

HW3 Report

תכנות מקבילי ומבוזר לעיבוד נתונים
236370

מיכל עוזרי 325719052
גיא סודאי 214300550

שאלה 1:

naïve all-reduce:

ראשית אתחל מערך $recv$ של כל תהליך להיות המערך $send$ (המיועד לשליחה). עבור כלולאה על כל מזהי התהליכים. לכל מזהה i , אם $i = comm.Get_rank()$ (כלומר, i הוא מזהה התהליך המריץ את הקוד), אז שדר לכל התהליכים האחרים את המערך לשליחה. אחרת, קבל מתהליך בעל המזהה i ($= source$) את המערך לתוך $temp$ (מערך עזר), הפעל על $recv, temp$ את האופרטור op והכנס את התוצאה ל- $recv$. (כלומר, $recv$ מתחיל מהמערך של התהליך ובכל פעם שמקבלים מערך מתהליך אחר, מפעילים עליו עם $recv$ את האופרטור ושומרים ב- $recv$).

ring all-reduce:

אתחול- $recv \leftarrow send$

1. נחלק את מערך $send$ של כל תהליך ל- $num_of_processes$ חלקים (כדי שנוכל לבצע מעגל צריך שמספר החלקים של $send$ שווה למספר התהליכים)
2. $curr \leftarrow comm.Get_Rank()$
3. כלולאה על מספר התהליכים פחות 1:
 - 3.1. שלח את החלק $curr$ לתהליך בעל המזהה העוקב (ציקלית)
 - 3.2. קבל את החלק ה- $curr - 1$ (ציקלית) מהתהליך בעל המזהה הקודם (ציקלית) לתוך $temp$
 - 3.3. בצע $recv \leftarrow op(temp, send_{curr-1})$ (כלומר, op על החלק שעכשיו קיבלתי ועל חלק $curr - 1$ שנתון)
 - 3.4. $curr \leftarrow curr - 1$ (ציקלית)

4. $curr \leftarrow comm.Get_Rank() + 1$ (ציקלי)

5. כלולאה על מספר התהליכים פחות 1:

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

5.2. קבל את החלק $curr - 1$ מהתהליך בעל המזהה הקודם (ציקלית)

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

העיקרון הוא כפי שראינו בכיתה- התהליכים נמצאים במעין "מעגל" לפי המזהים שלהם, הם מעבירים כל פעם א המידע הרלוונטי להתהליך עם המזהה העוקב ומקבלים את המידע הרלוונטי מהתהליך עם המזהה הקודם (ציקלית). בסוף, לכל אחד מהתהליכים יש פיסת מידע שלמה אחת, אותה הם יעבירו באופן דומה לכל שאר התהליכים.

:async network

:worker

1. אתחול מספר batches לעובד כך שיתכנס ל- $num_of_batches$ על פני כל העובדים.

2. לכל epoch:

2.1. ניצור $mini_batches$

2.2. לכל $mini_batch$:

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

2.2.2. לכל שכבה l נשלח את הגרדיאנטים $nabla_b[l], nabla_w[l]$ ל-master האחראי על l .

ה-master האחראי על l הוא בעל המזהה $l \bmod num_of_masters$ (לפי

ההגדרה שלנו).

כדי להבדיל בין ההודעות השונות לאותו master, את $nabla_w[l]$ נשלח עם $tag = l$ ואת

$nabla_b[l]$ נשלח עם $tag = l + num_of_layers$. זה יבטיח הבדלה בין כל

ההודעות לאותו master שכן שכבות שונות בעלות אינדקס שונה ומהסיבה שאין חיתוך בין

ההודעות לאותו

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

$[0..l - 1]$ ל-

2.2.3. לכל שכבה נקבל את ה- $weights, biases$ החדשים מהמאסטר האחראי על השכבה.

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

להבדיל בין ההודעות השונות מאותו master, ניעזר ב- tag שהגדרנו מקודם עבור שכבה ו-

כדי

באותה שכבה.

w, b

:master

1. אתחול אינדקסי ה- $layers$ ל-master כך שכל masters מכסים את כל השכבות. לכל master ניתן את

השכבות l המקיימות: $l \bmod master_rank = 0$.

2. לכל epoch:

2.1. בלולאה על $num_of_batches$:

2.1.1. המתן וקבל מעובד כלשהו (הראשון) שישלח את הגרדיאנט $nabla_w$ המתאים לשכבה

$master_rank$ (באמצעות $tag = l$ כפי שהגדרנו מקודם).

באינדקס

2.1.2. שמור את ה- $rank$ של העובד ממנו קיבלנו ב- src .

2.1.3. קבל את שאר $nabla_b, nabla_w$ עבור השכבות של המאסטר מהעובד src באמצעות

קונבנציית ה- tag שהגדרנו.

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

$nabla_b, nabla_w$.

2.1.5. שלח את ה- $weights, biases$ החדשים לעובד src .

3. לבסוף, נשלח את כל הנתונים ל-master עם מזהה 0.

שאלה 2:

הרצנו- צילומי מסך בשאלה 3.

שאלה 3:

4 תהליכים:

```
(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 תהליכים:

```
(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 תהליכים:

```
(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 התהליכים, זמן הריצה של *sync* עולה משמעותית (בקצב שאינו לינארי). זאת ניתן להסביר מהעובדה שככל שיש יותר תהליכים, אמנם כל תהליך מקבל פחות עבודה אבל ניאלץ לבצע יותר תקשורת בין כולם, וכנראה שזה מעפיל על התייעלות ב *workload* פר תהליך. ביחס למימוש של הרשת הרגילה, יש פער משמעותי בזמנים לטובת הרשת הרגילה, שמתחיל בפקטור 2 עבור 4 תהליכים, פקטור 2.5 עבור 8 תהליכים, ומגיע לפקטור 12.5 (!) עבור 16 תהליכים. בכל הנוגע לאחוז הדיוק של *Test*, לא שיפרנו בשימוש ב *sync* ואף עבור 16 תהליכים ניתן לראות הרעה בדיוק של 5 אחוזים.

שאלה 4:

השיטה שלפיה נקבל $speedup$ היא חוק אמדהל. "גודל הבעיה" במקרה שלנו הוא גודל הdataset. נחשוב מהו החלק בתוכנית שאינו ממוקבל. זה יהיה החלק שבו אנו מחשבים מחדש את w, b . לא משנה כמה נגדיל את הdataset עדיין חלק זה של התוכנית לא ימוקבל. נראה שחלק זה לא נהיה זניח עם גדילת הdataset. החלק שכן ממוקבל הוא חישוב הגרדיאנטים על כל batch. גודל החלק הזה (ביחידות של מספר דגימות):

for each epoch :

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

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

נשים לב שחלק הבעיה שניתן למקבול תלוי ב $num_of_batches$ ו- $mini\ batch\ size$ שבהנחה והם נשארים קבועים ככל שמגדילים את הdataset, החלק הממוקבל נותר קבוע. ולכן A בחוק אמדהל הוא קבוע. כדי לקבל $speedup$ גם לפי השיטה של גוסטפסון נרצה להתנות את גודל הbatch שכל עובד עובד עליו לפי גודל הדאטא. בצורה כזו, החלק הממוקבל כן יהיה תלוי בגודל הבעיה ונקבל $speedup$ לפי שיטת גוסטפסון.

שאלה 5:

הרצנו- צילומי מסך בשאלה 6.

שאלה 6:

עם 2 masters ו-4/8 תהליכים:

```
(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 masters ו-8/16 תהליכים:

```
(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 ההרצות פרט לאחת עם 4 מאסטרים ו16 תהליכים. השיפור נבע מכך שמקבלנו את תהליך חישוב הגראדינטים, שהינו חלק כבד ועיקרי באימון, ובצורה א-סינכרונית שמקטינה את תקורת התיאום והתקשורת בהשוואה לסינכרוני. במקרה עם 16 התהליכים, כבר קיבלנו כי תקורת התקשורת גדולה עד כדי כך שקיבלנו האטה בביצועים.

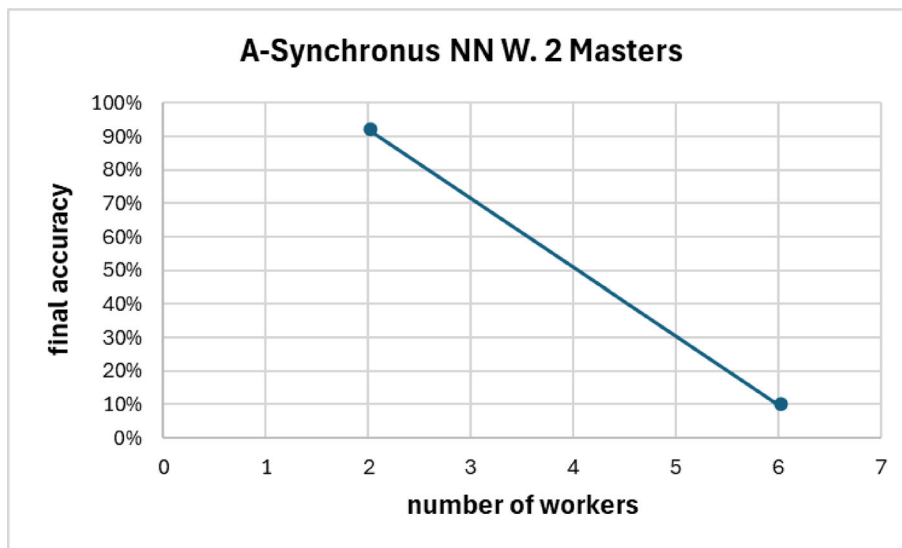
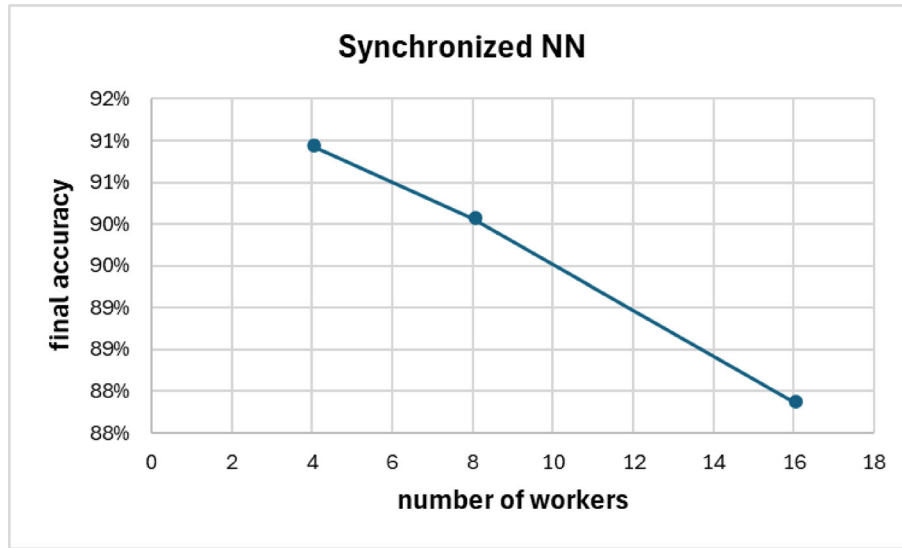
ובאשר לתוצאות ה training במהלך ה epochs:

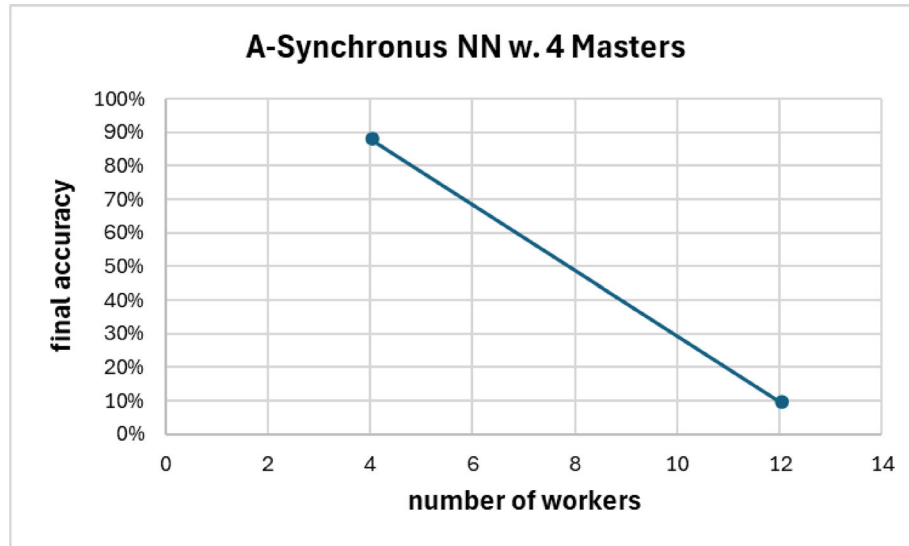
הסיבה שבמהלך הריצה, בשיטה הא-סינכרונית רואים דיוק נמוך לאורך כל התהליך, היא שדיוק זה מחושב ומודפס ע"י מאסטר 0. שעד לסוף האימון, בידיו רק השכבות עליו אחראי, בעוד ששאר השכבות שלו בעלות ערך חסר משמעות שכן עדכון נעשה ע"י מאסטרים אחרים והתוצאות מאוגדות רק בסוף. לכן בכל שלב בהדפסה ידפיס חיזוי לפי הרשת החלקית שלו והתוצאות יהיו חסרות הגיון - דיוק נמוך. וכן בסוף האימון, בחלק מההרצות ראינו תוצאות טובות, שהשתמשו ברשת המלאה. ובחלק ראינו תוצאות לא טובות, וזאת מוסבר בשאלה 9.

שאלה 7:

בשיטה הא-סינכרונית, לכל batch המחושב ע"י worker, הworkers שולח את התוצאות ל parameter server כדי לקבל את הweights והbaises המעודכנים. אותם ערכים מעודכנים לא מגיעים בחינם, על ה parameter server לחשב אותם בהתאמה לתוצאות חישוב ה worker. **עבודת חישוב זו יכולה להתבצע במקביל**, כיוון שחישוב השכבות השונות לא תלוי וניתן לחלקן לתהליכים שונים, ובכך נאיץ את קטע חישוב זה. parameter server מהיר יותר \Leftarrow העובדים ממתינים פחות לפרמטרים חדשים לצורך חישוב האיטרציה הבאה \Leftarrow זמן החישוב הכולל מהיר יותר.

שאלה 8:





final accuracy	number of workers
Synchronus 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 מתוצאות ההרצה שלנו, שככל שמספר ה workers גדל, דיוק המודל יורד. הסיבה לכך היא ה $gradient staleness$ הגדלה בהתאמה למספר הפועלים. נסתכל על worker מסויים, מרגע בו קיבל את הרשת המעודכנת ועד לרגע בו מסיים את החישובים הרלוונטים עבור הרשת וה batch, פועלים אחרים עשויים לסיים את פעילותם ולשלוח ל parameter server ערכי גרדיאנט חדשים וכתוצאה מכך לעדכן את הרשת. הדבר גורם לכך, שהגרדיאנט שחישב ה worker שלנו, כבר לא עדכני לרשת, ועדכון הרשת לפי ערכים אלו יהיה בהתאם. ככל שיש יותר פועלים, מספר העדכונים שעשוי לקרות בזמן החישוב של גרדיאנט גדל, וכך גדל ה $gradient staleness$ והתכנסות המודל בהתאמה. הדבר יכול להוריד מעט את הדיוק כפי שרואים אצלנו עבור 4 פועלים, או ממש להרוס את התכנסות המודל כפי שרואים עבור 12 פועלים (במקרה של 4 מאסטרים) ו-6 פועלים (במקרה של 2 מאסטרים).

שאלה 10:

נשווה את תוצאות הגישה הסינכרונית והגישה הא-סינכרונית. מבחינת זמנים, ניתן לראות שהגישה הא-סינכרונית טובה יותר עם ממוצע של $4.65 \text{ sec} \approx$ לעומת הגישה הסינכרונית עם ממוצע של $36.11 \text{ sec} \approx$ לזמן אימון המודל. מבחינת תוצאות, הגישה הסינכרונית מביאה דיוק טוב ($90\% \approx$) ללא שונות גבוהה בין הרצות כתלות במספר העובדים. זאת מהסיבה שכאשר מסנכרנים את כל העובדים, כולם מקבלים בכל איטרציה $weights, biases$ מעודכנים ונכונים. לעומת זאת, הגישה הא-סינכרונית מביאה לעיתים דיוק טוב ($90\% \approx$) ולעיתים דיוק לא טוב בכלל ($10\% \approx$) כתלות במספר העובדים - ככל שיש יותר עובדים, כך הגישה נשברת ואחוזי הדיוק קורסים (שאלה 9). זה גורם לגישה זו להיות פחות עקבית בדיוק שהיא מספקת כתלות במספר העובדים שהיא משתמשת. אך אם נבחר נכון את מספר העובדים באימון, נוכל להגיע גם לדיוק גבוה ב $test$ בזמן נמוך יחסית. לסיכום:

Synchronus NN	A-Synchronus NN
זמנים פחות טובים	זמנים טובים יותר
דיוק יותר עקבי כתלות במספר העובדים (באופן יחסי)	דיוק משתנה דרסטית כתלות במספר העובדים אינו scalable
מובטחת נכונות מסוימת	לא מובטחת נכונות. כל עובד יכול לקבל w, b לא נכונים

שאלה 11:

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

נשווה זמנים בין 2 השיטות:

נשים לב כי ring all reduce תמיד מהיר יותר מ-naïve all-reduce.
במימוש שלנו- ללא MPI.sum, הring מהיר יותר בפקטור הנע בין פי 5 \approx לפי 30 \approx , ובמקרה הממוצע הוא 7 \approx . פער זה לא זניח כלל ומעיד על יעילות שיטת ring all reduce אל מול שיטת naïve all-reduce.
(לא ראינו לפי התוצאות מגמת עליה/ ירידה מונוטונית של פקטור השינוי בין 2 השיטות כתלות בגודל המערך)

נשווה זמנים בין מספר התהליכים:

2 תהליכים:

```
(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 (\approx)
4096	0.0010	0.0001	10
32768	0.0021	0.0003	9
262144	0.0066	0.0017	4
2097152	0.0382	0.0149	3

```

(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 (\approx)
------------	-------	------	----------------------

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 תהליכים:

```
(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: 0.00039505958557128906
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 (\approx)
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 n את מספר האיברים המערך שיש לתאם בין התהליכים, וב k את מספר התהליכים המערכת. נספור כמה ערכים שולח כל תהליך, וכמה נשלחים סה"כ בכל התהליכים יחד.

תהליך בודד:

בשיטה הנאיבית, התהליך שולח את המידע שברשותו (שגודלו n) לכל אחד מהתהליכים האחרים $k - 1$ (כאלו).

ולכן שולח $(k - 1) \cdot n$ ערכים סה"כ.

כל התהליכים יחד:

ישנם k תהליכים, שכל אחד מהם שולח ערכים כפי שחישבנו קודם, ולכן מספר השליחות הכולל: $k \cdot (k - 1) \cdot n$.

ובמונחים חסמים וסיבוכיות: $O(nk^2)$.

שאלה 13:

נחשב את סיבוכיות $ring\ allreduce$

נסמן ב n את מספר האיברים המערך שיש לתאם בין התהליכים, וב k את מספר התהליכים המערכת. נספור כמה ערכים שולח כל תהליך, וכמה נשלחים סה"כ בכל התהליכים יחד.

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

שלב 1: $k - 1$ פעמים, כל תהליך שולח את המקטע המתאים לו לתהליך הבא אחריו, ומתקדם למקטע הבא.
שלב 2: לכל תהליך יש כעת מקטע מעודכן כלשהו, יש להעביר באופן דומה לשלב אחד את המידע לתהליכים הבאים.

תהליך בודד:

בשלב הראשון יבוצעו $k - 1$ שליחות של מקטעים, כשגודל כל מקטע $\frac{n}{k}$. ולכן $\frac{n}{k}(k - 1) = n - \frac{n}{k}$ שליחות.
בשלב השני, שוב, יבוצעו $k - 1$ שליחות של מקטעים, כשגודל כל מקטע $\frac{n}{k}$. ולכן $\frac{n}{k}(k - 1) = n - \frac{n}{k}$ שליחות.

$$\text{סה"כ } 2 \cdot \left(n - \frac{n}{k} \right)$$

כל התהליכים יחד:

ישנם k תהליכים, שכל אחד מהם שולח ערכים כפי שחישבנו קודם, ולכן מספר השליחות הכולל:

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

ובמונחים חסמים וסיבוכיות: $O(nk)$.

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

