יונתן בתן 302279138

אליה בן אברהם 305104580

.1

.1.1

בקוד

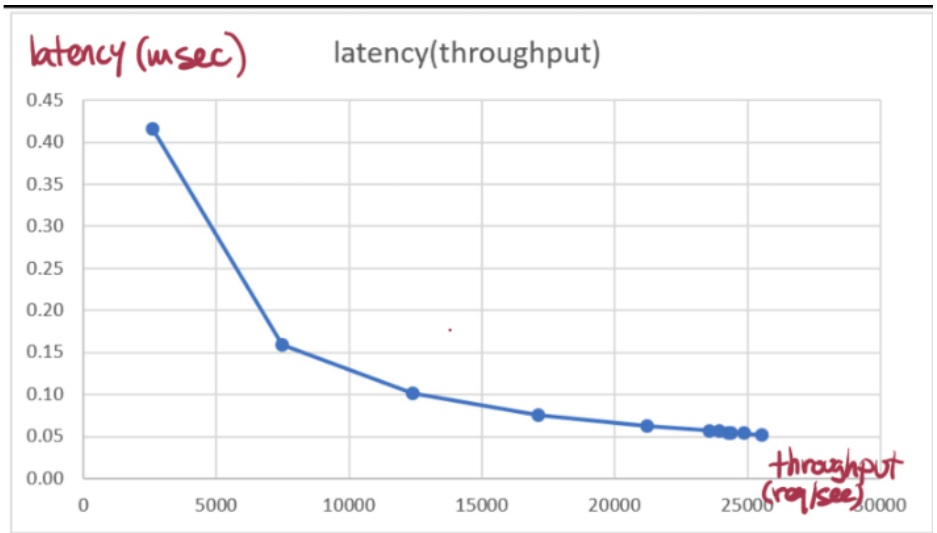
.1.2

קיבלנו $maxLoad = 25{,}925$

.1.3

Load	Latency (msec)	Throughput (req/sec)
2590	0.416	2610
7515	0.159	7487
12440	0.101	12384
17365	0.075	17119
22290	0.062	21228
27215	0.054	24878
32140	0.056	23954
37065	0.057	23556
41990	0.054	24401
46915	0.054	24288
51840	0.052	25533

.1.4



ניתן לראות כי ה- latency יורד ככל שה- throughput גדל וזאת מכיוון שה- GPU תוכנן לעבוד בצורה הטובה ביותר עבור ניצול מקסימלי שלו (עומסים גדולים). בנוסף שימוש ב- streams מאפשר לנו overlap של חישובים והעתקות זיכרון אסינכרוניות, כלומר ניהול יעיל של הפעולות השונות ולכן אין אנו משלמים overhead קשה מדי כאשר ה- throughput גדל

.2

.2.1

נחשב באופן הבא:

- **צריכת בלוק:**
  - **זיכרון משותף:**
    - כל בלוק משתמש ב-2 מערכים של int בגודל 256, מערך של uchar בגודל 256 ועוד int נוסף (לשימוש המימוש הספציפי שלנו).
    - סה"כ 2308 בתים
  - **רגיסטרים:**
    - 32 רגיסטרים לפי שורת הקמפול
  - **מספר חוטים:**
    - קלט מהמשתמש

- **מגבלת חמרה:**
  - **זיכרון משותף:**
    - ע"י שימוש ב- $cudaGetDeviceProperties()$
  - **רגיסטרים:**
    - ע"י שימוש ב- $cudaGetDeviceProperties()$
  - **מספר חוטים:**
    - ע"י שימוש ב- $cudaGetDeviceProperties()$

- לבסוף עבור כל אחת מהמגבלות (זיכרון משותף, רגיסטרים, חוטים) מחלקים את התוצאה של SM בתוצאה של block ולוקחים את המינימום מבניהם ולבסוף מכפילים במספר ה- SMs במערכת

.2.2

בקוד

.2.3

בקוד

.2.4

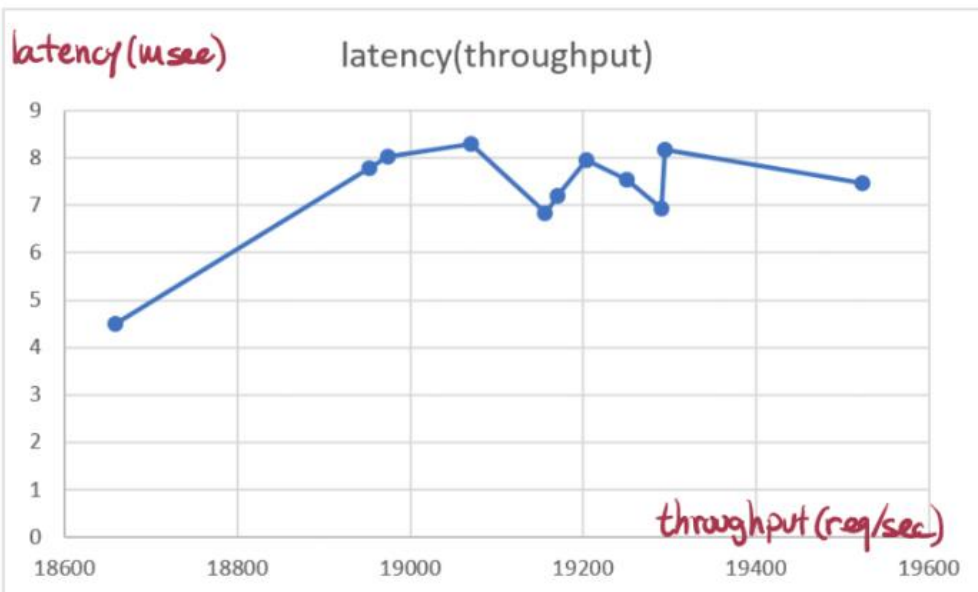
.2.4.1

קיבלנו $maxLoad = 19{,}296$

.2.4.2

Load	Latency (msec)	Throughput (req/sec)
1929	7.545851	19249.8
5595	7.782754	18952.46
9261	6.834462	19156.07
12927	8.297448	19069.88
16593	6.940938	19290.55
20259	8.185568	19294.4
23925	7.477757	19523.42
27591	4.501441	18658.1
31257	7.947155	19202.99
34923	8.034112	18974.25
38589	7.203482	19170.56

.2.4.3



.2.5

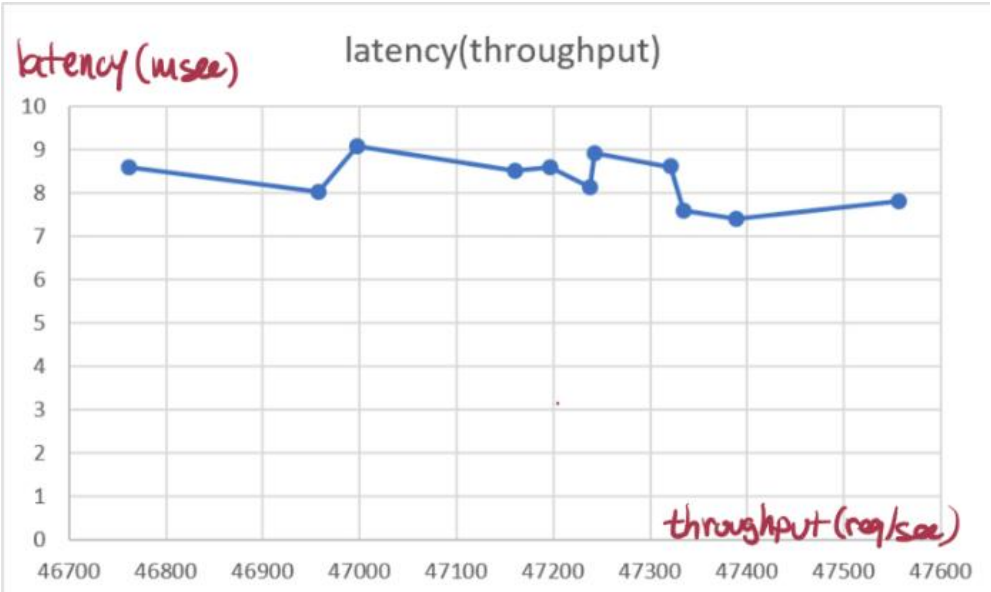
.2.5.1

קיבלנו $maxLoad = 19{,}296$

.2.5.2

Load	Latency (msec)	Throughput (req/sec)
1929	8.030475	46957.4
5595	7.395161	47388.79
9261	8.523662	47160.19
12927	8.130937	47237.29
16593	7.817079	47557.08
20259	8.61018	47320.55
23925	9.075137	46997.17
27591	7.604131	47334.05
31257	8.918309	47242.35
34923	8.585147	47196.26
38589	8.588212	46762.11

.2.5.3



.2.6

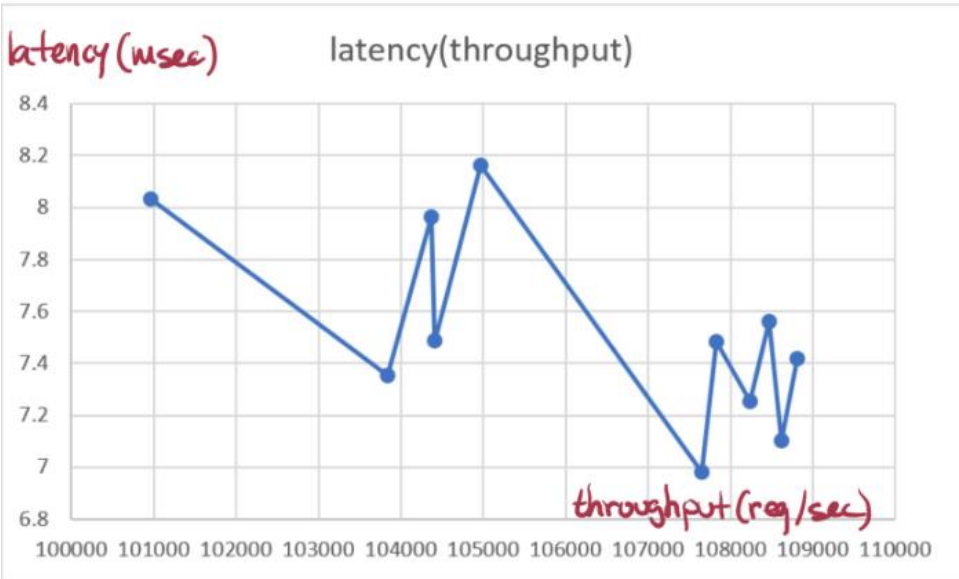
.2.6.1

קיבלנו $maxLoad = 19{,}296$

.2.6.2

Load	Latency (msec)	Throughput (req/sec)
1929	6.983232	107645.8
5595	7.103301	108624
9261	8.161536	104962.4
12927	7.962057	104368.9
16593	7.481808	107829.7
20259	7.255995	108240.1
23925	7.355528	103833.6
27591	8.032151	100969
31257	7.41807	108809.2
34923	7.5605	108462.3
38589	7.485446	104408.9

.2.6.3



.2.7

ניתן לראות שהשינויים ב- latency אינם משתנים בצורה דרסטית כתלות ב- throughput וזאת מכמה סיבות:

- ניהול המשימות מתבצע ידנית ע"י המתכנת – מימוש ה- producer consumer queue ולכן אין לנו את הדעיכה כפי שקיבלנו ע"י שימוש ב- streams
- חלוקת המשימות נעשה בצורת round robin בין הבלוקים השונים כך שיצרנו load balancing כמעט מושלם (עד כדי request יחיד יותר בבלוקים הראשונים) ולכן ה- latency אינו גדל עם הגדלת ה- throughput

בסופו של דבר GPU נועד למשימות עם scale רחב ולכן אם הביצועים שלו היו נפגעים עם הגדלת הקלט הוא לא היה רכיב כ"כ מוצלח ולכן נסכם ונאמר שהמימוש שלנו לא רע בכלל שכן הוא מאפשר scaling עם הגדלת קצב קבלת המשימות

.2.8

הסיבה שבגלל זוהי החלטה נבונה טמונה בהבנה שעדיף לעשות פעולת write מעל PCI מאשר read מעל PCI וזאת מכיוון שפעולת write הינה posted transaction כלומר לא צריכים לחכות שהיא תסתיים על מנת להמשיך לעבוד לעומת פעולת read שהינה non-posted transaction.

אם נעביר את התור CPU-GPU לזיכרון ה- GPU אז כאשר ה- CPU יכתוב אליו הוא יכתוב מעל PCI ואילו כאשר ה- GPU יקרא ממנו הוא יקרא ללא צורך ב- PCI כיוון שהמידע נמצא בזיכרון הלוקאלי שלו לעומת המימוש הנוכחי שם הקריאה מתבצעת מעל PCI ואילו הכתיבה מתבצעת לוקאלית וכפי שהסברנו קודם כתיבה מעל PCI עדיפה מקריאה מעל PCI.

.2.9

במקרה זה ה- GPU יצטרך "לחשוף" את הזיכרון המוקצה לתור ל- CPU ע"י שימוש ב- MMIO, דבר זה יאפשר ל- CPU לגשת לזיכרון ה- GPU באותו אופן בו הוא ניגש לכל כתובת אחרת במרחב הכתובות שלו והכתובת תתורגם לכתובת בזיכרון של ה- GPU ע"י תרגום ל- PCI bus ומשם לכתובת הנכונה בזיכרון ה- GPU עצמו כפי שלמדנו בהרצאה.