GPU Computing

Christian Mösl
University of Salzburg
Austria
cmoesl@cs.uni-salzburg.at

Zusammenfassung

GPU Computing zieht starkes Interesse auf sich, seit dem Machine Learning einen wahnsinnigen Boom erfährt. In dieser Arbeit wird die Geschichte von GPU Computing erklärt und welche Probleme gelöst werden. Source: `https://github.com/ChristianMoesl/ChristianMoesl.github.io/tree/master/assets/wap-abgabe`

1 Einführung

GPU Computing ist 2020 ist in der Informatik ein sehr gefragtes Thema, vorallem weil unter anderem durch Machine Learning ein grosser Leistungsbedarf da ist. Die Graphics Processing Unit (GPU) war aber nicht von Anfang an dafür für diese Aufgabe bestimmt. Es hat sich im Laufe der Zeit herausgestellt, dass die Architektur einer GPU sehr vorteilhaft ist und das Berechnen von großen Datenmengen stark beschleunigen kann. Dies ist vorallem auf die einfache parallelisierbarkeit dieser Probleme zurückzuführen und der Auslegung der Archtitektur auf möglich viel Datendurchsatz.

2 Geschichte

Die ursprüngliche Aufgabe einer Grafikkarte war es noch, Daten welche im Hauptspeicher des Systems liegen irgendwie in elektrische Signale umzuwandeln, welche dann an den Monitor gesendet werden können. Das musste auch bei PC Systemen in den 1960er Jahren bereits 60 mal in der Minute durchgeführt werden. Mit dem Aufkommen von immer aufwendigeren Graphical User Interfaces (GUI), wurden Berechnungen für eben diese bereits von der CPU auf die GPU ausgelagert, sodass die Grafikkarte schließlich zum 2D Beschleuniger wurde. Die Architektur dieser GPU's war noch auf wenige Befehle eingeschrängt und wenig flexibel. Den wirklichen Auslöser für den Boom im Grafikkartenmarkt brachten aber vor allem 3D Spiele. [1] Das zugrundeliegende Problem hierbei ist hierbei das Rendering einer virtuellen 3D Welt auf einen 2D Monitor. Hierbei müssen Objekte geometrisch richtig in einen 3D Raum eingefügt werden, was über verschiedene Vektoren beschrieben wird. Zusätzlich muss noch die Oberflächenstruktur (Textur), Lichtquellen, Schatten und die Perspektive des Betrachters mit ein bezogen werden. Bei hochauflösenden Texturen ist das natürlich wahnsinnig aufwendig zu berechnen. Noch verschärft wird diese Situation dadurch, dass in Spielen diese Szenen 60 mal in der Sekunde berechnet werden müssen um den Echtzeitanforderungen gerecht zu werden. Deshalb wurde aus dem 2D Beschleuniger dann ganz schnell ein 3D Beschleuniger um diesen Anforderungen genüge zu tragen. In diesem Stadium war die Recheneinheit bereits viel komplexer und programmierbarer. [3]

Als Nvidia schließlich das CUDA Framework 2007 veröffentlichte war die Zeit von GPU Computing gekommen. Gepaart mit einer moderneren und flexibleren Architektur konnte man die GPU nun viel leichter für das Verarbeiten von großen Datenmengen einsetzen.

3 Parallelität in Hardware

Die Architektur einer modernen Grafikkarte ist auf hohen Durchsatz ausgelegt. Das heißt es wird primär darauf wert gelegt, dass viele Daten auf einmal verarbeitet werden. Dabei werden 2 Kernkonzepte umgesetzt. In erster Linie wird versucht mehr kleine (einfachere) Kerne parallel laufen zu haben. [5] Außerdem werden in diesen Kernen möglichst viele Arithmetische Einheiten untergebracht (ALU). Diese vereinfachten Kerne (Shader) bauen auf einem anderen Programmiermodel auf. Sogenanntes Streamprocessing, bei dem sich zwar der ganze Shader den Instructionstream teilt, jedoch die Datenfragmente andere sind. Das schränkt das Programmiermodel massiv ein und ist nicht mit herkömlichen CPU Threads vergleichbar. Hierbei muss allerdings beachtet werden, dass diese Shader nicht nur einfache Single-Instruction-Multiple-Data (SIMD) Befehle unterschützen. Es wird zum Beispiel auch Branching in Hardware unterschützt. Damit das effizient umgesetzt werden kann, müssen die Shader mehrere Ausführungskontexte speichern können und zwischen diesen umschalten können um Stalls zu vermeiden.

4 Unterstütze Konzepte

Nachfolgend wird Überblick dafür gegeben, welche Programmier Methoden/Konzepte von einer GPU unterstützt werden.

Map Die Map-Operation wendet einfach die angegebene Funktion (den Kernel) auf jedes Element im Stream an. Ein einfaches Beispiel ist das Multiplizieren jedes Werts im Stream mit einer Konstanten (Erhöhen der Helligkeit eines Bildes). Die Map-Operation ist einfach auf der GPU zu implementieren. Der Programmierer erzeugt für jedes Pixel auf dem Bildschirm ein Fragment und wendet auf jedes ein Programm an. Der gleich große Ergebniss Stream wird im Ausgabepuffer gespeichert. [2]

Reduce Einige Berechnungen erfordern die Berechnung eines kleineren Streams (möglicherweise eines Streams von nur einem Element) aus einem größeren Stream. Dies wird als Reduzierung des Streams bezeichnet. Im Allgemeinen kann eine Reduzierung in mehreren Schritten durchgeführt werden. Die Ergebnisse aus dem vorherigen Schritt werden als Eingabe für den aktuellen Schritt verwendet, und der Bereich, über den die Operation angewendet wird, wird reduziert, bis nur noch ein Stream-Element übrig bleibt.

Stream filtering Stream filtering ist im Wesentlichen eine ungleichmäßige Reduktion. Beim Filtern werden Elemente anhand einiger Kriterien aus dem Stream entfernt.

Scan Die Scan-Operator, auch als parallele Präfixsumme bezeichnet, nimmt einen Vektor (Stream) von Datenelementen und eine (willkürliche) assoziative Binärfunktion '+' mit einem Identitätselement 'i' auf. Wenn die Eingabe $[a_0, a_1, a_2, a_3, \cdots]$ ist,

erzeugt ein exklusiver Scan die Ausgabe $[i, a_0, a_0 + a_1, a_0 + a_1 + a_2, \cdots]$ während ein inklusiver Scan die erzeugt Ausgabe $[a_0, a_0 + a_1, a_0 + a_1 + a_2, a_0 + a_1 + a_2 + a_3, \cdots]$ und erfordert keine Identität. Während die Operation auf den ersten Blick von Natur aus seriell erscheint, sind effiziente Parallel-Scan-Algorithmen möglich und wurden auf Grafikprozessoren implementiert. Die Scanoperation wird beispielsweise bei der Quicksortierung und der spärlichen Matrixvektormultiplikation verwendet.

Sort Die Sortieroperation wandelt eine ungeordnete Menge von Elementen in eine geordnete Menge von Elementen um. Die häufigste Implementierung auf GPUs ist die Verwendung der Radix-Sortierung für Ganzzahl- und Gleitkommadaten sowie der grobkörnigen Zusammenführungssortierung und der feinkörnigen Sortiernetzwerke für allgemein vergleichbare Daten.

Search Die Suchoperation ermöglicht es dem Programmierer, ein bestimmtes Element innerhalb des Streams zu finden oder möglicherweise Nachbarn eines bestimmten Elements zu finden. Die GPU wird nicht verwendet, um die Suche nach einem einzelnen Element zu beschleunigen, sondern um mehrere Suchvorgänge gleichzeitig auszuführen. Die verwendete Suchmethode ist meistens die binäre Suche nach sortierten Elementen.

Datenstrukturen Auf der GPU können verschiedene Datenstrukturen dargestellt werden wie Arrays (dense/sparse) und Union Types.

5 Optimale Probleme

Ob ein Problem sich für GPU Computing eignet hängt von der Struktur ab und ob sich das Problem gut Parallel berechnen lässt.

Embarrisingly-Parallel Zum Glück sind eine ganze Menge an Problemen ganz einfach parallelisierbar. Hierbei lassen sich zum Beispiel die Eingangsdaten einfach partitionieren, wobei dann für jede Partition ein Ergebniss unabhängig von den anderen Partiionen errechnet werden kann. Beispiele hierfür wären zum Beispiel das Bruteforce raten des Eingangswertes einer Einwegfunktion (Passwort cracking). Auch im 3D Rendering bei der Verwendung von Ray Tracing ist es ganz einfach nachvollziehbar, dass das Ergebniss mehrere Lichtstrahlen in einer gegebenen Umgebung parallel berechnet werden kann.

Inherently-Serial Andererseits gibt es dagegen auch Probleme welche komplett ungeeignet für das GPU Computing sind. Zum Beispiel wenn für das Lösen eines Problems sehr viel Branching gemacht werden muss. Ein Beispiel hierfür wäre das finden von Nullstellen durch das Newton Verfahren.

6 GPU Performance

Die Vorteile aus dieser Architektur lassen sich auch Anhand von Benchmarks eindeutig messen. Ein Standardbenchmark ist hierbei das Messen der möglichen Floating-Point Opertationen pro Sekunde.

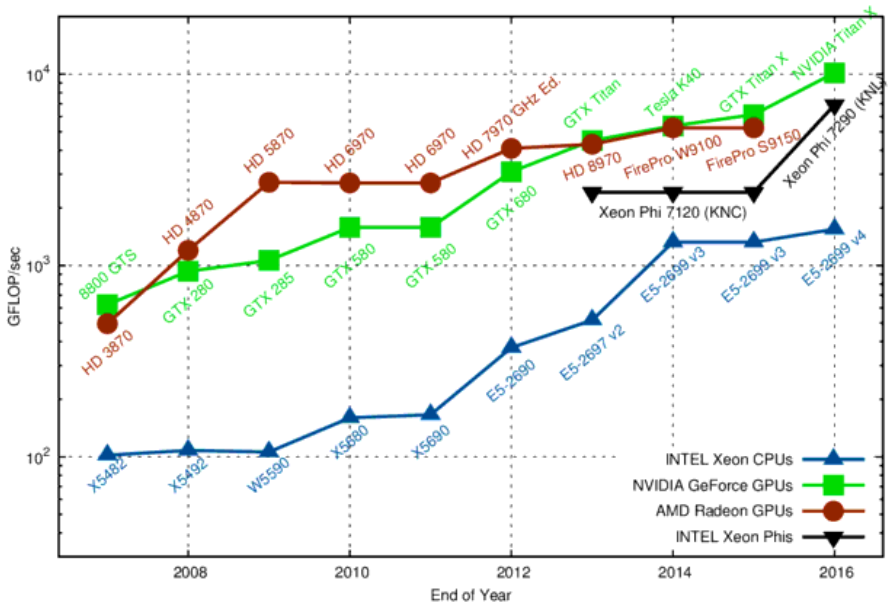

Abbildung 1: Leistungsunterschied von GPU und CPU

Durch die extrem hohe Anzahl an Arithmetischen Recheneinheiten ist hierbei mit einem vielfachen an Leistung gegenüber einer herkömlichen CPU zu rechnen. In der Vergangenheit hat sich das auch immer bestätigt. 1

7 Anwendungsgebiete

GPU Computing kommt Stand 2020 auch in vielen Anwendungsbereichen auch real zum Einsatz und wird viel genutzt.

Deep Learning Deep Learning steht vereinfacht gesagt für das Berechnen von Neuronalen Netzen mit vielen/riesigen Schichten. Eine Schicht besteht aus mehreren Neuronen, wobei jedes Neuron mit jedem anderen Neuron in der nächsten Schicht verbunden ist und verrechnet wird. Beim Training werden nun immer wieder Daten an das Netzwerk angelegt und durch alle Schichten durchgerechnet. Diese Anwendung eignet sich sehr gut für GPU Computing, da hier eine große Datenmenge immer nach dem gleichen Schema durchgerechnet wird (Stream Processing).

8 Fazit

GPU Computing ist ein sehr populäres und mächtiges Tool, das für viele Anwendungsgebiete geeignet ist. Vorallem mit dem Aufkommen von Machine Learning und Big Data ist das ein sehr mächtiges Tool um schnelle Datenverarbeitungs Pipelines zu erstellen. Auch gibt es klare Grenzen, denn nicht jedes Problem lässt sich damit schneller Berechnen.

Literatur

[1] A.S. Abyaneh and C.M. Kirsch. You can program what you want but you cannot compute what you want. In *Edward A. Lee Festschrift*, volume 10760 of *LNCS*. Springer, 2018.

[2] C.M. Kirsch. From logical execution time to principled systems engineering. *Dagstuhl Reports*, 8(2):133, 2018.

[3] C.M. Kirsch, M. Lippautz, and H. Payer. Fast and scalable k-FIFO queues. Technical Report 201204, Department of Computer Sciences, University of Salzburg, 6 2012.

[4] Robert Charles Rempel. *Relaxation Effects for Coupled Nuclear Spins*. PhD thesis, Stanford University, Stanford, CA, June 1956.

[5] Howard M. Shapiro. Flow cytometry: The glass is half full. In Teresa S. Hawley and Robert G. Hawley, editors, *Flow Cytometry Protocols*, pages 1-10. Springer, New York, NY, 2018.