Praktikum 7: OpenCL Introduction

M.Thaler, 2/2016, ZHAW

1 Einführung

In diesem Praktikum werden Sie die C-Bindings zu OpenCL, das `MCL`-Framework und einige wichtige Funktionen für die Kernelprogrammierung kennen lernen. `MCL` vereinfacht das Erstellen des Host-Programmes mit Funktionen für das Aufsetzen der Plattform (GPU oder CPU), Erzeugen eines Kontextes und Compilieren von Kerneln. Es stellt zudem mit der Struktur `mcl_t` einen Container für alle wichtigen Plattform-Komponenten von OpenCL zur Verfügung.

In einem zweiten Schritt werden Sie eine erste einfache Anwendung implementieren.

Hinweis: in diesem und den nachfolgenden Praktika werden wir der Einfachheit halber nur mit 1-dimensionalen Daten arbeiten.

2 Fist Steps

Wechseln Sie ins Verzeichnis `openclIntro/a0`: hier finden Sie das Modul `mcl`, ein Hostprogramm und einen einfachen OpenCL-Kernel der zwei Vektoren addiert. Machen Sie sich mit der Funktionalität und Anwendung von `MCL` vertraut, Informationen finden Sie in `mcl.h` (`mcl.c`) und `main.c`.

Compilieren und starten Sie das Programm. Als Parameter können Sie die Vektorlänge (Default 1024*1024) und eine Workgroup Size im Bereich 1 bis 256 übergeben (Default 256). Experimentieren Sie mit verschiedenen Workgroup Grössen und Arraylängen (`WIDTH`), sowie Berechnungen auf der GPU und der CPU. Was geschieht, wenn das Verhältnis von Vektorlänge und Workgroup Size nicht ganzzhalig ist? Welchen Einfluss hat die Workgroup Size auf die Performance? Vergleichen Sie die Rechenzeiten auf der GPU und auf der CPU.

3 Wer bin ich?

Für die meisten Berechnungen ist wichtig die ***eigene Position*** im Indexraum (1- bis 3-dimensional) zu kennen. Für die Abfrage der Position stehen mehrere Funktionen zur Verfügung (siehe auch Vorlesungsunterlagen), wobei mit `d` die Dimension (0, 1, 2) angegeben wird:

```
ndi = get_work_dim();      // get number of dimensions
gid = get_global_id(d);    // global index in dim d
bid = get_group_id(d);     // group  index in dim d
tid = get_local_id(d);     // local  index within group in dim d
gsz = get_global_size(d);  // global index range in dim d
lsz = get_local_size(d);   // local  index range in dim d
```

Wechseln Sie ins Verzeichnis `openclIntro/a1`. Sie finden hier einen Kernel, bei dem *printf*-Debugging aktiviert ist (`#pragma ....`). Erweitern Sie das Programm so, dass für jedes Work Item alle Positionsgrössen (resp. Indexgrössen) ausgegeben werden. Experimentieren Sie wiederum mit verschiedenen Daten und Workgroup Grössen.

Hinweis: wenn der Kernel nicht richtig compiliert, können Sie die Berechnung auch auf der CPU (und nicht auf der CPU) durchführen.

4 Kernel und Parameterübergabe

Im Verzeichnis `openclIntro/a2` haben wir ein Programm und einen Kernel vorbereitet, an den ein globaler Array, ein temporärer Array und ein skalarer Wert übergeben werden.

- Analysieren Sie, wie die Argumente für den Kernel aufgesetzt werden und wie die Parameter im Kernel-Code deklariert und genutzt werden.
- Experimentieren Sie mit verschiedenen Workgroup Grössen: wie werden die lokalen Daten des Kernels gehandhabt? Was lässt sich daraus schliessen?

5 The inner product

Eine sehr häufige Vektoroperation ist die Berechnung des inneren Produktes zweier Vektoren, also der komponentenweise Multiplikation der beiden Vektoren gefolgt von einer Aufsummierung der Teilprodukte:

$$C = \sum_{i=0}^{i<N} a[i] \cdot b[i] \qquad \text{resp.} \qquad c[i] = a[i] \cdot b[i] \rightarrow C = \sum_{i=0}^{i<N} c[i]$$

5.1 Aufgaben

Implementieren Sie ein OpenCL Programm, das das Skalarprodukt für zwei Vektoren vom Typ `float` implementiert. Vektorlänge und Workgroup Size sollen als Parameter frei wählbar sein, arbeiten Sie aber zu Beginn ausschliesslich mit 2er Potenzen.

Gehen Sie wie folgt vor:

1. Die Vektormultiplikation kann gleich wie die Vektoraddition implementiert werden. Schreiben und testen Sie einen entsprechenden Kernel. Verwenden Sie zwei globale Arrays für den Input und einen globalen Array für das Resultat (dieser Array kann wiederum als Input für den *Reduction*-Kernel verwendet werden.
2. Für die *Reduction* benötigen Sie einen Kernel, der die Datenwerte innerhalb einer Workgroup aufsummiert. Wählen Sie einen globalen Array sowohl für Input und Resultat, sowie einen lokalen Array für Zwischenresultate.

   Zeichnen Sie sich den Datenfluss für die Reduktion graphisch auf, für eine Workgroup und für die Reduktion der Workgroups. Besprechen Sie die Skizze mit dem Dozenten. Überlegen Sie sich nun, wie sie die Datenindices für die Eingangsdaten und den Index für das Resulat bestimmen können.

   Beide Kernel (Multiplikation und Reduktion) können im gleichen CL-File abgelegt werden.
3. Im Hauptprogramm müssen Sie den *Reduction*-Kernel iterative aufrufen resp. einplanen, um die Resultate der einzelnen Workgroups weiter auf einen Skalar zu reduzieren.

5.2 Implementation

Wechseln Sie ins Verzeichnis `openclIntro/a3`. Wir haben hier das Hauptprogramm zumindest teilweise vorbereitet, einige Funktionen finden Sie auch File mit der Vektoraddition.

Testen Sie das Programm, indem Sie z.B. beide Eingangsvektoren mit 1.0 füllen oder noch besser einen Eingangsvektor mit aufsteigenden Zahlen und den anderen Array mit 1.0.

5.2.1 Wenn Sie noch Zeit haben

Wenn Sie noch Zeit haben, können Sie das Programm für beliebige Vektorlängen erweitern: dazu wählt man als Länge die nächst grössere 2er Potenz und füllt die *überflüssigen* Datenwerte mit einem neutralen Wert. Passen Sie dazu den Kernel an, indem Sie einen zusätzlichen Parameter für die Vektorlänge übergeben.

6 Take Home

Auf was ist bei der Parallelisierung von Algorithmen für *SIMD*-Rechner zu achten und wo liegt dabei das Hauptproblem?

Notieren Sie sich die wichtigsten Punkte, die Sie im Zusammenhang mit diesem Praktikum kennengelernt haben.