המכללה האקדמית להנדסה בכרמיאל
בראודה

המחלקה להנדסת
חשמל ואלקטרוניקה

קורס מערכות הפעלה

פרויקט מסכם

מגישים:

עידו בן הרוש ת"ז - 316439116

ליאור יבדאיב ת"ז –314991753

מרצה :

ד"ר רפאל שללה

הקדמה - בפרויקט זה, פיתחנו מערכות לזיהוי חום יתר ולזיהוי חיישנים תקולים, בהתבסס על נתונים שמתקבלים מחיישני טמפרטורה שונים. הנתונים יועברו בקבצים נפרדים, כאשר כל קובץ מייצג חיישן טמפרטורה יחיד. מטרת הפרויקט היא ליישם תכנות מקבילי כדי לשפר את הביצועים ולנצל בצורה אופטימלית את משאבי המערכת (יחידות העיבוד- Processing Cores).

פירוט שלבי הפרויקט:

1. קריאת נתונים:

בשלב זה פתחנו את התיקייה Data אשר בתוכה ממקומיים הקבצי מידע של החיישנים זה עשינו על פי הקוד הבא:

```
directory = 'Data'

files = []

for file in os.listdir(directory):

    full_path = os.path.join(directory, file)

    if os.path.isfile(full_path):

        files.append(file)
```

לאחר מכן קראנו את המידע של הקבצים על ידי 3 -Threads עשינו זאת מכיוון שזהו תהליך קלאסי של I\O במקרים כאלו התכנות המקבילי המתאים הינו ריבוי תהליכונים, היה שימוש ב3 תהליכונים בגלל מספר החיישנים הקיימים כרגע ,אך אם היו יותר קבצי מידע על חיישנים אז בהתאם מספר התהליכונים.

בנוסף ,שימוש בתהליכונים לקריאה מקבצים במקביל מאפשר ניצול אופטימלי של זמן המעבד ומקצר את זמן ההמתנה הכולל על ידי חלוקת העומס בין תהליכים שונים. זה גם מאפשר גישה בטוחה למידע משותף ומייעל את ביצועי התוכנית בסביבות עם מספר ליבות מעבד.

```
        threads = []

for i, file in enumerate(files):

    full_path = os.path.join(directory, file)

    thread = threading.Thread(target=read_file, args=(full_path, all_data[i], locks[i]))

    threads.append(thread)

    thread.start()
```

```
def read_file(file_name, all_data, lock):
    try:
        with open(file_name, 'r') as f:
            data = [float(line.strip()) for line in f.readlines() if line.strip()]
            lock.acquire()
            all_data.extend(data)
            lock.release()
    except Exception as e:
        print(f"Error reading file {file_name}: {e}")
```

השתמשנו במנעול בזמן קריאת הקבצים כדי למנוע התנגשות בין התהליכונים למרות שבפייתון עקב הGIL זה לא קורה (כדי רק תהליכון אחד נכנס בכל פעם בפייתון בלבד) אבל בשביל המקרה החריג או הקיצוני ביותר עשינו זאת.

במידה ולא נצליח לקרוא מידע מהקובץ בגלל שגיאה אז נקבל הודעת שגיאה ואת שם הקובץ שבו מופיעה השגיאה.

נקבל לאחר השלב הזה את הפלט הבא שמכיל מערך של 3 מערכים(מערך פנימי עבור כל חיישן) וזה יהיה במשתנה בשם all_data :

```
[[76.36, 77.15, 75.93, 77.36, 77.64, 74.94, 75.67, 77.81, 75.92, 75.28, 73.45, 76.5, 74.29, 74.39, 73.26, 77.43, 74.89, 72.95, 7
```

2. החלקת הנתונים:

החלקת הנתונים התבצעה באמצעות ריבוי תהליכים , מכיוון שבמקרה של חישובים קשים ומורכבים יש להשתמש בתכנות מקבילי זה.

שימוש בריבוי תהליכים בקוד לחישוב EMA יכול לשפר ביצועים על ידי חלוקת העבודה בין מספר תהליכים, במיוחד כאשר יש לחשב EMA עבור מספר רשימות נתונים גדולות בו-זמנית. זה מאפשר ניצול טוב יותר של ליבות המעבד ומקצר את הזמן הכולל לחישוב.

כדי לבצע זאת בתחילה נמצא את מספר המעבדים הקיימים במחשב כדי להגיע לביצועים הטובים ביותר והיעילים ביותר (להימנע מתור מצד אחד ומצד שני להימנע מחוסר שימוש במעבדים זמינים).

```
def ema(data, alpha=0.3):
    if not data:
        return []
```

```
ema_values = [data[0]]
    ema_value = data[0]
    for value in data[1:]:
        ema_value = alpha * value + (1 - alpha) * ema_value
        ema_values.append(ema_value)
    return ema_values
num_process = multiprocessing.cpu_count()  # Get the number of CPU cores available on the system
    pool = multiprocessing.Pool(processes=num_process)
    results = pool.map(process_data_list, all_data)
    pool.close()
    pool.join()
```

- יש לציין כי בפונקציה של Process_data_list יש קריאה לפונקציה של החלקת הנתונים ונראה זאת בהמשך.

3. חישוב סטטיסטיקות:

חישובים אלו נעשה באמצעות שימוש בריבוי תהליכים שעשינו קודם באותן הסיבות , הפונקציה Process_data_list תרכז את הקריאות לפונקציות של החלקה הנתונים ובנוסף לחישוב הסטטיסטיקות. חישוב הקטעי זמן התבצעו במרווחים של 30 אך ניתנים לשינוי פשוט על ידי שינוי של משתנה הקבוע שהוגדר בתחילת התוכנית - TIME_SEGMENT = 30

```
def statics(ema_results, segment_size=TIME_SEGMENT):
    stat_result = []
    for i in range(0, len(ema_results), segment_size):
        res = ema_results[i:i + segment_size]
        if res:  # Avoid division by zero
            res_mean = sum(res) / len(res)
            res_max = max(res)
```

```
    res_min = min(res)
        stat_result.append((res_mean, res_min, res_max))


    return stat_result
def process_data_list(data_list):
    try:
      if data_list:
        ema_results = ema(data_list)
        segment_statistics = statics(ema_results, segment_size=30)
        return segment_statistics
    except Exception as e:
      print(f"Error in process_data_list for index : {e}")
```

במידה ולא יהיה לנו מידע לעיבוד על ידי החישובים הנדרשים נקבל הודעת שגיאה עם מספר האינדקס שבו יש את השגיאה.

דוגמא ספציפית להדפסה לאחר 2 שלבים של חישובים אלו במשתנה בשם results :

```
[[(75.22628449697613, 73.40459355752813, 76.97208099999999), (74.39332891060481, 73.36727026917868, 75.96791700361521), (77.88683
```

שלב ביניים:

ביצענו מספר חישובים ופעולות על גבי המידע שיש לנו כדי להעביר את המידע להיות מבני נתונים של מילון כדי שיהיה לנו יותר נוח לעבור על המידע , זאת בכדי להקל בהמשך על הוויזואליות , בדיקה ופתרון בעיות ההמשך:

```
segment_statistics_by_index = {}
for idx in range(len(results)):  # number of the sensors
    for idx2 in range(len(results[idx])):  # number of the samples/30
        if idx2 not in segment_statistics_by_index:
            segment_statistics_by_index[idx2] = []
        segment_statistics_by_index[idx2].append(results[idx][idx2])
```

לאחר סידור הנתונים נקבל מילון שבו במפתח נקבל את מספר האינדקס (שזה בעצם מקטע הזמן) ובערכים נקבל מערך של 3 טאפלים שמכילים את המידע עבור כל חיישן וחיישן במשתנה בשם segment_statistics_by_index :

```
{0: [(75.22628449697613, 73.40459355752813, 76.97208099999999), (75.07873553041404, 72.50825405033288, 78.121), (75.1525116040429
```

4.1. ניתוח וזיהוי – זיהוי חום יתר:

עבור חישובים של זיהוי יתר בחיישן לעומת שאר החיישנים על פי קטע זמן מסוים התבצע באמצעות ריבוי תהליכים מכיוון שהתבצעו פה חישובים ופעולות מתמטיות שמתאימים לריבוי תהליכים ,ובנוסף במקרה והמידע הנתון יהיה ארוך או מורכב יותר זה יעזור לנו ליעילות , ובכך פתחנו פתרון זה למקרים כללים ומסובכים יותר.

מכיוון שיש לבדוק את ממוצע החיישנים ביחס לרוב שאר החיישנים , במקרה הנתון מעל רף של 90 מעלות (גם במקרה הזה המספר הוא בר שינוי באופן פשוט כי הוא הוגדר כמספר קבוע THRESHOLD_TEMP = 90), אזי הגדרנו ולקחנו את גודל המערך וחילקנו ב2 ואם מספר הממוצעים (שגדולים מ90) גדול מכמות זו אזי נדפיס את הקטע שבו יש שגיאת זיהוי חום יתר.

כמו גם במקרים הקודים של ריבוי תהליכים גם פה נבצע שימוש במספר המעבדים הקיימים במחשב.

```
def excessive_heat(args):

    segment_idx, statistics_list = args

    count_above_90 = sum(mean > THRESHOLD_TEMP for mean, _, _ in statistics_list)

    if count_above_90 > len(statistics_list) / 2:

        start_time = TIME_SEGMENT * segment_idx

        end_time = TIME_SEGMENT * (segment_idx+1)

        print(f'Temperature alert detected at timeline {start_time}-{end_time}')

    pool = multiprocessing.Pool(processes=num_process)

    pool.map(excessive_heat, segment_statistics_by_index.items())

    mean_sensors_defect = pool.map(mean_ranges, segment_statistics_by_index.items())

    pool.close()

    pool.join()
```

במקרה של הקבצים המצורפים הספציפיים נקבל את ההדפסים הבאים , זאת אומרת במקטעי זמנים הללו ישנה שגיאה בדרישות סף:

```
Temperature alert detected at timeline 120-150
Temperature alert detected at timeline 150-180
Temperature alert detected at timeline 180-210
Temperature alert detected at timeline 210-240
Temperature alert detected at timeline 240-270
```

4.2 זיהוי חיישן תקול:

בפתרון שלב זה גם כן ביצענו חישובים וביצועים על ידי ריבוי תהליכים כי במקרה זה ישנם פעולות רבות שיש לבצע ובכדי לייעל ולגרום לרמת ביצועים הטובה ביותר נעשה שימוש בכל המעבדים הקיימים במחשב.

במקרה זה עלינו לבדוק האם ממוצע הטמפרטורה של החיישן לא נמצא בטווח מינימום ומקסימום של רוב החיישנים האחרים בכמות של לפחות חצי מקטעי זמן הנתונים, אם חיישן עומד בקריטריונים אלו אזי הוא יהיה החיישן התקול.

זה ייקרה באמצעות כך שעבור כל מקטע זמן אסמן על גבי מערך איזה חיישן לא נמצא בטווח המבוקש ואם זה קורה אסמן חיישן זה ב1 עבור מקטע זמן זה, וכך עבור כל מקטעי הזמן ואקבל בסופו של דבר לצורך הדוגמא הנוכחית את המערך הבא(יוצא לאחר מימוש הפונקציה):

```
def mean_ranges(args):
    segment_idx, statistics_list = args
    # Initialize counters for each sensor
    defect_counts = [0] * len(statistics_list)
    # Iterate through each sensor's statistics
    for i, (mean, min_val, max_val) in enumerate(statistics_list):
        outside_range_count = 0
        # Check if the mean is outside the range of other sensors
        other_sensors_stats = [stats for j, stats in enumerate(statistics_list) if j != i]
```

```
    for _, min_val_other, max_val_other in other_sensors_stats:
        if not (min_val_other <= mean <= max_val_other):
            outside_range_count += 1
    # Update count only if the mean is outside the range of at least two other sensors
    if outside_range_count > len(statistics_list)/2:
        defect_counts[i] += 1
    return defect_counts
```

פלט לדוגמא של הפונקציה הזו במשתנה בשם mean_sensors_defect :

```
[[0, 0, 0], [0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 1, 1], [0, 1, 0], [0, 1, 0], [0, 1, 0], [0, 1, 1], [1, 1, 1], [1, 1, 1], [0, 1
```

לאחר מכן אני נסכום את כל ה'1' של כל חיישן ואקבל את המערך הבא במשתנה בשם summed_list :

```
[4, 10, 4]
```

זאת אומרת שחיישן בעל אינדקס 1 יש לו מעל מספר הסף של שגיאות הממוצע (שבמקרה שלנו התנאי סף הינו לפחות 6) , בכל מקרה אחר יכול להיות שמספר חיישנים יהיו בעלי השגיאות ולא רק אחד .

לכן לבסוף אבצע את הקוד הבא כדי להדפיס את שב הקובץ של החיישן התקול:

```
defective_lst = []
defective_sensors = []
threshold_samples = len(segment_statistics_by_index)/2  # most of the samples time at least 6 in this case
# Sum the defect counts and determine defective sensors
summed_list = [sum(x) for x in zip(*mean_sensors_defect)]
print(summed_list)
for i in range(len(summed_list)):
    if summed_list[i] >= threshold_samples:
        defective_lst.append(i)
```

```
if defective_lst:
    for i in defective_lst:
        defective_sensors.append(files[i])

    print(f'Sensors {defective_sensors} are suspected for malfunction')
```

ולבסוף במקרה הספציפי הנ"ל אקבל את ההדפסה הזו שמייצגת לי מהו\ מהם החיישנים התקולים:

```
Sensors ['no_response_sensor.txt'] are suspected for malfunction
```

לסיכום, בפרויקט זה השתמשנו בתכנות מקבילי עבור הפעולות והשלבים הדרושים כדי לפתור בעיות אלו, חוץ מקריאת הנתונים מהקבצים שבו השתמשנו ריבוי תהליכונים השאר היה ריבוי תהליכים עקב הסיבות שצוינו מעלה. תוכנית זו בכללותה תפעל גם עבור מספר רב יותר של חיישנים ועבור פרמטרים אחרים הניתנים לשינוי.