תרגיל בית 3

איתי לביא 212147326 הראל ימין 213099492

1. **<u>מימוש חלק סינכרוני-</u>**

נסביר בדיוק מה עשינו בכל אחת מהמשימות של החלק הראשון.
תחילה, עבור המימוש של ringallreduce באמצעות mpiCollective. פשוט החלפנו את הקריאה ל ringAllReduce-בקריאה לפונקציה של comm.
במימוש הנאיבי של ringAllReduce אנחנו דאגנו שכל עובד יעבור על כל אחד מהתהליכים ואו יקבל ממנו את המידע ו"יסכום" אותו, או שהוא ישלח את המידע שלו לכל התהליכים האחרים.
המימוש שלנו של ringAllReduce כבר יותר מוצלח. תחילה חילקנו את המערך שליחה לחלקים שווים לפי מספר התהליכים, אם מספר התהליכים היה גדול מגודל המערך אנחנו הרחבנו אותו עד למינימום האפשרי עבור אותו מספר תהליכים כדי שהכל יתרחש באופן סימטרי. לאחר שחילקנו את המערך שווה בשווה בדקנו כי השארית של החלוקה מתחלקת יפה, ושלא יישארו מקומות במערך אשר אף תהליך לא שולח או להיפך, שיהיו מקומות שישלחו על ידי שני תהליכים שונים. יצרנו מערך של גודל השליחה של כל תא ושל נקודת ההתחלה שלו ובעזרתם כל תהליך ידע עבור תא מסוים לקבל ולשלוח בדיוק תא הקואורדינטות הנכונות.
לאחר מכן כל תהליך בדרגה מסוימת ישלח אל התהליך בדרגה הבאה אחריו כל פעם משבצת אחרת מהמערך, כמו פס נע וכמו שראינו בתרגולים בעצם בכל איטרציה נעביר את החלק הבא במערך לפי החלוקה שלנו כך שאנחנו מתחילים בחלק במקום rank.וכמובן מקבל מקודמו משבצת אחרת. במעבר הראשון כל תהליך מבצע זאת size-1פעמים וסוכם בעזרת (op לא בהכרח סוכם אבל מבצע פעולת שמסכמת את המידע) את המידע שהוא מקבל עד שהסכום של כל המשבצות המתאמות במערך נמצא לפחות אצל תהליך אחד וכעת מתחיל השלב השני בו אנו פשוט נבצע את אותו הדבר אך במקום לסכום את המידע פשוט נשים אותו במקום המתאים. גם זה יתבצע size-1איטרציות. בסוף שני הסבבים כל תהליך אמור להחזיק את כל המידע כדרוש.

<u>מימוש חלק אסינכרוני-</u>

<u>פונקציית ה-worker</u>

ראשית, כדי לחשב על כמה batches כל עובד צריך להיות אחראי, ביצענו חילוק בין מספר הbatches למספר העובדים (ועיגלנו למטה כדי לקבל מספר שלם). אם הם לא מתחלקים זה בזה, פיזרנו את הbatches שנותרו בין העובדים הראשונים (לפי סדר הrank).
(לדוגמה: אם מספר הbatches הכולל שעלינו לחשב הוא 60 ומספר העובדים הוא 9 אז על כל עובד לבצע $\left\lfloor \frac{60}{9} \right\rfloor = 6$, ו-6 העובדים הראשונים יבצעו batches אחד נוסף, כלומר 7 batches.)
בכל epoch יצרנו mini batches ולכל אחד מהם ביצענו forward ו backward propagation כדי לחשב את ערכי הגרדיאנט. שלחנו באופן אסינכרוני לכל מאסטר את תוצאות הגרדיאנטים שלנו (של המשקלים והbiases) עבור השכבות עליהן הוא אחראי ובו זמנית שלחנו בקשה לקבל ממנו את המשקלים והbiases החדשים שנשתמש בהם בbatch הבא. לאחר ששלחנו את כל הבקשות למאסטרים חיכינו שכל התשובות מהמאסטרים תתקבלנה (אין צורך לחכות לבקשות ששלחנו למאסטרים, כי בהכרח הם ישלחו אלינו מידע רק לאחר שנבקש אותן מהם מלכתחילה). לאחר מכן עוברים ל mini

batch הבא, וכשנגמרים הbatchים מתחילים בepoch חדש.

<u>פונקציית ה-master</u>

רצנו בלולאה על כמות הepochים והbatchים שאמורים להיות ובכל איטרציה ביצענו את השלבים הבאים:

- ביצענו לולאת busy wait שבודקת האם יש worker שמחכה לקבל מאיתנו שירות. עשינו זאת בעזרת Iprobe.
- כאשר יש עובד ממתין, קיבלנו ממנו את הגרדיאנטים הנוכחיים שלו עבור השכבות שעליהן המאסטר שלנו אחראי. (עשינו זאת באופן אסינכרוני) ולאחר מכן חיכינו שכל המידע ממנו יתקבל.
- לאחר מכן חישבנו את המשקלים והbiases החדשים עבור השכבות שלנו, ושלחנו אותן באופן אסינכרוני לworker שבו טיפלנו.

מחכים שכל השליחה תסתיים ואז ממשיכים לאיטרציה הבאה.

לאחר שכל הepochים הסתיימו, שלחנו במקביל את המשקלים והbiases מכל המאסטרים האחרים למאסטר שהrank שלו הוא אפס וחיכינו שכולם יתקבלו על מנת לסיים את הריצה.

2. ההדפסות שהתקבלו מההרצות:

ליבות	צילום מסך	ליבות	צילום מסך
4	Epoch 1, accuracy 52.77 %. Epoch 2, accuracy 87.22 %. Epoch 3, accuracy 89.92 %. Epoch 4, accuracy 91.61 %. Epoch 5, accuracy 91.99 %. Time reg: 7.678053617477417 Test Accuracy: 91.47% Epoch 1, accuracy 10.64 %. Epoch 2, accuracy 63.86 %. Epoch 3, accuracy 85.73 %. Epoch 4, accuracy 89.24 %. Epoch 5, accuracy 90.43 %. Time sync: 34.08777737617493 Test Accuracy: 90.42%	8	Epoch 1, accuracy 49.99 %. Epoch 2, accuracy 86.34 %. Epoch 3, accuracy 90.18 %. Epoch 4, accuracy 91.17 %. Epoch 5, accuracy 92.33 %. Time reg: 8.75504732131958 Test Accuracy: 91.86% Epoch 1, accuracy 10.64 %. Epoch 2, accuracy 29.99 %. Epoch 3, accuracy 68.88 %. Epoch 4, accuracy 86.96 %. Epoch 5, accuracy 90.16 %. Time sync: 42.034382820129395 Test Accuracy: 89.65%
16	Epoch 1, accuracy 59.76 %. Epoch 2, accuracy 86.47 %. Epoch 3, accuracy 89.47 %. Epoch 4, accuracy 91.17 %. Epoch 5, accuracy 92.32 %. Time reg: 14.595786809921265 Test Accuracy: 91.93% Epoch 1, accuracy 10.9 %. Epoch 2, accuracy 10.9 %. Epoch 3, accuracy 35.87 %. Epoch 4, accuracy 75.84 %. Epoch 5, accuracy 88.33 %. Time sync: 58.13197946548462 Test Accuracy: 87.43%		

3. כפי שניתן לראות ככל שגדל מספר התהליכים כך גם גדל זמן הביצוע, ההסבר לזה יכול להיות שאנחנו צריכים לסנכרן בין התהליכים וככל שיש יותר כאלו הזמן גדל.
הסנכרון יכול להיות יקר מאוד כפי שאנחנו רואים כאן, ככל שמספר התהליכים גדל כל תהליך צריך לשלוח יותר פיסות מידע וגם לחכות להן ובכך אנחנו מוסיפים זמן לתהליך.

4. האלגוריתם שלנו משיג האצה באמצעות העקרון של אמדהל, מכיוון שאנו שומרים על גודל בעיה קבוע ומנסים להגיע לביצועים טובים יותר באמצעות הוספת חוטים. לעומת זאת העקרון של גוסטפסון אומר שאנחנו מגדילים את הבעיה ואת מספר החוטים יחד על מנת לפתור בעיה גדולה באותו פרק זמן. לכן ככל שנוסיף חוטים נרצה גם להגדיל את הבעיה ולנסות לשמור על אותו זמן של פתרון.
5. ההדפסות שהתקבלו מההרצות:

מאסטרים	ליבות	צילום מסך	ליבות	צילום מסך
2	4	Epoch 1, accuracy 57.08 %. Epoch 2, accuracy 86.06 %. Epoch 3, accuracy 90.37 %. Epoch 4, accuracy 91.64 %. Epoch 5, accuracy 92.25 %. Time reg: 7.136333703994751 Test Accuracy: 91.88% Epoch 1, accuracy 9.15 %. Epoch 2, accuracy 9.15 %. Epoch 3, accuracy 9.9 %. Epoch 4, accuracy 9.15 %. Epoch 5, accuracy 9.15 %. Time async: 4.371190071105957 Test Accuracy: 91.2%	8	Epoch 1, accuracy 57.26 %. Epoch 2, accuracy 87.19 %. Epoch 3, accuracy 90.62 %. Epoch 4, accuracy 91.42 %. Epoch 5, accuracy 92.09 %. Time reg: 8.307649850845337 Test Accuracy: 91.61% Epoch 1, accuracy 9.67 %. Epoch 2, accuracy 9.67 %. Epoch 3, accuracy 9.67 %. Epoch 4, accuracy 9.67 %. Epoch 5, accuracy 9.67 %. Time async: 3.7144644260406494 Test Accuracy: 10.1%
4	8	Epoch 1, accuracy 54.39 %. Epoch 2, accuracy 85.63 %. Epoch 3, accuracy 90.31 %. Epoch 4, accuracy 91.37 %. Epoch 5, accuracy 91.85 %. Time reg: 7.826135873794556 Test Accuracy: 91.54% Epoch 1, accuracy 9.36 %. Epoch 2, accuracy 9.36 %. Epoch 3, accuracy 9.36 %. Epoch 4, accuracy 9.36 %. Epoch 5, accuracy 9.36 %. Time async: 3.33595609664917 Test Accuracy: 88.04%	16	Epoch 1, accuracy 65.89 %. Epoch 2, accuracy 86.55 %. Epoch 3, accuracy 89.92 %. Epoch 4, accuracy 91.39 %. Epoch 5, accuracy 91.62 %. Time reg: 16.81480598449707 Test Accuracy: 90.65% Epoch 1, accuracy 9.61 %. Epoch 2, accuracy 9.61 %. Epoch 3, accuracy 9.61 %. Epoch 4, accuracy 9.61 %. Epoch 5, accuracy 9.61 %. Time async: 3.83156156539917 Test Accuracy: 10.09%

6. כפי שניתן לראות מצילומי המסך, **אחוז הדיוק יורד משמעותית** כאשר מספר הליבות גדול בהרבה ממספר המאסטרים.

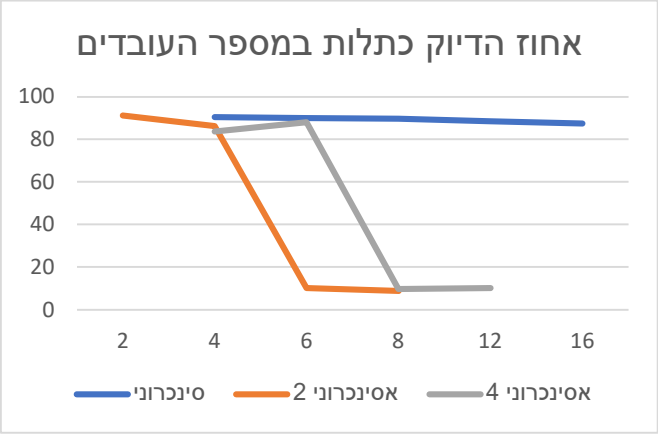
זה קורה מפני שכאשר מגדילים את מספר הליבות, מגדילים את מספר העובדים ולכן מספר הbatchים שאנחנו מריצים (שנשאר קבוע) מתחלק בין יותר עובדים. לכן זה יוצר מצב שכל עובד אחראי על פחות batchים ובכך המידע שלו פחות אמין והדיוק הכללי של הרשת פוחת.

בנוסף, **זמן הריצה כמעט ולא השתפר כלל** כשהרצנו את התוכנית על יותר ליבות. המאסטרים יעבדו כנראה ללא הפסקה (יש יותר עובדים כעת ולכן בכל רגע המאסטרים יקבלו בקשה מעובד), לכן לכאורה אנחנו אמורים לסיים לבצע את העבודה מהר יותר.

אבל היתרון פה הוא גם החיסרון, בגלל שיש יותר עובדים אז כל מאסטר מתקשר עם יותר חוטים ומקבל יותר בקשות בכל רגע ולכן תהיה תקורה של זמן שנובעת גם מההמתנה של העובדים לקבלת שירות (הם צריכים לקבל שירות מכל מאסטר!) וגם ממספר הודעות גדול יותר שהמאסטרים צריכים לטפל בו.

7. אנחנו מפצלים את הפרמטר סרבר בין חוטים (מאסטרים) שונים על מנת לשפר את הסקלביליות של הtraining. אם נשתמש במאסטר אחד ויהיו המון עובדים, הוא יצטרך

לשרת את כולם ולחשב את הגרדיאנטים החדשים עבור כל עובד באופן סדרתי ולא מקבילי. לכן זה יכול לגרום להרעבה ולהאטה של התוכנית באופן משמעותי.



8.

9. המימוש האסינכרוני סובל מאחוזי דיוק נמוכים כאשר מס' העובדים גדול מפני שכאשר מס' העובדים גדול, הגרדיאנטים שמחושבים על ידי המאסטר מבוססים על weights, biases ישנים יותר מהפרמטרים הנוכחיים של המאסטר. לכן כל עדכון של מידע מתבסס על מידע מיושן והאימון לא מתקדם. תופעה זו נקראת Gradient Staleness.

10. היתרון של השיטה הסינכרונית הוא שהיא מספקת **דיוק** מכיוון שבכל epoch, כל העובדים עובדים על מידע זהה ולכן הנתונים שלהם עדכניים ורלוונטיים. בפרט לא נסבול מGradient Staleness.
לעומת זאת, החסרון של השיטה הסינכרונית הוא שהמאסטר מחכה בכל epoch לקבל מידע מכל העובדים במערכת ולכן השיטה יותר **איטית.**
בשיטה האסינכרונית המצב בדיוק הפוך. היתרון שלה הוא שהמידע מתעדכן לכל עובד מבלי שעליו לחכות לכל שאר העובדים, וגם שניתן להשתמש במספר מאסטרים כדי למקבל את חישוב הגרדיאנטים. לכן היא **מהירה יותר**. לעומת זאת, החיסרון שלה הוא שכפי שראינו, עבור מס' רב של עובדים אחוזי הדיוק קורסים. (Gradient Staleness)

11. אנו רואים שבכל שמספר המעבדים גדל, כמו קודם, זמן הריצה גדל בשתי השיטות ובאופן כללי רואים ש ringAllReduce-מהיר יותר מהשיטה הנאיבית שכן הוא רץ בסיבוכיות יותר טובה. בנוסף ניתן לראות כי גם זמן הריצה גדל כתלות בגודל המערך

שכן סיבוכיות הריצה תלויה גם בו.

```
(tf23-gpu) harely@lambda:~/MC3$ srun -K -n 4 --mpi=pmi2 --pty python3 allreduce_test.py
array size: 4096
naive impl time: 0.0019881725311279297
ring impl time: 0.0007369518280029297
array size: 8192
naive impl time: 0.002443552017211914
ring impl time: 0.0008413791656494141
array size: 16384
naive impl time: 0.026414155960083008
ring impl time: 0.0014796257019042969
(tf23-gpu) harely@lambda:~/MC3$ srun -K -n 2 --mpi=pmi2 --pty python3 allreduce_test.py
array size: 4096
naive impl time: 0.0012078285217285156
ring impl time: 0.0004177093505859375
array size: 8192
naive impl time: 0.0010616779327392578
ring impl time: 0.0005838871002197266
array size: 16384
naive impl time: 0.002003908157348633
ring impl time: 0.0009949207305908203
(tf23-gpu) harely@lambda:~/MC3$ srun -K -n 8 --mpi=pmi2 --pty python3 allreduce_test.py
array size: 4096
naive impl time: 0.005797386169433594
ring impl time: 0.0008797645568847656
array size: 8192
naive impl time: 0.01948857307434082
ring impl time: 0.0060710906982421875
array size: 16384
naive impl time: 0.0370020866394043
ring impl time: 0.0015218257904052734
```

12. במימוש הנאיבי כל תהליך שולח לכל התהליכים האחרים את כל המידע ולכן כל תהליך שולח s*n מידע כאשר s זה אורך המערך ו- n זה מספר התהליכים. לכן סך הכל הסיבוכיות תהיה $s \cdot n^2$.
13. במימוש הרגיל כל תהליך שולח לכל התהליכים האחרים חלק מהמערך בכל פעם ולא את כולו וסך הכל את כל המערך. ולכן כל תהליך שולח s מידע כאשר s הוא אורך המערך.

    סך הכל יש n תהליכים לכן סך הכל הסיבוכיות תהיה $s \cdot n$.