# תכנות מקבילי ומבוזר- תרגיל בית 1

איתי לביא 212147326 והראל ימין 213099492

1. פונקציית ה-max_kernel שלנו מקבלת 2 מטריצות A ו-B ופועלת על 1000 בלוקים כך שכל בלוק מכיל 1000 ת'רדים.
הפונקצייה מעדכנת את התוצאה במטריצה C שיצרנו. (צורתה שווה לצורה של B,A)
כל ת'רד j בבלוק ה-i אחראי על עדכון התא ה-[i,j] להיות $\max\{A_{i,j}, B_{i,j}\}$.
מפני שלא ייתכנו 2 חוטים עם אותו בלוק ID ו-thread id, החוטים אינם תלויים זה בזה.
כשכל החוטים יסיימו לרוץ, נעתיק חזרה את המטריצה C מהזיכרון של ה-GPU ל-CPU ונחזיר אותה.

זמני הריצה שהתקבלו מהרצת max_functions.py:

```
(tf23-gpu) itaylavi@lambda:~/CDPDML/hw1$ srun --gres=gpu:1 -c 1 --pty python3 max_functions.py
    [*] CPU: 18.344823117367923
    [*] Numba: 0.03030107542872429
    [*] CUDA: 0.18186597991734743
```

ניתן לראות שההרצות על ה-CPU עם ובלי Numba הן בסדרי גודל שונים, מפני שהפעולה החישובית על הקלט עצמו היא זניחה לעומת התקורה שלוקחות הפעולות בשפת פייתון.
Numba משתמשת באופטימיזציות כך שהקוד מתורגם לשפה נמוכה יותר וגם משתמשת במקביליות בעזרת חוטים שונים, לכן נקבל שNumba עובדת מהר יותר.
בנוסף, נשים לב שמשום שאנחנו מבצעים פעולה יחסית קצרה, רוב הזמן ב-GPU מבוזבז על תקורת ההעתקות של המטריצות A,B,C והעתקת C חזרה ל-CPU.
לכן הגיוני שהרצה ב-Cuda תיקח זמן רב יותר מהרצה בעזרת Numba.

ה-speedup שהתקבל מ-Numba:

$$numba\_speedup = \frac{CPU_{TIME}}{NUMBA_{TIME}} \approx \frac{18.335}{0.03} = 611.166$$

ה-speedup שהתקבל מ-Cuda:

$$gpu\_speedup = \frac{CPU_{TIME}}{GPU_{TIME}} \approx \frac{18.335}{0.182} = 100.741$$

2. ראשית, לצורך נוחות פרשנו את המטריצה שגודלה $(row, row)$ להיות וקטור שאורכו $row^2$.
חילקנו את הוקטור לבלוקים של 1024 איברים רצופים, כך שכל חוט יעדכן איבר אחד בכל בלוק, לפי ה-ID שלו.
לדוגמה:
   a. חוט מס' 0 יעדכן את האיברים 0,1024,2048 וכך הלאה בוקטור.
   b. חוט מס' 1 יעדכן את האיברים 1,1025,2049 וכך הלאה בוקטור.
   וכו'.
משום שהמטריצה $X \cdot X^T$ היא מטריצה סימטרית (מטריצת גראם), מספיק לנו לחשב רק את המטריצה המשולשת התחתונה. כלומר כאשר $i < j$ לא נבצע כלום.
עבור כל איבר שנחשב, נעדכן את התוצאה ב2 המקומות המתאימים במטריצת התוצאה, גם כאשר אנחנו נמצאים על האלכסון הראשי כדי לחסוך בתנאים.
נעצור כל חוט כאשר הוא עומד לעדכן איבר שהאינדקס שלו מחוץ לגבולות הוקטור.
לבסוף, נשנה את התוצאה מוקטור שאורכו $row^2$ להיות המטריצה $(row, row)$ כפי שנדרש.

3. זמני הריצה שהתקבלו מהרצת matmul_functions.py:

```
(base) harely@lambda:~$ srun --gres=gpu:1 -c 1 --pty python3 matmul_functions.py
Numpy: 0.4098039949312806
Numba: 6.89081802405417
/home/harely/miniconda3/lib/python3.9/site-packages/numba/cuda/compiler.py:726: NumbaPerformanceWarn
tilization due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
/home/harely/miniconda3/lib/python3.9/site-packages/numba/cuda/compiler.py:726: NumbaPerformanceWarn
tilization due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
CUDA: 2.8439395325258374
```

כפי שניתן לראות, numpy הכי יעיל מבין 3 ההרצות. הספרייה מכילה אופטימיזציות שונות לפעולות מתמטיות וכתובה בשפה יותר נמוכה. לכן כפי שציפינו numpy עובד הכי מהר.
Numba אמנם משפר את זמני הריצה בהרצה על הCPU ומשתמש בקוד מקבילי, אך לא ניתן להשוות זאת לכוח החישובי שקיים ב-GPU והריצה עליו.
נשים לב שהשימוש ב-Cuda מאלץ גם תקורה של העתקת המטריצות A,B,C לזיכרון ה-GPU והעתקה של התוצאה חזרה ל-CPU ולכן זה מרחיק אותו עוד יותר מזמני הריצה של numpy.