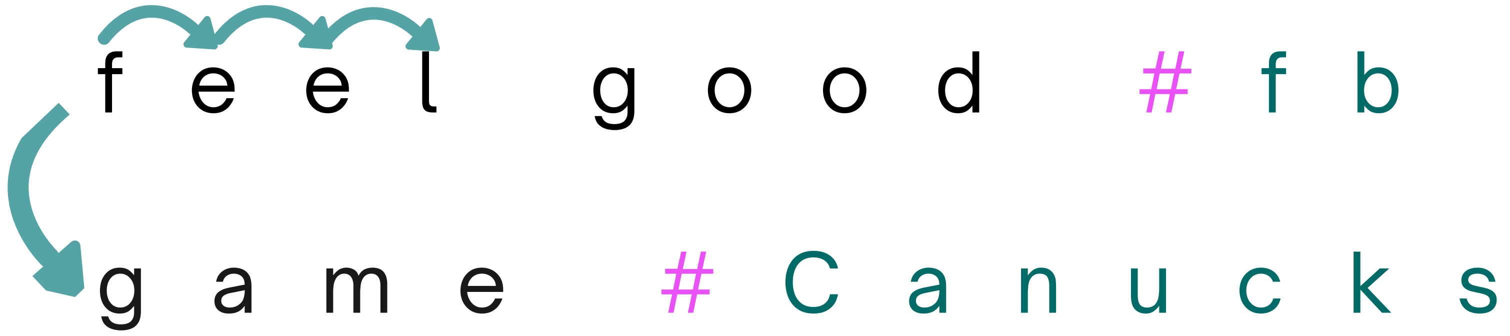
I don't feel good #fb

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

Sequential Solving



```c
void count_hashtag(char** h_str, int* h_len, int numstr, char hashtags[MAX_TAGS][MAX_TAGS_LEN], int tag_count[MAX_TAGS], int* tags_count)
{
    int cht = 0;
    for (int i = 0; i < numstr; i++)
    {
        char* s = h_str[i];
        for (int j = 0; s[j] != '\0'; j++)
        {
            if (s[j] == '#')
            {
                char hashtag[MAX_TAGS_LEN];
                int tag_len = 0;
                for (int k = j; s[k] != '\0'; k++)
                {
                    if (!isValidHashtagChar(s[k]) || (s[k] == '#' && k != j) || tag_len >= MAX_TAGS_LEN-1)
                    {
                        break;
                    }
                    hashtag[tag_len++] = s[k];
                }
                hashtag[tag_len] = '\0';

                // counting
                bool tag_exist = false;
                for(int l = 0; l < (*tags_count); l++)
                {
                    if (strcmp(hashtag, hashtags[l]) == 0)
                    {
                        tag_count[l]++;
                        tag_exist = true;
                        break;
                    }
                }

                // init new hashtag
                if (!tag_exist && (*tags_count) < MAX_TAGS)
                {
                    strcpy(hashtags[(*tags_count)], hashtag);
                    tag_count[(*tags_count)] = 1;
                    (*tags_count)++;
                }
            }
        }
    }
}
```



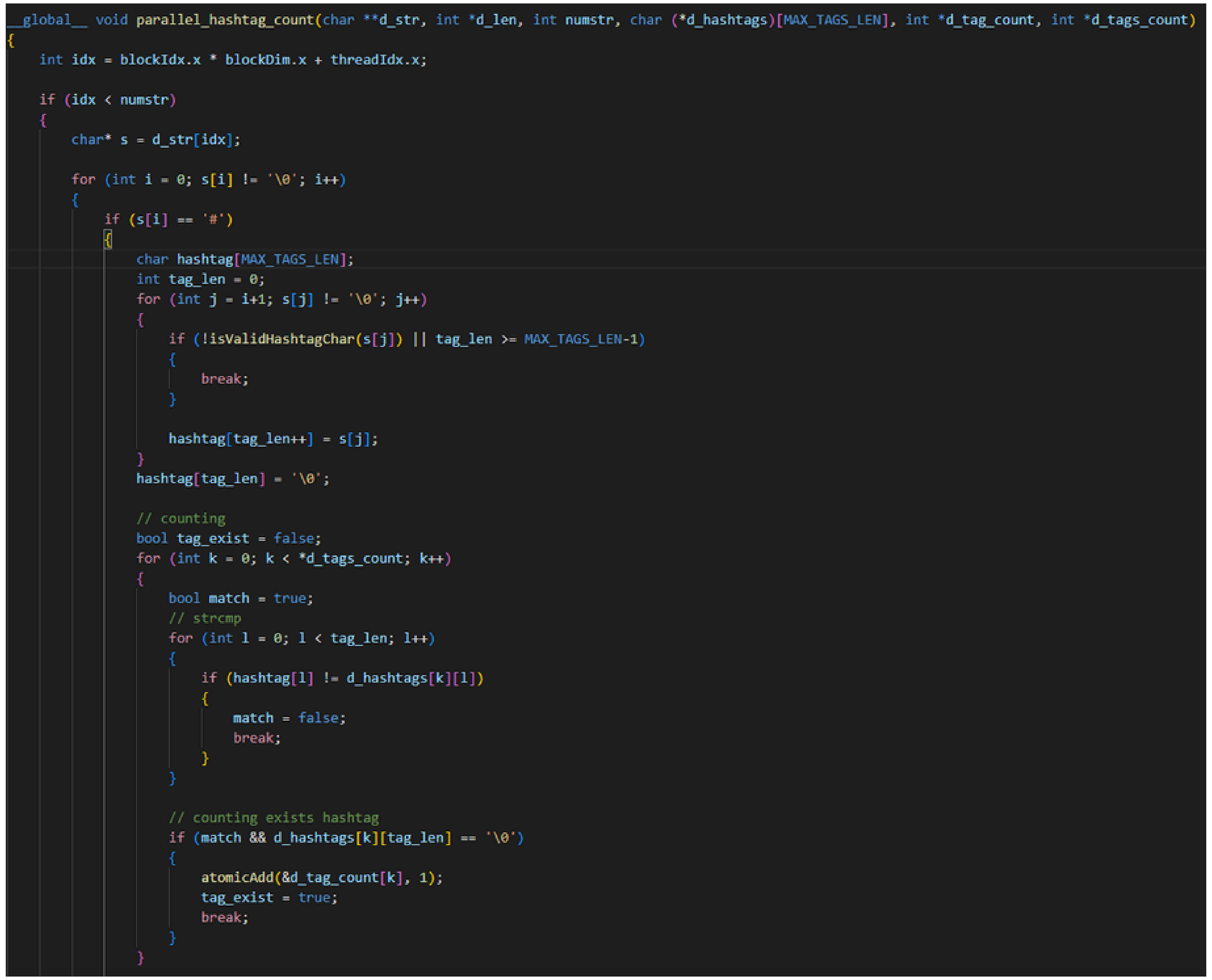
feel good #fb

game #Canucks

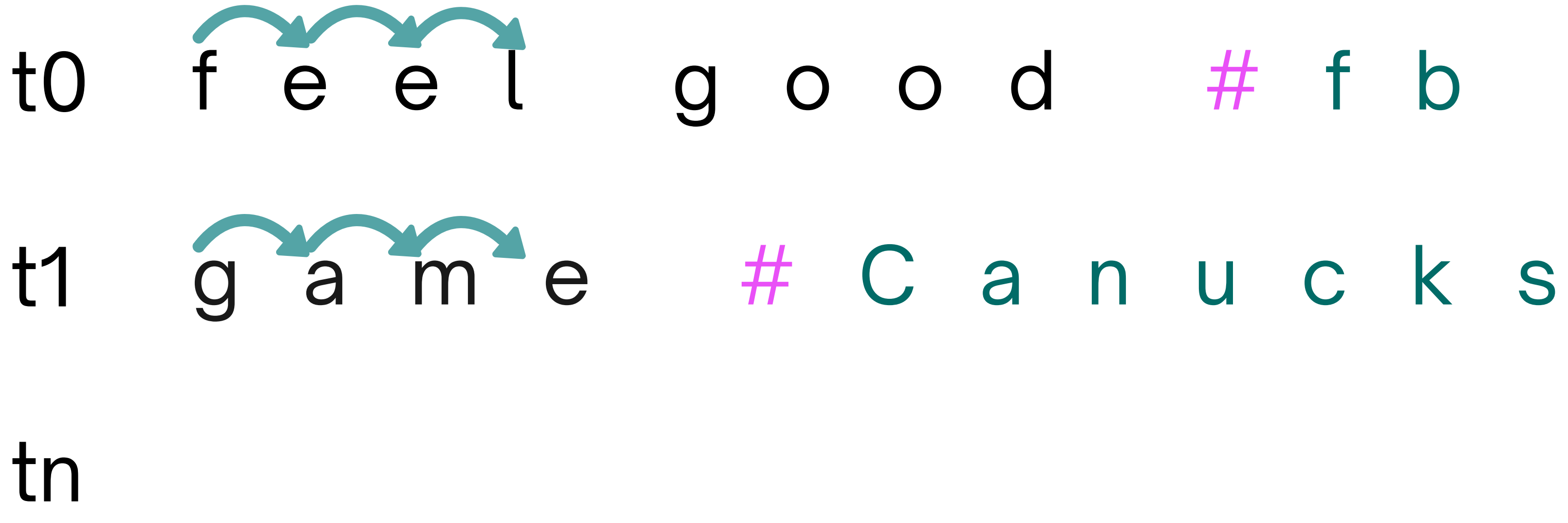
Time Complexity: O(n × m × k × t)

- n = จำนวนของข้อความ
- m = ความยาวเฉลี่ยของแต่ละบรรทัด
- k = จำนวน hashtag เฉลี่ยต่อบรรทัด
- t = จำนวน unique hashtags ที่เจอแล้ว

Parallel optimize



```
__global__ void parallel_hashtag_count(char **d_str, int *d_len, int numstr, char (*d_hashtags)[MAX_TAGS_LEN], int *d_tag_count, int *d_tags_count)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    if (idx < numstr)
    {
        char* s = d_str[idx];

        for (int i = 0; s[i] != '\0'; i++)
        {
            if (s[i] == '#')
            {
                char hashtag[MAX_TAGS_LEN];
                int tag_len = 0;
                for (int j = i+1; s[j] != '\0'; j++)
                {
                    if (!isValidHashtagChar(s[j]) || tag_len >= MAX_TAGS_LEN-1)
                    {
                        break;
                    }

                    hashtag[tag_len++] = s[j];
                }
                hashtag[tag_len] = '\0';

                // counting
                bool tag_exist = false;
                for (int k = 0; k < *d_tags_count; k++)
                {
                    bool match = true;
                    // strcmp
                    for (int l = 0; l < tag_len; l++)
                    {
                        if (hashtag[l] != d_hashtags[k][l])
                        {
                            match = false;
                            break;
                        }
                    }

                    // counting exists hashtag
                    if (match && d_hashtags[k][tag_len] == '\0')
                    {
                        atomicAdd(&d_tag_count[k], 1);
                        tag_exist = true;
                        break;
                    }
                }
```



t0 feel good #fb

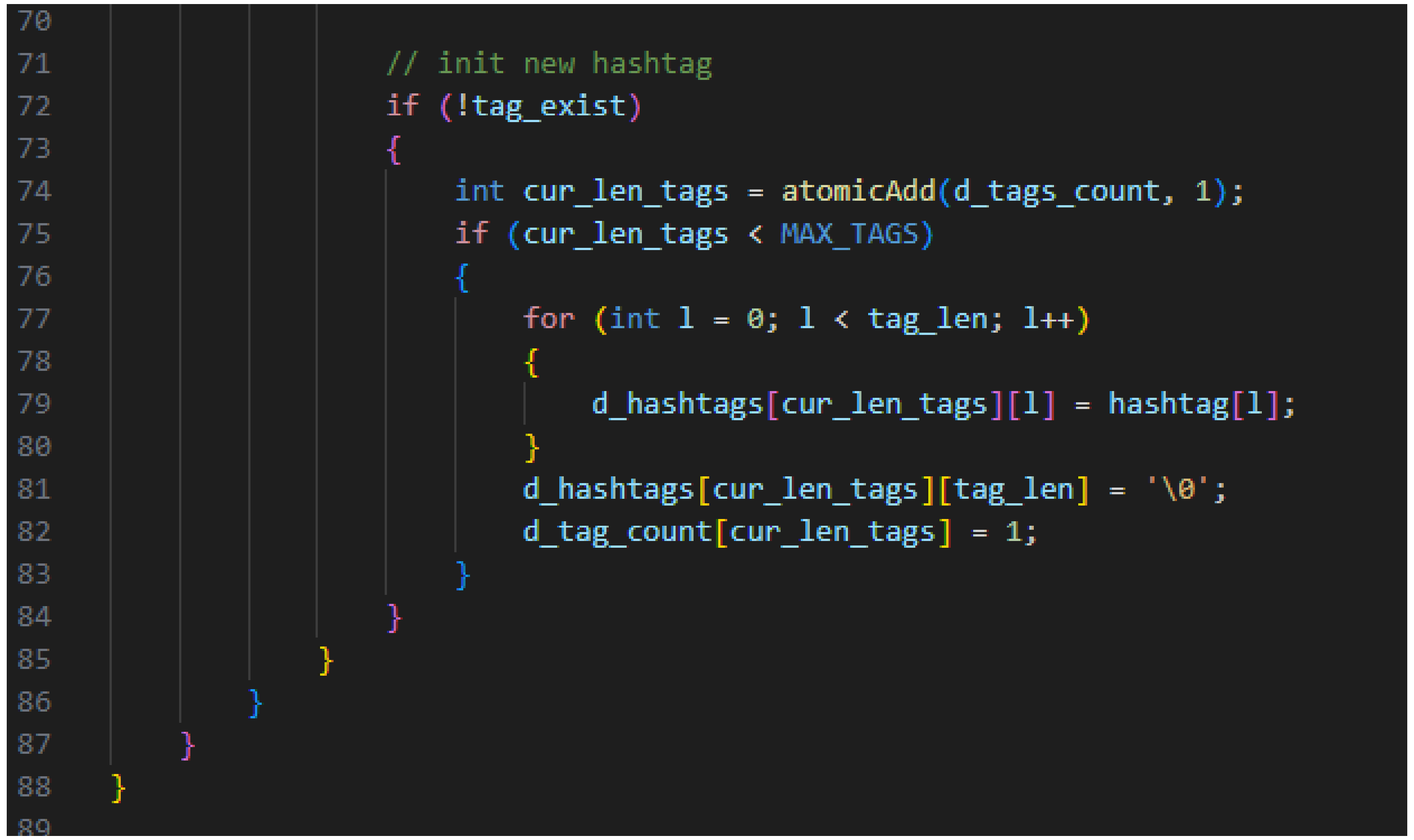
t1 game #Canucks

tn

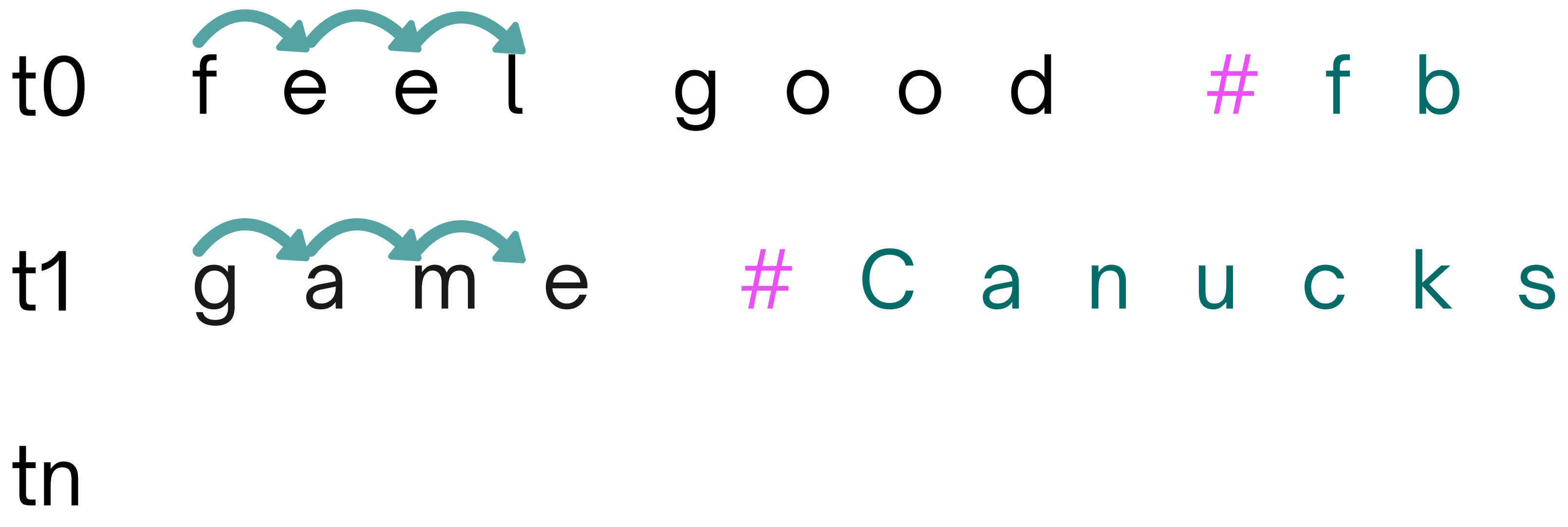
Time Complexity: O(m × k × t / P)

- m = ความยาวบรรทัด
- k = จำนวน hashtags ต่อบรรทัด
- t = จำนวน unique hashtags (linear search)
- P = จำนวน threads ที่ active พร้อมกัน

Parallel optimize



```
                    // init new hashtag
                    if (!tag_exist)
                    {
                        int cur_len_tags = atomicAdd(d_tags_count, 1);
                        if (cur_len_tags < MAX_TAGS)
                        {
                            for (int l = 0; l < tag_len; l++)
                            {
                                d_hashtags[cur_len_tags][l] = hashtag[l];
                            }
                            d_hashtags[cur_len_tags][tag_len] = '\0';
                            d_tag_count[cur_len_tags] = 1;
                        }
                    }
                }
            }
        }
    }
```



t0 feel good #fb

t1 game #Canucks

tn

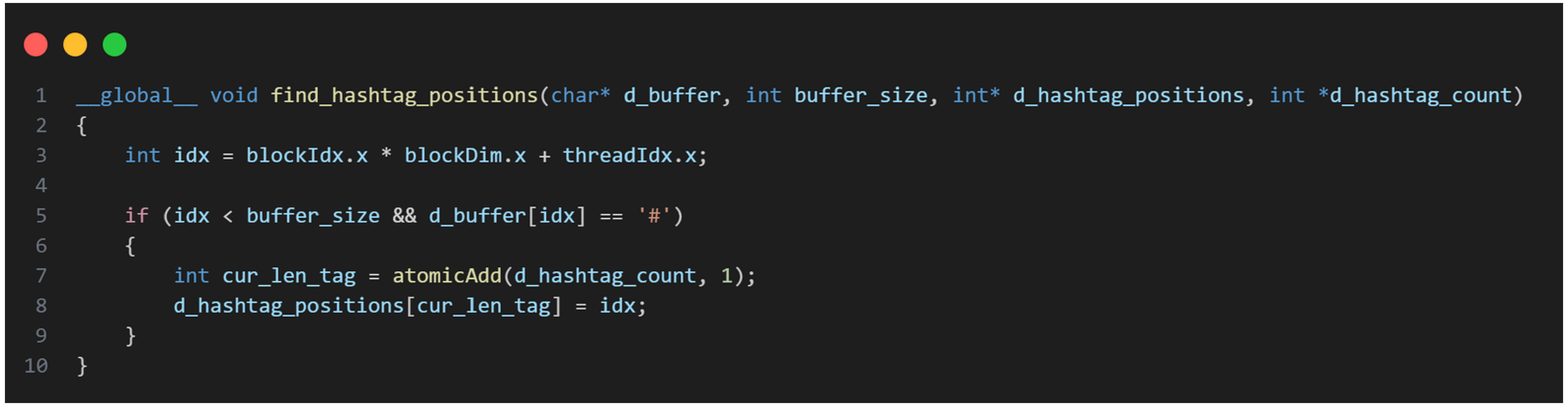
Time Complexity: O(m × k × t / P)

- m = ความยาวบรรทัด
- k = จำนวน hashtags ต่อบรรทัด
- t = จำนวน unique hashtags (linear search)
- P = จำนวน threads ที่ active พร้อมกัน

Parallel optimize reduce

phase 1

Time Complexity: O(m x n / P)



```
__global__ void find_hashtag_positions(char* d_buffer, int buffer_size, int* d_hashtag_positions, int *d_hashtag_count)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    if (idx < buffer_size && d_buffer[idx] == '#')
    {
        int cur_len_tag = atomicAdd(d_hashtag_count, 1);
        d_hashtag_positions[cur_len_tag] = idx;
    }
}
```

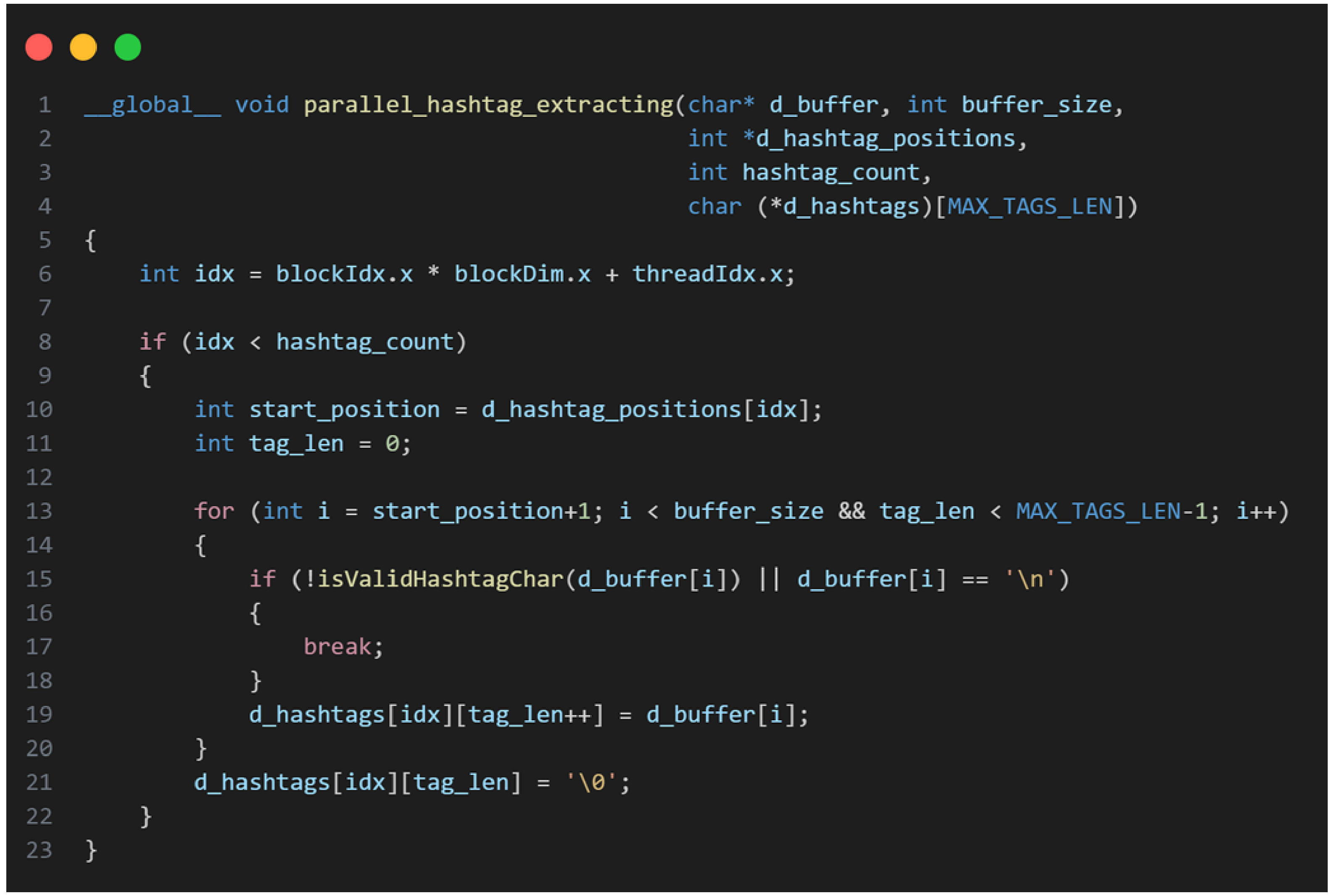
t6 -> checks '#' -> atomicAdd -> position[0] = 6

t20 -> checks '#' -> atomicAdd -> position[1] = 20

t40 -> checks '#' -> atomicAdd -> position[2] = 40

Parallel optimize reduce

phase 2



```
__global__ void parallel_hashtag_extracting(char* d_buffer, int buffer_size,
                                            int *d_hashtag_positions,
                                            int hashtag_count,
                                            char (*d_hashtags)[MAX_TAGS_LEN])
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    if (idx < hashtag_count)
    {
        int start_position = d_hashtag_positions[idx];
        int tag_len = 0;

        for (int i = start_position+1; i < buffer_size && tag_len < MAX_TAGS_LEN-1; i++)
        {
            if (!isValidHashtagChar(d_buffer[i]) || d_buffer[i] == '\n')
            {
                break;
            }
            d_hashtags[idx][tag_len++] = d_buffer[i];
        }
        d_hashtags[idx][tag_len] = '\0';
    }
}
```

Time Complexity: O(L)

- L = avg hashtag length

Thread 0 → position[0] = 6
-> Extract from index 7: "world"
-> d_hashtags[0] = "world"

Parallel optimize reduce

phase 3

Time Complexity: O(k × t × L / P)

- k = total hashtags
- t = unique hashtags
- L = avg hashtag length

Thread 0 -> "world"

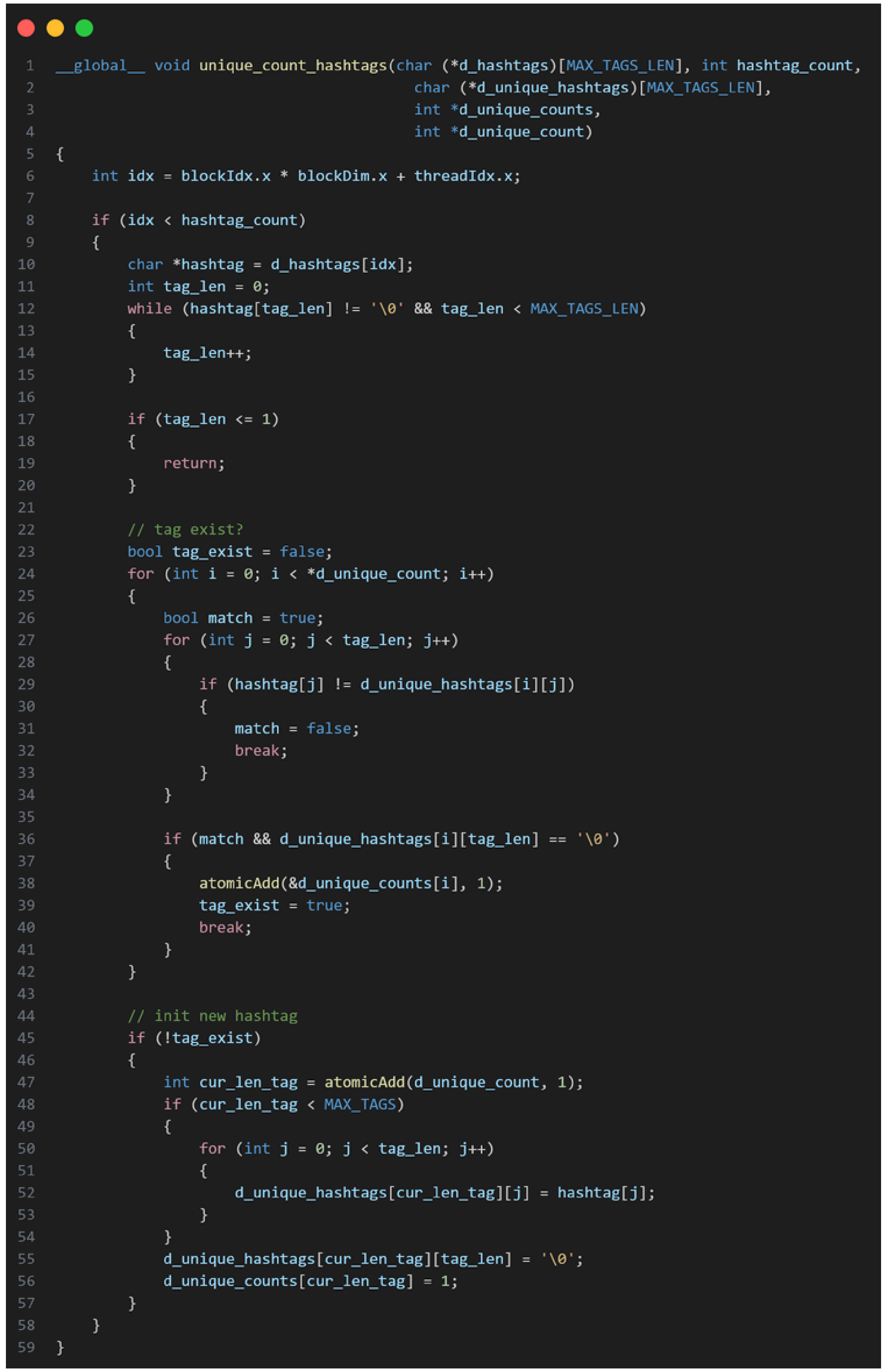
-> Not found -> Add to position 0

-> unique_hashtags[0] = "world", count[0] = 1

Thread n -> "CUDA"

-> Found at position m -> atomicAdd(&count[m])

-> count[m]++



```
__global__ void unique_count_hashtags(char (*d_hashtags)[MAX_TAGS_LEN], int hashtag_count,
                                      char (*d_unique_hashtags)[MAX_TAGS_LEN],
                                      int *d_unique_counts,
                                      int *d_unique_count)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    if (idx < hashtag_count)
    {
        char *hashtag = d_hashtags[idx];
        int tag_len = 0;
        while (hashtag[tag_len] != '\0' && tag_len < MAX_TAGS_LEN)
        {
            tag_len++;
        }

        if (tag_len <= 1)
        {
            return;
        }

        // tag exist?
        bool tag_exist = false;
        for (int i = 0; i < *d_unique_count; i++)
        {
            bool match = true;
            for (int j = 0; j < tag_len; j++)
            {
                if (hashtag[j] != d_unique_hashtags[i][j])
                {
                    match = false;
                    break;
                }
            }

            if (match && d_unique_hashtags[i][tag_len] == '\0')
            {
                atomicAdd(&d_unique_counts[i], 1);
                tag_exist = true;
                break;
            }
        }

        // init new hashtag
        if (!tag_exist)
        {
            int cur_len_tag = atomicAdd(d_unique_count, 1);
            if (cur_len_tag < MAX_TAGS)
            {
                for (int j = 0; j < tag_len; j++)
                {
                    d_unique_hashtags[cur_len_tag][j] = hashtag[j];
                }
            }
            d_unique_hashtags[cur_len_tag][tag_len] = '\0';
            d_unique_counts[cur_len_tag] = 1;
        }
    }
}
```

Result

process on 1.6m post with
cpu: i5 13500
GPU: Nvidia 3070Ti

Algorithm	Time process	Cmp
Sequential	476.734 ms	1X
Parallel optimize	181.837 ms	2.6X
Parallel optimize reduce	8.07261 ms	59X

Thank you