תרגיל בית 3 מחשוב מקבילי סמסטר קיץ 2020

בתרגיל יש שילוב של MPI, OpenMP ו- Cuda.

יש לכתוב תכנית שמחשבת היסטוגרמה (histogram) של ערכים המופיעים בקלט. כל ערך הוא מספר שלם בין 1 ל- 256. במילים אחרות, עבור כל ערך יש לחשב את מספר המופעים שלו בקלט. לדוגמא אם הקלט הוא

10 10 3 10 150 150 180 3 150 10 אז הפלט יהיה

```
3: 2
10: 4
150: 3
180: 1
```

כלומר המספר 3 מופיע בקלט פעמיים, המספר 10 מופיע 4 פעמים וכן הלאה. ניתן להשמיט מהפלט מספרים שאינם מופיעים בקלט.

השורה הראשונה בקלט כוללת מספר N ובהמשך מופיעים N מספרים (בטווח 1-256) המופרדים ע"י whitespace (רווחים, טאבים או newlines).

התוכנית תקרא את הקלט מה- standard input. היא תכתוב את הפלט ל- standard output.

מבנה התוכנית.

יהיו בתוכנית שני תהליכים של MPI. תהליך 0 יקרא את הקלט לתוך מערך של מספרים וישלח חצי מהמערך לתהליך השני. כל תהליך יחשב את ההיסטוגרמה של אחד מחצאי המערך. התהליך השני ישלח את ההיסטוגרמה שחישב לתהליך 0 שימזג אותה עם ההיסטוגרמה שהוא עצמו חישב ויכתוב את הפלט. אם למשל באחת ההיסטוגרמות למספר 7 יש 10 מופעים ובהיסטוגרמה השניה למספר 7 יש 25 מופעים אז בהיסטוגרמה הממוזגת למספר 7 יש 35 מופעים.

כל אחד משני התהליכים (של MPI) ישתמש ב- OpenMP וב- Cuda כדי לחשב את ההיסטוגרמה שלו: בעזרת OpenMP תחושב ההיסטוגרמה של חצי מהערכים ובעזרת OpenMP תחושב ההיסטוגרמה של החצי הנותר.

שתי ההיסטוגרמות האלו ימוזגו להיסטוגרמה אחת (שבהמשך תמוזג עם ההיסטוגרמה שתחושב ע"י התהליך השני).

ניתן לייצג היסטוגרמה בעזרת מבנה נתונים פשוט:

int histogram[256+1]; ואז histogram[1] יהיה מספר המופעים

של המספר 1, histogram[2] יהיה מספר המופעים של המספר 2 כן הלאה . (כאן histogram[0] לא בשימוש כי ערכי הקלט מתחילים מ- 1).

<u>Cuda</u>

יש להשתמש במספר בלוקים שבכל אחד מהם מאות threads (נוח להגדיר

שבכל בלוק יהיו 256 threads -- ראו בהמשך).

דרך פשוטה היא לשמור את ההיסטוגרמה בזיכרון הגלובלי (שכל ה- threads בכל הבלוקים יכולים לגשת אליו). מאחר וכל ה- threads מעדכנים את המערך הזה נוצר race condition.

לכן כדי לעדכן כניסה בהיסטוגרמה יהיה צורך להשתמש ב- atomicAdd.

אם מספר threads ינסו בו זמנית לעדכן (בעזרת atomicAdd) את אותה כניסה בהיסטוגרמה, העדכונים יעשו זה אחר זה באופן סדרתי ולא בו זמנית. זה מבטיח שהעדכונים יעשו בצורה נכונה אבל מאט את התוכנית. תופעה זו של threads שמנסים באותו זמן לעדכן את אותה כניסה צפויה להתרחש לעיתים קרובות אם

כל ה- threads יגשו לאותה היסטוגרמה.

כדי להקל על בעיה זאת ניתן לעבוד בשני שלבים:

בשלב ראשון כל בלוק של threads יבנה היסטוגרמה נפרדת שתאוחסן ב- shared memory (נזכיר שלכל בלוק יש shared memory פרטי שכל ה- threads בבלוק (ורק הם) יכולים לגשת אליו. הגישה ל- shared memory מהירה בהרבה מהגישה ל- global memory). גם כאן יהיה צורך להשתמש ב- atomicAdd (כי כל ה-threads בבלוק

יעדכנו את ההיסטוגרמה של הבלוק) אבל מאחר ובבלוק אחד יש פחות threads מאשר בכל הבלוקים, מספר "ההתנגשויות" צפוי להיות קטן יותר.

בשלב שני ממזגים את כל ההיסטוגרמות של כל הבלוקים להיסטוגרמה אחת שתשב בזיכרון הגלובלי. (היא צריכה להיות בזיכרון הגלובלי כדי שניתן יהיה להעתיקה

לזיכרון של ה- host. ל- host אין גישה ל- shared memory של ה- GPU) .

לצורך כך נוח לקבוע שבכל בלוק יהיו 256 threads ואז כל thread יהיה אחראי

על עדכון של כניסה אחת בהיסטוגרמה הגלובלית.

OpenMP

גם כאן ניתן לחלק את העבודה בין threads. בשלב ראשון כל thread יבנה היסטוגרמה פרטית ובשלב שני ההיסטוגרמות הפרטיות ימוזגו להיסטוגרמה אחת.

המיזוג להיסטוגרמה המאוחדת יכול להיעשות ע"י קבוצה של threads כאשר הגישות להיסטוגרמה מוגנות בתוך `#pragma omp critical`

אבל אז הגישות להיסטוגרמה יעשו באופן סדרתי.

אפשרות טובה יותר היא שכל thread יהיה אחראי על חישוב מספר כניסות בהיסטוגרמה המאוחדת ואז לא נוצר

race condition כי אין מצב בו מספר threads ניגשים לאותה כניסה בהיסטוגרמה.

דוגמא: למען הפשטות נניח שכל המספרים בקלט הם בטווח 1 עד 6 ושיש שלושה threads.

`thread` אחד חישב את ההיסטוגרמה

```
4, 6, 5, 20, 0, 8
```

הכוונה שהמספר 1 הופיע 4 פעמים, המספר 2 הופיע 6 פעמים וכן הלאה.

`thread` שני חישב את ההיסטוגרמה

```
7,  2, 30, 2, 4, 1
```

`thread` שלישי חישב:

```
1, 7, 2, 0, 0, 9
```

ההיסטוגרמה המאוחדת תהיה לכן:

```
12, 15, 37, 22, 4, 18
```

במקרה זה כל `thread` יכול להיות אחראי על חישוב 2 כניסות בהיסטוגרמה המאוחדת. למשל

thread אחד יחשב את 2 הכניסות הראשונות:

```
4+7+1 = 12
6+2+7=15
```

אם משתמשים ב- `#pragma omp for` אז המערכת של openMP כבר תדאג לחלק את העבודה בין ה- threads.

הערה: המיזוג של ההיסטוגרמות היא בעצם פעולה של רדוקציה הפועלת על מערכים (בהנחה שכל היסטוגרמה מיוצגת ע"י מערך). גרסאות חדשות יחסית של OpenMP תומכות בזה אבל השימוש ב- reduction כאן אינו חובה בתרגיל זה).