

יונתן בתן 302279138

אליה בן אברהם 305104580



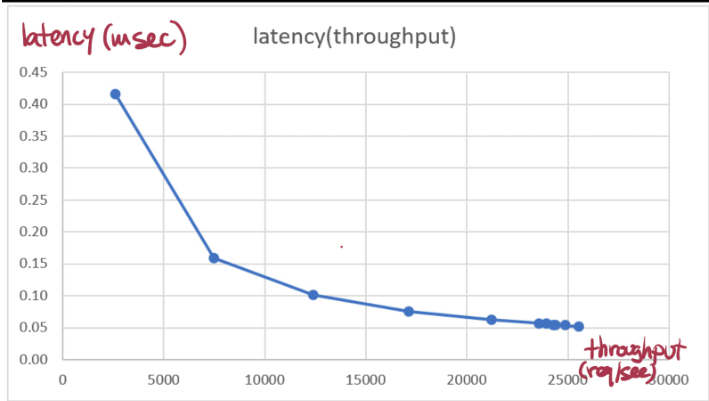
בקוד



קיבלנו

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





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



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-
* **רגיסטרים:**
* ע"י שימוש ב-
* **מספר חוטים:**
* ע"י שימוש ב-
* 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



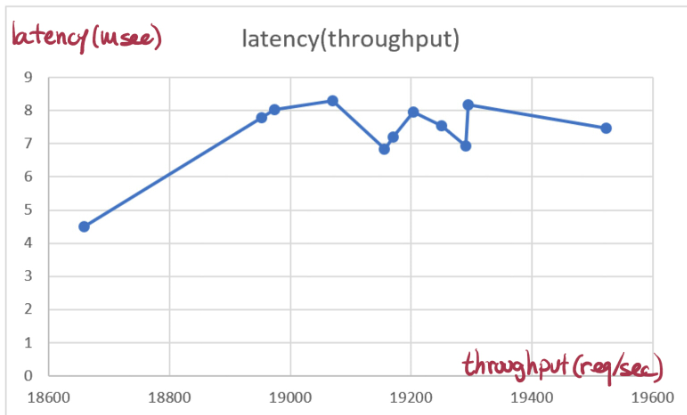
בקוד



בקוד

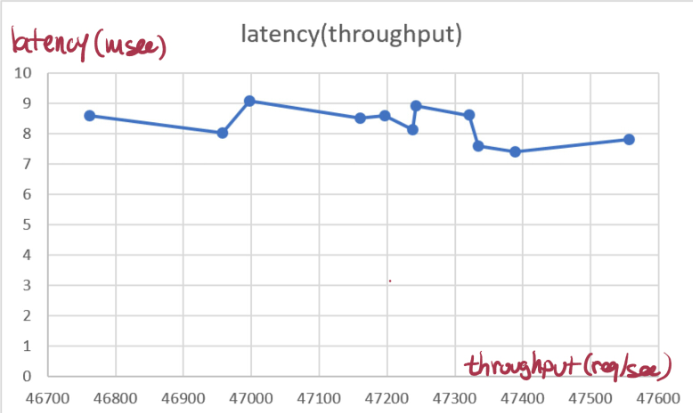
קיבלנו

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

* + 1. 

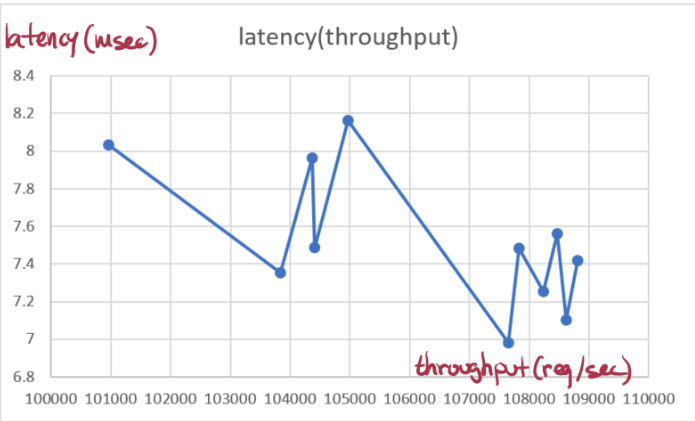
קיבלנו

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

* + 1. 

קיבלנו

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



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

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.

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.