

# CDP – HW1

## Part 2

להלן השוואה בין זמני הריצה של פונקציית ה-max על ה-CPU, GPU ובאופן מקבילי על ידי: NUMBA:

```
(tf23-gpu) uri.kasher@lambda:~/CDP-236370/hw1_cdp$ srun --gres=gpu:1 -c 1 --pty python3 max_functions.py
[+] max_cpu passed
[+] max_numba passed
[+] max_gpu passed
[+] All tests passed

[*] CPU: 14.682468060404062
[*] Numba: 0.028537258505821228
[*] CUDA: 0.14293109625577927
```

### הסבר עבור מימוש ל-GPU:

כנדרש, חילקנו את העבודה על המטריצות ל-1000 בלוקים כאשר לכל בלוק 1000 threads. בהתאם, כל thread מבצע עבודה המתאימה לבדיקת אחד במטריצה בגודל  $1000 \times 1000$ . נתאים לכל thread את התא המתאים לו באופן הבא:

```
1 @cuda.jit
2 def max_kernel(A, B, C):
3     """
4         Find the maximum value in values and store in result[0]
5     """
6     idx = cuda.blockIdx.x * cuda.blockDim.x + cuda.threadIdx.x
7
8     if idx < A.shape[0] * A.shape[1]:
9         row = idx // A.shape[1]
10        col = idx % A.shape[1]
11
12        C[row, col] = max(A[row, col], B[row, col])
```

כמתואר בשורה 6 לכל thread  $0 \leq t \leq 999$  השייך לבלוק  $0 \leq b \leq 999$  כלשהו, נתאים אינדקס:

$$i = 1000 \cdot b + t$$

באופן זה נוכל להתייחס לכל thread כאילו מתייחס לאינדקס במערך באורך  $10^6$ .

כעת נרצה לבצע חישוב רק באינדקסים המתאימים למימדי המטריצות (שורה 8). נתאים את מספר השורה על ידי חלוקת האינדקס במספר העמודות של A (הזהב ממדיה למטריצות B, C) ואת מספר העמודה על ידי חישוב האינדקס מודולו מספר העמודות של A – שורות 9, 10.

לבסוף נחשב את המקסימום עבור התא המתאים ב-A וב-B ונעדכן את C בהתאם (שורה 12).

### הסבר התוצאות:

הצלחנו להריץ רק עבור ליבה אחת או שתיים לכל היותר ולכן נסביר תוך שיערוך השפעת מספר הליבות על תוצאות הריצה.

- CPU: מכיוון ש-max\_cpu אינה משתמשת בתכנות מקבילי לא נצפה לראות השפעה על זמן הריצה אם נסיף ליבות.

- Numba: max\_numba משתמשת בתכנות מקבילי. בהתאם לכך, צפה שהוספת ליבות תוריד את זמן הריצה של פונקציה זו. אף על פי כן, מכיוון שהפונקציה מבצעת חישוב פשוט וקל מאוד, לא צפה לשינוי משמעותי – החלפות ההקשר יקחו זמן הולך וגדל ביחס למשך הפעולה של התוכנית והמשקל החישובי יעבור להחלפות ההקשר ולא לפעולת החישוב עצמה.
- GPU: ה-GPU כלל אינו תלוי במספר הליבות. המעבד אכן אחראי על ניהול הריצה של ה-GPU אך הדבר לא צפוי לקצר את זמן החישוב עצמו.

עבור שתי ליבות קיבלנו תוצאה דומה מאוד לליבה אחת (ככל הנראה מהנימוקים הנ"ל):

```
(tf23-gpu) uri.kasher@lambda:~/CDP-236370/hw1_cdp$ srun --gres=gpu:1 -c 2 --pty python3 max_functions.py
[+] max_cpu passed
[+] max_numba passed
[+] max_gpu passed
[+] All tests passed

[*] CPU: 14.560555268079042
[*] Numba: 0.030139360576868057
[*] CUDA: 0.14977407082915306
(tf23-gpu) uri.kasher@lambda:~/CDP-236370/hw1_cdp$
```

Speedup (עבור ליבה אחת):

$$\frac{T_{max-gpu}}{T_{max-cpu}} \approx \frac{0.143}{14.682} = 0.0097 \Rightarrow Speedup \approx 1000\%$$

$$\frac{T_{max-gpu}}{T_{max-numba}} \approx \frac{0.143}{0.028} = 5.107 \Rightarrow Speedup \approx 20\%$$

## Part 3

### matmul\_kernel

```
1 @cuda.jit
2 def matmul_kernel(A, C):
3     tx = cuda.threadIdx.x
4
5     for idx in range(tx, C.shape[0] * C.shape[1], 1024):
6         row = idx // C.shape[1]
7         col = idx % C.shape[1]
8
9         s = 0
10        for k in range(A.shape[1]):
11            s += A[row, k] * A[col, k]
12        C[row, col] = s
```

יש לנו 1024 חוטים ואנחנו צריכים לחשב את התוצאה של הכניסות של המטריצה.  $C = X \cdot X^T$  - לכן כל חוט יפעל על  $\frac{n^2}{1024}$  תאים. החישוב של תא הוא  $C_{i,j} = \sum_{k=1}^n X_{i,k} \cdot X_{j,k}$  (מתבצע בשורות 9 עד 12) כלומר כל חוט צריך לבצע לולאה אחת לכל תא שהוא אחראי לחשב את התוצאה עבורו. את החלוקה אנחנו עושים על ידי שימוש באינדקס של החוט, מתבצע בשורה 5. ההמרה מהאינדקס של החוט לאינדקס של איבר במטריצה מתבצע בשורות 6 ו-7.

### תוצאות ההרצה:

```
(tf23-gpu) uri.kasher@lambda:~/CDP-236370/hw1_cdp$ srun --gres=gpu:1 -c 1 --pty python3 matmul_functions.py
[+] matmul_transpose_trivial passed
[+] matmul_transpose_numba passed
/home/uri.kasher/miniconda3/envs/tf23-gpu/lib/python3.8/site-packages/numba/cuda/compiler.py:726: NumbaPerformanceWarning(msg)
warn(NumbaPerformanceWarning(msg))
[+] matmul_transpose_gpu passed
[+] All tests passed

Numpy: 0.4238302782177925
Numba: 6.9528176337480545
/home/uri.kasher/miniconda3/envs/tf23-gpu/lib/python3.8/site-packages/numba/cuda/compiler.py:726: NumbaPerformanceWarning(msg)
warn(NumbaPerformanceWarning(msg))
CUDA: 5.762527622282505
```

כיוון שאנחנו מוגבלים לשימוש ב-1024 חוטים בלבד, אנחנו לא מנצלים את ה-GPU למלוא היכולת שלו ולכן השיפור לא יהיה משמעותי כמו ב-`max_kernel`, במיוחד כאשר לוקחים בחשבון את התקורה של העברת נתונים מה-CPU ל-GPU וחזרה. בכל זאת כן רואים שיפור מהתוצאות של ה-Numba.