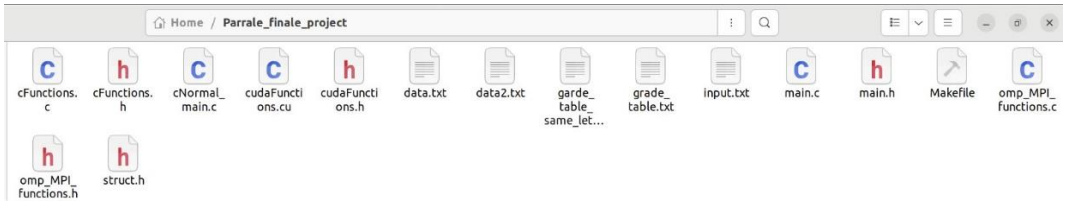
מחשוב מקבילי Project

שם מגישים: ים חורין, ירין מנוח

ת"ז: 208945956 318853256

בקובץ ישנה תיקייה שנקראת Parrale_finale_project ויש בה את הקבצים הנ"ל:



3 קבצי מידע שונים:

data.txt- 1000 מחרוזות שונות data2.txt- 10 מחרוזות שונות

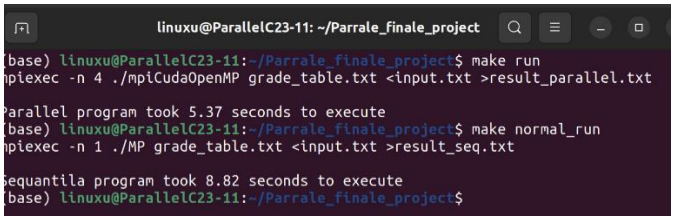
Input.txt- קובץ המידע הראשי- 4 מחרוזות בגודל של עד כ3000 תווים.

<u>שלבי הרצה:</u>

1. למען נוחיותך יש לעשות **Make all** על למנת ליצור את כלל קבצי ההרצה לבדיקה

ואם תרצה להריץ כל גרסה בנפרד יש לעשות כדלקמן:

2. **<u>גרסה מקבילה-Parallel:</u>** יש לכתוב make run- מאותחלת ל4 פרוססים פלט: `result_parallel.txt`
3. **<u>גרסה סדרתית- sequential:</u>** יש לכתוב make normal_run- מאותחל ל1 פרוססים פלט `result_seq.txt`



ההרצה הנ"ל תדפיס את הפלט הבא:
ניתן לראות כי הגרסה המקבילית אכן

מהירה יותר מהגרסה הסדרתית עבור קלט גדול

בשתי התוכניות הנ"ל RANK=0 מטפל גם בקלט וגם בפלט(לנוחיותך הפלט מופנה לקבצי txt)

הקלטים והפלטים מסודרים ע"פ אותו סדר הופעה וכלל תנאי הפרויקט מתקיימים.
במידה ותשמש ב make clean היא תנקה לך את כלל קבצי ההרצה בסיום.

יצרנו struct שנקרא score_alignment ובנוסף יצרנו MPI_Datatype:

```
struct score_alignment {
    int score;
    int K;
    int off_set;
    char str [MAX_STRING_SIZE];
};
```

אשר מחזיק בכלל המידע המודפס.

מצ"ב גם חלקי הקוד בהם באים לידי ביטוי:

MPI,OpenMp,CUDA

```
// Function to create a custom MPI data type for 'struct score_alignment'
void make_datatype(MPI_Datatype *mpi_score_alignment_type)
{
    // Making a new type for 'struct score_alignment'
    MPI_Datatype types[4] = {MPI_CHAR, MPI_INT, MPI_INT, MPI_INT};
    int block_lengths[4] = {MAX_STRING_SIZE, 1, 1, 1}; // Initialized to zeros
    MPI_Aint displacements[4];
    struct score_alignment temp; // Used to calculate displacements

    // Calculate displacements
    MPI_Get_address(&temp.str, &displacements[0]);
    MPI_Get_address(&temp.K, &displacements[1]);
    MPI_Get_address(&temp.score, &displacements[2]);
    MPI_Get_address(&temp.off_set, &displacements[3]);

    for (int i = 3; i > 0; i--)
    {
        displacements[i] -= displacements[0];
    }
    displacements[0] = 0;

    // Create the custom data type
    MPI_Type_create_struct(4, block_lengths, displacements, types, mpi_score_alignment_type);
    MPI_Type_commit(mpi_score_alignment_type);
}
```

```
// for each string in strings_to_check we caculate the result and print-the print
for (int i = 0; i < master_chunk_size; i++)
{
    char *str_to_check = strings_to_check[i];
    int score;

    if (how_to_caculate == NO_MATRIX_SCORE)
        score = caculate_cuda_without_matrix(str_to_check, first_str, my_rank);
    else
        score = caculate_cuda(str_to_check, first_str, matrix, my_rank);
}
```

```
// using a parallel space in the main
#pragma omp parallel for shared(alignment_scores_for_strings)
for (int i = 0; i < chunk_size; i++)
{

    if (how_to_caculate == THERE_IS_MATRIX_SCORE)
        caculate_max_score_grade_table(alignment_scores_for_strings[i].str, first_str, matrix, &alignment_scores_
    else
        caculate_max_score_no_grade_table(alignment_scores_for_strings[i].str, first_str, &alignment_scores_for_s
}

MPI_Send(alignment_scores_for_strings, chunk_size, mpi_score_alignment_type, ROOT, DONE, MPI_COMM_WORLD);
}
```