

HW3 Report

תכנות מקבילי וմבוזר לעיבוד נתונים
236370

מיכל עזרי 325719052
גיא סודאי 214300550

שאלה 1:

:naïve all-reduce

ראשית אתחל מערך $recv$ של כל תהילה להיות המערך $send$ (המיועד לשילחה). עברו בולאה על כל מזהה התהיליכים. לכל מזהה i ,

אם $(i = comm)$. $Get_rank(i)$ הוא מזהה התהילה המריצ את הקוד, אז שדר לכל התהיליכים האחרים את המערך לשילחה.

אחרת, קיבל מטהlixir בעל המזהה $i = source$ את המערך לתוך $temp$ (מערך עז), הפעל על $recv$ את האופרטור op והכנס את התוצאה ל- $temp$.
(לומר, $recv$ מתחילה מהמערך של התהילה ובסך פעם מקבלים מערך מטהlixir אחר, מפעילים עליו עם $recv$ את האופרטור ושומרים ב- $recv$).

:ring all-reduce

$recv \leftarrow send$

1. נחלק את מערך $send$ של כל תהילה למספר $num_of_processes$ חלקים (כדי שנוכל לבצע מעגל צרי' שמספר החלקים של $send$ שווה למספר התהיליכים)

2. $curr \leftarrow comm. Get_Rank()$

3. בולאה על מספר התהיליכים פחות 1:

3.1. שלח את החלק $curr$ לתהילה בעל המזהה העוקב (ציקלית)

3.2. קיבל את החלק $curr - 1$ (ציקלית) מהתהילה בעל המזהה הקודם (ציקלית) לתוך $temp$

3.3. בצע $curr \leftarrow op(temp, send_{curr-1})$ על החלק שעשוי קיבלתי ועל חלק $curr - 1$ (ציקלית)
(שנתנו)

3.4. $curr \leftarrow curr - 1$

4. $curr \leftarrow comm. Get_Rank()$

5. בולאה על מספר התהיליכים פחות 1:

5.1. שלח את החלק $curr$ לתהילה בעל המזהה העוקב (ציקלית)

5.2. קיבל את החלק $curr - 1$ מהתהילה בעל המזהה הקודם (ציקלית)

5.3. $curr \leftarrow curr - 1$ (ציקלית)

העיקרון הוא כפי שראינו בכיתה- התהיליכים נמצאים בمعنى "מעגל" לפי המזהים שלהם, הם מעבירית כל פעם א המידע הרלוונטי לתהילה עם המזהה העוקב ומתקבלים את המידע הרלוונטי מהתהילה עם המזהה הקודם (ציקלית).

בסוף, לפחות אחד מהתהיליכים יש פיסת מידע שלמה אחת, אותה הם יעבירו באופן דומה לכל שאר התהיליכים.

:async network

:worker

1. אתחול מספר batches ליעובד כך שיסתכם $4 * num_of_batches$ על פני כל העובדים.
2. לכל epoch :

2.1. ניצור $mini_batches$

2.2. לכל $mini_batch$

2.2.1. נבצע $nabla_b$, $nabla_w$ - $back_prop$ ו- $forward_prop$

2.2.2. לכל שכבה l נשלח את הגרדי-אנטום $[l][nabla_b[l], nabla_w[l]]$ המזוהה האחראי על l .

הmaster l הוא בעל המזהה $l \bmod num_of_masters$ (לפי

ההגדרה שלנו).

כדי להבדיל בין ההודעות השונות לאוטו master, נשלח עם l tag = l ואות

$[l][nabla_b[l] + num_of_layers * tag]$. זה יבטיח הבדלה בין כל

master אשר שוכנות שונות בעלות אינדקס שונה ומהסיבה שאין חיתוך בין

$[l..l + num_of_layers - 1]$

ההודעות לאוטו

$[0..l - 1]$

2.2.3. לכל שכבה קיבל את החדשים $weights$, $biases$ מהmaster האחראי על השכבה.

שוב, המאסטר האחראי על שכבה l הוא בעל המזהה $l \bmod num_of_masters$. ושוב

להבדיל בין ההודעות השונות מאוטו master, ניעזר בtag שהגדכנו מוקדם עבור שכבה ו-

באותה שכבה. w, b

:master

1. אתחול אינדקסי $layers$ כך שכל masters מכוסים את כל השכבות. לכל master ניתן את

השכבות l המקיימות: $0 \leq l \leq master_rank$.

2. לכל epoch :

2.1. בלולאה על $num_of_batches$:

2.1.1. המtan וקובע index כלשהו (הראשון) שישלח את הגרדי-אנטום w המתאים לשכבה

באינדקס $l = master_rank$ (באמצעות tag = l כפ"י שהגדכנו מוקדם).

2.1.2. שומר את rank של העובד ממנו קיבלנו src .

2.1.3. קיבל את שאר w ו- $biases$ עבור השכבות של המאסטר מהעובד src באמצעות

קונבנציית tag שהגדכנו.

2.1.4. חשב את $weights$, $biases$ החדשים עבור השכבות עליהן אחראי master באמצעות

$.nabla_b$, $nabla_w$

2.1.5. שלח את $weights$, $biases$ לעובד src

3. לבסוף, נשלח את כל הנתונים לmaster עם מזהה 0.

שאלה 2:

הרצנו- ציומי מסך בשאלת 3.

שאלה 3:

4 תהליכיים:

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 4 --pty python3 main.py sync
Epoch 1, accuracy 53.16 %.
Epoch 2, accuracy 87.36 %.
Epoch 3, accuracy 90.47 %.
Epoch 4, accuracy 91.64 %.
Epoch 5, accuracy 92.12 %.
Time reg: 6.463186740875244
Test Accuracy: 91.85%
Epoch 1, accuracy 10.9 %.
Epoch 2, accuracy 51.09 %.
Epoch 3, accuracy 84.23 %.
Epoch 4, accuracy 89.18 %.
Epoch 5, accuracy 91.37 %.
Time sync: 14.583302021026611
Test Accuracy: 90.94%
```

8 תהליכיים:

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 8 --pty python3 main.py sync
Epoch 1, accuracy 51.49 %.
Epoch 2, accuracy 86.96 %.
Epoch 3, accuracy 90.02 %.
Epoch 4, accuracy 91.41 %.
Epoch 5, accuracy 91.85 %.
Time reg: 7.012646913528442
Test Accuracy: 91.16%
Epoch 1, accuracy 10.9 %.
Epoch 2, accuracy 34.43 %.
Epoch 3, accuracy 76.89 %.
Epoch 4, accuracy 87.23 %.
Epoch 5, accuracy 90.19 %.
Time sync: 18.165966510772705
Test Accuracy: 90.07%
```

16 תהליכיים:

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 16 --pty python3 main.py sync
Epoch 1, accuracy 59.03 %.
Epoch 2, accuracy 87.38 %.
Epoch 3, accuracy 90.14 %.
Epoch 4, accuracy 91.1 %.
Epoch 5, accuracy 92.77 %.
Time reg: 10.95359992980957
Test Accuracy: 92.03%
Epoch 1, accuracy 10.3 %.
Epoch 2, accuracy 10.97 %.
Epoch 3, accuracy 41.69 %.
Epoch 4, accuracy 76.73 %.
Epoch 5, accuracy 88.62 %.
Time sync: 67.42281818389893
Test Accuracy: 87.87%
```

נשים לב שככל שמעלים את מספר התהיליכים, זמן הריצה של *synus* עולה משמעותית (בڪב שאינו לינארי). זאת ניתן להסביר מהעובדת שככל שיש יותר תהיליכים, אמןם כל תהיליך מקבל פחות עבודה אבל ניאלץ לבצע יותר תקשורת בין כולם, וכנראה שהוא מופיע על התיעילות *workload* פר תהיליך. ביחס לימוש של הרשות הרגילה, יש פער משמעותי בזמן לטעות הרשות הרגילה, שמתחל בפקטור 2 עבור 4 תהיליכים, פקטורי 2.5 עבור 8 תהיליכים, ומגיע לפקטורי 12.5 (!) עבור 16 תהיליכים. בכל הנוגע לאחיז הדיק של *Test*, לא שיפרנו בשימוש *synus* ואך עבור 16 תהיליכים ניתן לראות הרעה בדיק של 5 אחוזים.

שאלה 4:

השיטה שלפייה קיבל speedup היא חוק אמדהיל. "גודל הבעה" במקורה שלנו הוא גודל הdataset. נחשב מה החלק בתוכנית שאינו ממוקבל. זה יהיה החלק שבו אנו מחשבים מחדש את b , w . לא משנה כמה גודיל את הdataset עדין חלק זה של התוכנית לא ימוקבל. נראה שחלק זה לא נהיה זנich עם גידלת הdataset. החלק שכן ממוקבל הוא חישוב הגרדיינטס על כל batch. גודל החלק זהה (ביחיות של מספר דגימות):

for each epoch :

$$\text{each worker computes : } \#batches \cdot \frac{\text{mini batch size}}{N}$$

$$\implies \text{all workers compute : } \#batches \cdot \frac{\text{mini batch size}}{N} \cdot N = \#batches \cdot \text{mini batch size}$$

נשים לב שחלק הבעה שניתן למקובל תלוי $\#batches$ ובהתאם mini batch size ונשארים קבועים ככל שגדילים את הdataset, החלק הממוקבל יותר קבוע. ולכן A בחוק אמדהיל הוא קבוע. כדי לקבל speedup גם לפי השיטה של גוסטפסון נרצה להתנות את גודל batch שכל עובד עליו לפי גודל הדאטא. בדומה לזה, החלק הממוקבל כן יהיה תלוי בגודל הבעה ונקבל speedup לפי שיטת גוסטפסון.

שאלה 5:

הרצנו- צילומי מסך בשאלת 6.

שאלה 6:

עם 2 ו 8 תהליכי masters

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 4 --pty python3 main.py async 2
Epoch 1, accuracy 58.49 %.
Epoch 2, accuracy 87.36 %.
Epoch 3, accuracy 90.13 %.
Epoch 4, accuracy 91.21 %.
Epoch 5, accuracy 92.15 %.
Time reg: 6.473708629608154
Test Accuracy: 91.64%
Epoch 1, accuracy 16.26 %.
Epoch 2, accuracy 10.9 %.
Epoch 3, accuracy 11.15 %.
Epoch 4, accuracy 10.3 %.
Epoch 5, accuracy 11.34 %.
Time async: 4.193937540054321
Test Accuracy: 92.03%
```

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 8 --pty python3 main.py async 2
Epoch 1, accuracy 51.22 %.
Epoch 2, accuracy 86.82 %.
Epoch 3, accuracy 90.19 %.
Epoch 4, accuracy 91.76 %.
Epoch 5, accuracy 92.21 %.
Time reg: 7.060853958129883
Test Accuracy: 91.63%
Epoch 1, accuracy 10.09 %.
Epoch 2, accuracy 10.09 %.
Epoch 3, accuracy 10.09 %.
Epoch 4, accuracy 10.09 %.
Epoch 5, accuracy 10.09 %.
Time async: 2.745373249053955
Test Accuracy: 10.09%
```

עם 4 ו 8 תהליכי masters

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 8 --pty python3 main.py async 4
Epoch 1, accuracy 57.12 %.
Epoch 2, accuracy 86.1 %.
Epoch 3, accuracy 89.73 %.
Epoch 4, accuracy 91.35 %.
Epoch 5, accuracy 92.6 %.
Time reg: 7.064531564712524
Test Accuracy: 92.1%
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: 3.2245969772338867
Test Accuracy: 87.98%
```

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 16 --pty python3 main.py async 4
Epoch 1, accuracy 60.67 %.
Epoch 2, accuracy 87.22 %.
Epoch 3, accuracy 89.31 %.
Epoch 4, accuracy 91.38 %.
Epoch 5, accuracy 92.33 %.
Time reg: 6.281158447265625
Test Accuracy: 91.66%
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: 8.456143140792847
Test Accuracy: 9.8%
```

ניתן לראות שהצלחנו לשפר את זמן האימון באמצעות השיטה הא-סינכרונית בכל החרצאות פרט לאחת עם 4 מאסטרים ו16 תהליכיים. השיפור נבע מכך שמקבלנו את התהיליך חישוב הגראדיינטים, שהינו חלק כבד ועיקרי באימון, ובאזור א-סינכרונית שמקטינה את תקורת התייאום והתקשרות בהשוואה לסתינכרוני. במקרה עם 16 התהליכיים, כבר קיבלנו כי תקורת התקשרות גדולה עד כדי כך שקיבלנו האטה בביצועים.

ובאשר לתוצאות ה training epochs במהלך ה epochs:

הסיבה שבמהלך הריצה, בשיטה הא-סינכרונית רואים דיווק נמוך לאורך כל התהיליך, היא שדיקון זה מחושב ומודפס ע"י מאסטר 0. שעד לסיוף האימון, בידיו רק השכבות עליו אחראיה, בעוד ששאר השכבות שלו בעלות ערך חסר משמעות שכנן עדכון נעשה ע"י מאסטרים אחרים והතוצאות מאוגדות רק בסופו. لكن בכל שלב בהדפסה יופיע חיזוי לפי הרשות החלקית שלו והතוצאות יהיו חסרות הגיון - דיווק נמוך.

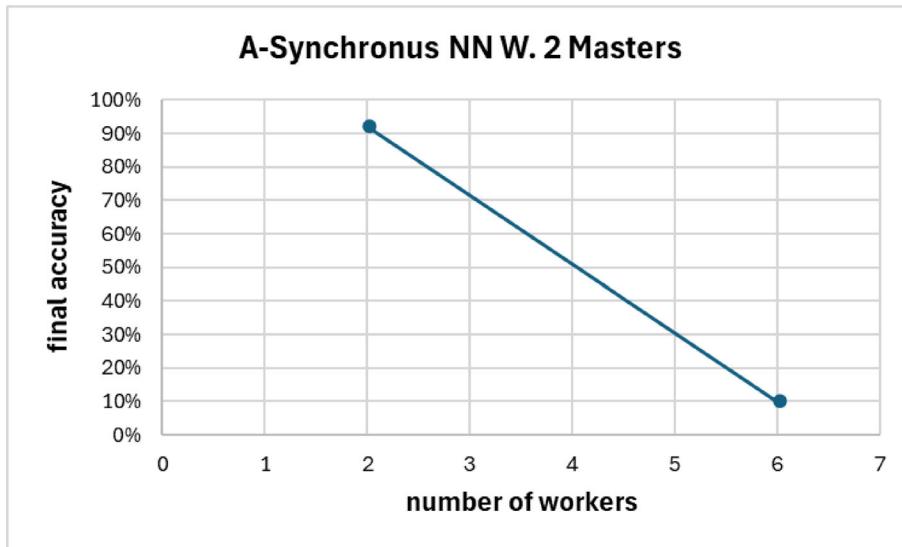
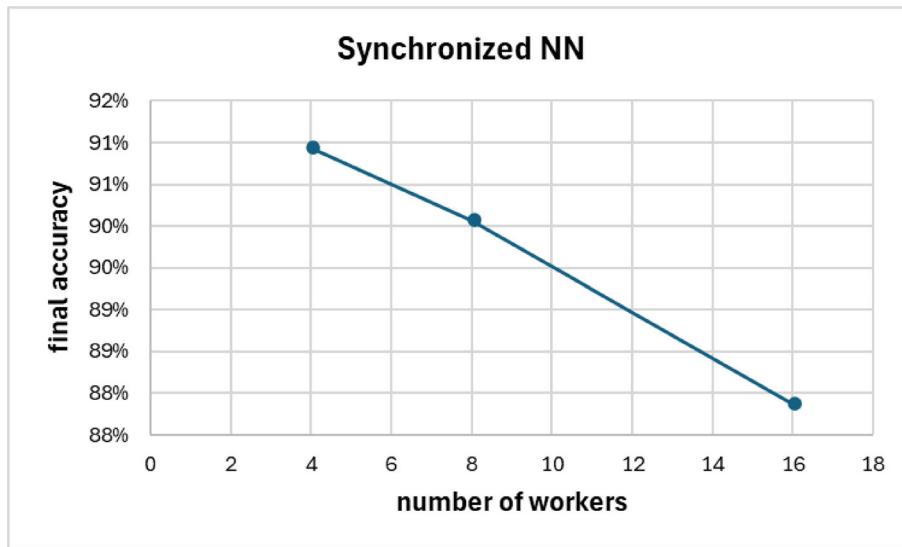
וכן בסופי האימון, בחלק מהחרצאות ראיינו תוצאות טובות, שהשתמשו ברשות המלאה.

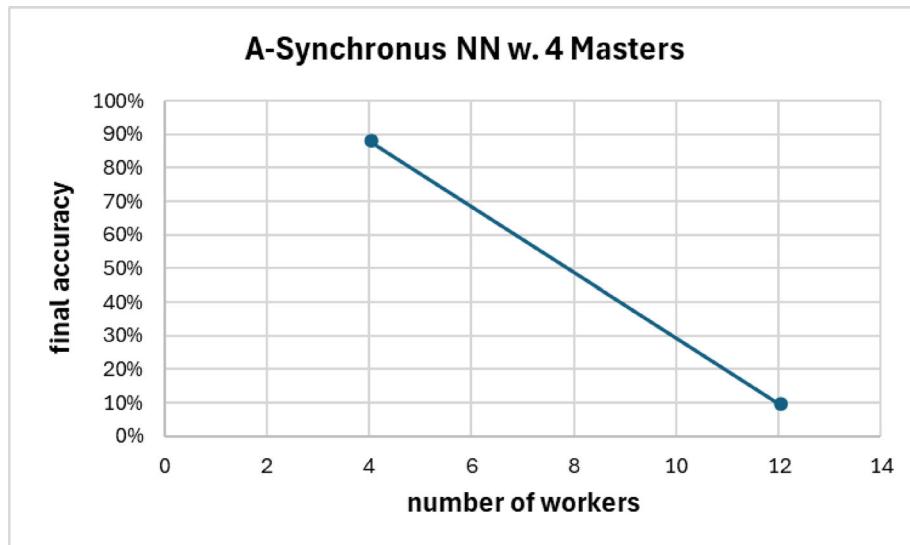
ובחלק ראיינו תוצאות לא טובות, וזאת מוסבר בשאלת 9.

שאלה 7:

בשיטת הא-סינכרונית, לכל המחשב ע"י batch parameter server, workers שלוח את התוצאות ל parameter server כדי לקבל את ה *weights* וה *baises* המעודכנים. אולם ערכים מעודכנים לא מגעים בחינם, על ה parameter server לחשב אותם בהתאם לתוצאות חישוב ה *worker*. **עובדת חישוב זו יכולה להתבצע במקביל**, כיוון שчисוב השכבות השונות לא תלוי וניתן לחלקו לתהליכיים שונים, ובכך נאיץ את קטע חישוב זה. ה *parameter server* מאריך יותר \Leftarrow העובדים ממתינים פחות לפרמטרים חדשים לצורך חישוב האיתרציה הבאה \Leftarrow זמן החישוב הכולל מהיר יותר.

שאלה 8





final accuracy	number of workers
Synchronous NN	
90.94%	4
90.07%	8
87.87%	16
A-Sync NN w. 2 masters	
92.03%	2
10.09%	6
A-Sync NN w. 4 masters	
87.98%	4
9.8%	12

שאלה 9:

ניתן לראות מהתוצאות ההרצאה שלנו, שככל שמספר ה workers גדל, דיקט המודל יורד. הסיבה לכך היא ה *gradient staleness* הגדלה בהתאם למספר הפעלים. נסתכל על worker מסוים, מרגע בו קיבל את הרשות המעודכנת ועד לרגע בו מסיים את החישובים הרלוונטיים עבור הרשות וה batch, פעילים אחרים עשויים לסיים את פעילותם ולשלוח ל parameter server ערכי גרדיאנט חדשניים וכתווצה מכך לעדכן את הרשות. הדבר גורם לכך, שהగרדיאנט שהחישב ה worker שלנו, כבר לא עדכני לרשות, ועדכון הרשות לפי ערכים אלו יהיה בהתאם. ככל שיש יותר פעילים, מספר העדכනים שעשיין לקירות בזמן החישוב של גרדיאנט גדול, וכך גודל ה *gradient staleness* והתכונות המודל בהתאם. הדבר יכול להוביל מעט את הדיקט כפי שראויים אצלנו עבור 4 פעילים, או ממש להרוו את התכונות המודל כפי שראויים עבור 12 פעילים (במקרה של 4 מאסטרים) ו-6 פעילים (במקרה של 2 מאסטרים).

שאלה 10:

נשווה את תוצאות הגישה הסינכרונית והגישה הא-סינכרונית.

מבחןת זמנים, ניתן לראות שהגישה הא-סינכרונית טובה יותר עם ממוצע של $4.65 \text{ sec} \approx$ לעומת הגישה הסינכרונית עם ממוצע של $36.11 \text{ sec} \approx$ בזמן אימון המודל.

מבחינת תוצאות, הגישה הסינכרונית מביאה דיוק טוב ($\approx 90\%$) ללא שונות גבוהה בין הרצות כתלות במספר העובדים. זאת מחסיבה שכארס מסyncrhonous את כל העובדים, כלם מקבלים בכל איטרציה *weights, biases* מעודכנים ונכונים.

לעומת זאת, הגישה הא-סינכרונית מביאה לעיתים דיוק טוב ($\approx 90\%$) ולעתים דיוק לא טוב בכלל ($\approx 10\%$) כתלות במספר העובדים - ככל שיש יותר עובדים, כך הגישה נשברת ווחזק הדיוק קורסים (שאלה 9). זה גורם לגישה זו להיות פחות יעכנית בבדיקה שהיא מספקת כתלות במספר העובדים שהיא משתמשת. אך אם נבחר נכון את מספר העובדים באימון, נוכל להגיע גם לדיק גובה ב-test בזמן נמוך יחסית. **לסיכום:**

Synchronous NN	A-Synchronous NN
זמןנים פחות טובים	זמןנים טובים יותר
דיוק יותר עקבי כתלות במספר העובדים (באופן יחס)	דיוק משתנה דרסטית כתלות במספר העובדים איןנו scalable
מובטחת נכונות. כל עובד יכול לקבל b_i , לא נכונים	לא מובטחת נכונות. כל עובד יכול לקבל b_i , לא נכונים

שאלה 11:

נשווה בין naïve all-reduce ל-ring all reduce לפי מספר התהיליכים. ראשית רואים שבשתי השיטות מקבל אותן תוצאות מה שמבטיח לנו נכונות של השיטות.

נשווה זמנים בין 2 השיטות:

נשים לב כי ring all reduce תמיד מהיר יותר מ-all-reduce naïve.
בשימוש שלנו - לא MPI.sum, אלא ring מהיר יותר בפקטור הנע בין פי 5 ≈ לפי 30 ≈, ובמקרה הממוצע הוא 7 ≈. פער זה לא זניח כל ומייד על יעילות שיטת all reduce מול שיטת all-reduce naïve.

(לא ראיינו לפि התוצאות מוגמת עלייה/ירידה מונוטונית של פקטור השינוי בין 2 השיטות כתלות בגודל המערך)

נשווה זמנים בין מספר התהיליכים:

2 תהיליכים:

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 2 --pty python3 allreduce_test.py
Testing array size: 4096
Naive all-reduce time: 0.0010979175567626953
Ring all-reduce time: 0.00014662742614746094
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 32768
Naive all-reduce time: 0.0021195411682128906
Ring all-reduce time: 0.0003108978271484375
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 262144
Naive all-reduce time: 0.006681680679321289
Ring all-reduce time: 0.0017848014831542969
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 2097152
Naive all-reduce time: 0.03828763961791992
Ring all-reduce time: 0.014960765838623047
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 4096
Naive all-reduce time: 0.00015234947204589844
Ring all-reduce time: 0.0001246929168701172
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True
```

```

Testing array size: 32768
Naive all-reduce time: 0.0005037784576416016
Ring all-reduce time: 0.00020503997802734375
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 262144
Naive all-reduce time: 0.003692626953125
Ring all-reduce time: 0.0017940998077392578
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 2097152
Naive all-reduce time: 0.044982194900512695
Ring all-reduce time: 0.015478849411010742
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

```

Array size	naïve	ring	factor (≈)
4096	0.0010	0.0001	10
32768	0.0021	0.0003	9
262144	0.0066	0.0017	4
2097152	0.0382	0.0149	3

4 תהליכיים:

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 4 --pty python3 allreduce_test.py
Testing array size: 4096
Naive all-reduce time: 0.0005788803100585938
Ring all-reduce time: 0.0001933574676513672
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 32768
Naive all-reduce time: 0.005846738815307617
Ring all-reduce time: 0.0003638267517089844
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 262144
Naive all-reduce time: 0.012033939361572266
Ring all-reduce time: 0.0020868778228759766
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 2097152
Naive all-reduce time: 0.10183095932006836
Ring all-reduce time: 0.02232837677001953
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 4096
Naive all-reduce time: 0.0002751350402832031
Ring all-reduce time: 0.0001468658447265625
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True
```

```
Testing array size: 32768
Naive all-reduce time: 0.0012483596801757812
Ring all-reduce time: 0.0002772808074951172
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 262144
Naive all-reduce time: 0.008581161499023438
Ring all-reduce time: 0.0020284652709960938
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 2097152
Naive all-reduce time: 0.1069493293762207
Ring all-reduce time: 0.02215743064880371
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True
```

Array size	naïve	ring	factor (≈)
------------	-------	------	--------------

4096	0.0005	0.0001	5
32768	0.0058	0.0003	20
262144	0.0120	0.0020	5
2097152	0.1018	0.0223	5

8 תהליכי:

```
(tf23-gpu) michal.ozeri@lambda:~/hw3$ srun -K -c 2 -n 8 --pty python3 allreduce_test.py
Testing array size: 4096
Naive all-reduce time: 0.0012192726135253906
Ring all-reduce time: 0.0003097057342529297
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 32768
Naive all-reduce time: 0.009556293487548828
Ring all-reduce time: 0.0003950958557128906
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 262144
Naive all-reduce time: 0.03161764144897461
Ring all-reduce time: 0.003515005111694336
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 2097152
Naive all-reduce time: 0.3095259666442871
Ring all-reduce time: 0.041742563247680664
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 4096
Naive all-reduce time: 0.0007710456848144531
Ring all-reduce time: 0.00024008750915527344
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True
```

```

Testing array size: 32768
Naive all-reduce time: 0.0041849613189697266
Ring all-reduce time: 0.0005135536193847656
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 262144
Naive all-reduce time: 0.029738664627075195
Ring all-reduce time: 0.0036401748657226562
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

Testing array size: 2097152
Naive all-reduce time: 0.3247535228729248
Ring all-reduce time: 0.04028773307800293
Comparing results...
Naive all-reduce correct: True
Ring all-reduce correct: True

```

Array size	naïve	ring	factor (≈)
4096	0.0012	0.0003	4
32768	0.0095	0.0003	30
262144	0.0316	0.0035	10
2097152	0.3095	0.0417	8

שאלה 12:

נחשב את סיבוכיות *naive allreduce*

נסמן ב a את מספר האיברים המערך שיש לאמת בין התהליים, וב k את מספר התהליים המערכת. נספור כמה ערכים שולח כל תהלייר, וכמה נשלחים סה"כ בכל התהליים יחד.

תהלייר בודד:

בשיטת הנאיית, התהלייר שולח את המידע שברשותו (שגודלו a) לכל אחד מהטהליים האחרים ($1 - k$ כלו).
ולכן שולח $(1 - k) \cdot a$ ערכים סה"כ.

כל התהליים יחד:

ישנם k תהליים, שכל אחד מהם שולח ערכים כפי שהישבנו קודם, ולכן מספר השליחות הכלול: $k(1 - k) \cdot a$.

ובמונחים חסמים וסיבוכיות: $\mathcal{O}(nk^2)$.

שאלה 13:

נחשב את סיבוכיות *ring allreduce*

נסמן ב n את מספר האיברים המערך שיש למתאים בין התהיליכים, וב k את מספר התהיליכים המערך. נספור כמה ערכים שולח כל תהיליך, וככמה נשלחים סה"כ בכל התהיליכים יחד.

תחילה, נזכיר שבשיטה זו המערך מחולק באופן שווה ל k מקטעים (שגודלם $\frac{n}{k}$), ואז:

שלב 1: $1 - k$ פעמים, כל תהיליך שולח את המקטע המתאים לו ל תהיליך הבא אחריו, ומתקדם למקטע הבא.

שלב 2: לכל תהיליך יש כעט מקטע מעודכן כלשהו, יש להעביר באופן דומה לשלב אחד את המידע לתהיליכים הבאים.

תהליך בודד:

בשלב הראשון יבוצעו $1 - k$ שליחות של מקטעים, כמשמעותו כל מקטע $\frac{n}{k}$. וכך:

בשלב השני, שוב, יבוצעו $1 - k$ שליחות של מקטעים, כמשמעותו כל מקטע $\frac{n}{k}$. וכך:

$$\text{ו如此: } 2 \cdot \left(n - \frac{n}{k} \right)$$

כל התהיליכים יחד:

ישנו k תהיליכים, שכל אחד מהם שולח ערכים כפי שהסבירנו קודם, וכך:

$$2k \cdot \left(n - \frac{n}{k} \right)$$

ובמונחים חסמים וסיבוכיות: $O(nk)$.

$$\left(2k \cdot \left(n - \frac{n}{k} \right) \leq 2kn \implies O(nk) \right)$$

