|  |  |
| --- | --- |
|  | tud_logo |
|  | Hochleistungssimulation im Ingenieurwesen 1. Hausübung |
|  |  |
|  |  |
|  | Xiangjun Wei 2594637 Bauingenieurwesen M.Sc  Guanlin Wang 2872498 Bauingenieurwesen M.Sc  Wintersemester 2018/19  21. Nov 2018 |
|  |  |

**1.1 Parallelisierung des Programms zur Berechnung des Matrix-Vektor-Produkts**

Für diese Hausübung machen wir zuerst die Verbindung mit JOCL. Dann schreiben wir eine Programmsource im Hauptclass. Das ist für Programm in OpenCL zu erstellen.

Dann schreiben wir die Mainmethode. Vor allem initialisieren wir die Matrix. Eine m\*m quadratische Matrix ist als A, ein m\*1 Vektor ist als b und das Ergebnis ist m\*1 Vektor als c. Dann gaben wir 3 Pointer für die 3 Arrays.

Danach bekommen wir die Zahl der Platforms und die ID der Platforms. Folgend erstellen wir ein Context für die Platforms. Und die Devices sind ähnlich wie Platforms. Wir bekommen die Anzahl der Devices, die ID der Devices und erstellen das Context für Devices. Dann stellen wir das Programm mit Programmsource und bauen das Programm.

Für Kernel stellen wir das mit Programm und verteilen wir das Memory für die 3 Arrays. Dann bauen wir ein Commad-queue für die ausgewählte Device. Weiterhin geben wir die global\_work\_size als m und local\_work\_size. Für die Aufgaben 1 und 2 geben wir keinen Parameter, und im Aufgabe 3 testen wir mit unterschiedlichen local\_work\_size die Ausführungszeit. Danach geben wir des Arguments für Kernel und führen die Kernels aus. Gleichzeitig bekommen wir die Ausführungszeit mit einer Methode. Zum Schluss geben wir Kernel, Programm und Memory frei.

Dann machen wir die Aufgaben 2 und 3. Im Aufgabe 2 bleiben wir local\_work\_size als null in der Methode „clEnqueueReadBuffer“, und verändern wir die Größe der Matrix. Im Aufgabe 3 bleiben wir die Größe der Matrix immer gleich und verändern wir local\_work\_size. Dann machen wir den Vergleich zu der sequentiellen Abarbeitung.

**1.2 Auswertung des Ergebnisses**

Vergleichen wir die Dauer der sequentiellen Abarbeitung mit der Dauer für die parallelisierte Abarbeitung in verschiedene n – Matrix und konstanter Anzahl der work-groups.

Nehmen wir n in 2500, 5000, 7500,10000,12500,15000 und 17500. Hier folgt das Ergebnis:



Feststellung und Erklärung:

1. Aus die obige Grafik können wir sehen, dass die Dauer der parallelisierten Abarbeitung immer viel kurzer als die Dauer der sequentiellen Abarbeitung. Außerdem Je größer der Matrix, steigt der Zeitaufwand der parallelisierten Abarbeitung langsamer als der von sequentieller Abarbeitung. D.h die parallelisierte Abarbeitung ist mehr effizient als die sequentielle Abarbeitung, wenn der Matrix ganz groß ist. Der Grund liegt darin, dass GPU höhere Rechnenleistung und höhere Speicherbandbreite als CPU hat.

2.Durch mehrmals Versuch finden wir, dass der Fehler von der Dauer der zwei Abarbeitung ganz unterschiedlich ist. Die Dauer für die parallelisierte Abarbeitung verändert sich innerhalb 10 ms, während die Dauer für die sequentiellen Abarbeitung verändert sich innerhalb ungefähr 100 ms. Daher können wir versichern, dass die Dauer von parallelisierter Abarbeitung mehr stabil ist. Der Grund für die Situation ist, dass die sequentielle Abarbeitung ein Aufgabe muss immer warten auf vorher Aufgaben, während parallelisierte Abarbeitung gleichzeitig durchführen kann.

**1.3 Variation der Anzahl der work-groups**

Variieren wir local\_size\_work zwischen 10, 20, 25, 40 und 50 mit 10000x10000 Matrix



Durch den Versuch von 1.2 erhalten wir das optimal local\_work\_size bei 10000x10000 ist 25. Danach nehmen wir die geeignete local\_work\_size neben 25 und vergleichen die Dauer der parallelisierten Abarbeitung. Die obige Grafik ist das Ergebnis. Verwendete Hardware und relevante OpenCL Information liegen hinter:



Feststellung und Erklärung:

Hier können wir klarsehen, die Dauer der Abarbeitung ist am kürzest bei 25 Local\_work\_size. Dies entspricht die optimale Anzahl Local\_work-size bei 10000x10000 Matrix. (Sehe in 1.2 Grafik) Je größer oder kleinere Anzahl verlangsamt die Geschwindigkeit des Rechnens.

Es gibt sogenannt „preferred\_Preferreed\_Work\_Group\_Size” im GPU. Das heißt, wenn man die local\_work\_size erhöhe, wird die Ausführungszeit sich nicht immer verringern, weil die ganzen GPU Prozessoren sequenziell hintereinander liegen. Z.B. wenn man jetzt 50 Sequenzielle Prozessoren hat und die GPU verteilt wird, dass die ersten 30 Problem A bearbeiten sollen und die letzten 20 Problem B bearbeiten sollen. Dann laufen die beide Teile nicht parallel. Sondern erst wird Problem A bearbeitet und dann Problem B, eben wegen dieser sequenziellen Anordnung. Aus diesem Grund bekommen wir die schwankenden Ergebnisse. Außerdem haben Wir bekommen die von OpenCL ermittelte geeignete local\_work\_size für 10000x10000 Matrix in der Aufgabe 2. Das ist 25 und in der Aufgabe 3 könnten wir entdecken, dass die Ausführungszeit mit 25 local\_work\_size relativ kurz ist.

P.S: Bei Überprüfung des Programms pass die Adresse der kernel.cl in parallelisiertes Programm auf.