תכנות מקבילי ומבוזר תרגיל בית 3

שמות המגישים: יובל נהון, סהר כהן

ת.ז:206866832,206824088

.2

רגיל:

```
Epoch 1, accuracy 58.72 %.
Epoch 2, accuracy 86.97 %.
Epoch 3, accuracy 90.18 %.
Epoch 4, accuracy 91.67 %.
Epoch 5, accuracy 92.43 %.
Time reg: 99.07910776138306
Test Accuracy: 92.13%
```

סינכרוני:

4 ליבות:

```
Epoch 1, accuracy 10.64 %.
Epoch 2, accuracy 65.86 %.
Epoch 3, accuracy 83.76 %.
Epoch 4, accuracy 88.67 %.
Epoch 5, accuracy 91.31 %.
Time sync: 1000.5303640365601
Test Accuracy: 90.45%
```

8 ליבות:

```
Epoch 1, accuracy 9.15 %.
Epoch 2, accuracy 29.68 %.
Epoch 3, accuracy 75.78 %.
Epoch 4, accuracy 87.07 %.
Epoch 5, accuracy 89.36 %.
Time sync: 16.979737281799316
Test Accuracy: 89.37%
```

16 ליבות:

```
Epoch 1, accuracy 9.61 %.
Epoch 2, accuracy 10.75 %.
Epoch 3, accuracy 43.43 %.
Epoch 4, accuracy 81.23 %.
Epoch 5, accuracy 88.58 %.
Time sync: 145.32925701141357
Test Accuracy: 88.05%
```

ניתן לראות שעבור 8 ליבות מתקבלת האצה אופטימלית. הסיבה לכך היא שבהרצה עם 4 ליבות המיקבול הוא מצומצם יותר ועבור 16 ליבות ה-overhead של הסנכרון בין כל התהליכים גובר על ההאצה שמתקבלת בזכות כמות התהליכים.

3. ההאצה תואמת לגישה של אמדל מכיוון שעל פיו לא ניתן להגדיל את מהירות החישוב(speedup) מעבר לגבול מסויים עבור תוכנית כלשהיא. לכן המעבר מ-8 ליבות ל-16 ליבות לא חולל שינוי ואף פגע בביצועי התוכנית בגלל הסינכרון הנדרש בין כל 16 התהליכים.

.5

:2 masters 4 cores

```
Epoch 1, accuracy 9.9 %.
Epoch 2, accuracy 11.94 %.
Epoch 3, accuracy 9.91 %.
Epoch 4, accuracy 9.91 %.
Epoch 5, accuracy 9.91 %.
Time async: 52.6949348449707
Test Accuracy: 90.32%
```

:2 masters 8 cores

```
Epoch 1, accuracy 9.9 %.
Epoch 2, accuracy 9.9 %.
Epoch 3, accuracy 9.9 %.
Epoch 4, accuracy 9.9 %.
Epoch 5, accuracy 9.9 %.
Time async: 28.294236183166504
Test Accuracy: 9.58%
```

:4 masters 8 cores

```
Epoch 1, accuracy 9.91 %.
Epoch 2, accuracy 9.91 %.
Epoch 3, accuracy 9.91 %.
Epoch 4, accuracy 9.91 %.
Epoch 5, accuracy 9.91 %.
Time async: 34.19700813293457
Test Accuracy: 88.82%
```

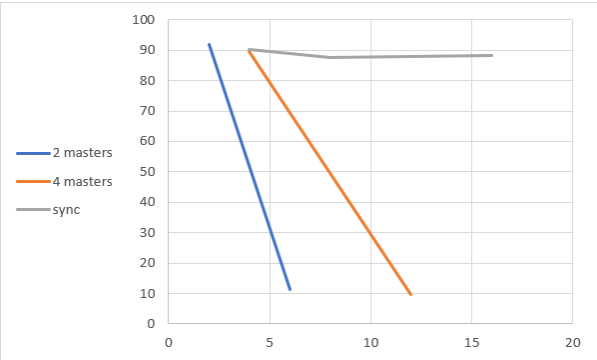
:4 masters 16 cores

```
Epoch 1, accuracy 9.83 %.
Epoch 2, accuracy 9.83 %.
Epoch 3, accuracy 9.83 %.
Epoch 4, accuracy 9.83 %.
Epoch 5, accuracy 9.83 %.
Time async: 25.340981483459473
Test Accuracy: 10.09%
```

כפי שניתן לראות, עלייה בכמות הליבות תמיד מאיצה את החישוב מכיוון שאין צורך בסנכרון בין התהליכים השונים – בניגוד להרצות הסינכרוניות מקודם. מצד שני, בגלל חוסר הסנכרון אחוזי הדיוק יורדים דרסטית (למשל בהרצה האחרונה בה רמת הדיוק הגיעה רק ל-10 אחוז). הסיבה לכך היא חוסר הסנכרון בין המידע שה-workers עובדים עליו לעומת מה שנמצא כרגע ב-masters שלהם.

6. הפיצול למספר מרובה של masters נעשה כדי שהן יהיו זמינות יותר. בצורה זו ה-masters שולחים ערכים מעודכנים לעובדים בפרקי זמן קצרים יותר.

.7



8. כאשר כמות העובדים במימוש האסינכרוני גדל יותר מדי, ה-workers מבצעים את עבודתם על מידע יותר ויותר לא מסונכרן ולכן החישוב שהם מבצעים הופך לפחות ופחות רלוונטי.

9. בגישה האסינכרונית העלאת מספר ה-workers תמיד תוביל לשיפור מבחינת זמן הריצה אך מרף מסויים גם תגרום לירידה בדיוק של החישוב בצורה משמעותית ולכן תוצאתה תהיה איכותית פחות. לעומתה, הגישה הסינכרונית חסומה מבחינת שיפור זמן ריצה שיכול להיווצר משימוש ביותר ליבות בגלל ה-overhead שנגרם מהתקשורת (הסינכרון) של העובדים, אבל גישה זו תמיד תוביל לתוצאות טובות מבחינת אחוזי דיוק.

.10

:2 cores

```
array size: 4096
naive impl time: 0.0002601146697998047
ring impl time: 0.0001995563507080078
array size: 8192
naive impl time: 0.00020766258239746094
ring impl time: 0.0002696514129638672
array size: 16384
naive impl time: 0.00033330917358398844
ring impl time: 0.00026512145996093 75
```

:4 cores

```
array size: 4096
naive impl time: 0.00101804733276367 19
ring impl time: 0.00019240379333496094
array size: 8192
naive impl time: 0.00046181678771972656
ring impl time: 0.0003902912139892578
array size: 16384
naive impl time: 0.0007636547088623047
ring impl time: 0.0006394386291503906
```

:8 cores

```
array size: 4096
naive impl time: 0.0013346672058105469
ring impl time: 0.0006425380706787109
array size: 8192
naive impl time: 0.0015063285827636719
ring impl time: 0.0008180141448974609
array size: 16384
naive impl time: 0.0024249553680419 92
ring impl time: 0.0010416507720947266
```

כפי שניתן לראות ring all-reduce תמיד מהיר יותר מ-naïve all-reduce והיחס בין המהירויות שלהם (speedup) גדל ככל שיש יותר ליבות במערכת. לפי ניתוח הסיבוכיות בסעיפים הבאים ניתן להבין גם למה.

11. כל תהליך שולח את כל המערך שלו לכל שאר התהליכים בעלות כוללת של $O(n*s)$ כאשר n זהו גודל המערך ו-s זוהי כמות התהליכים. לכן נשלחים בסך הכל: $O(n*s*s) = O(n*s^2)$

12. ring allreduce מחולק ל-2 שלבים:

1) כל תהליך שולח $\frac{1}{s}$ מהמערך לתהליך הבא (לפי rank) s-1 פעמים למטרת סכימה.

2) כל תהליך שולח $\frac{1}{s}$ מהמערך לתהליך הבא (לפי rank) s-1 פעמים למטרת העתקה.

סה"כ עבור תהליך בודד:

$$O\left(n * \frac{2 * (s-1)}{s}\right) = O(n)$$

לכן בסך הכל נשלחים $O(n * s)$ בתים.

לכן ככל שיש יותר ליבות, סיבוכיות all-reduce הנאיבי גדלה הרבה יותר בתלות בכמות הליבות ביחס ל- ring all-reduce שסיבוכיותו לינארית בכמות הליבות.