Matrix Vektor Multiplikation

Harry Findeis

Juli 2020

Inhaltsverzeichnis

1 Einleitung

Lange Zeit konnte man sich auf das Mooresche Gesetz verlassen, welches einen Leistungszuwachs von Mikroprozessoren pro Jahr um 50 Prozent beschrieb. Mittlerweile ist die Halbleiterphysik an ihre Grenzen gestoßen, weswegen sich die Leistungssteigerungen bei CPUs pro Jahr deutlich verringert haben. Im Vergleich dazu wird NVIDIA GPU-Computing bis 2025 eine 1000-fache Beschleunigung der Leistung realisieren [1]. NVIDIA bezeichnet das GPU Computing als das beschleunigte Rechnen durch Grafikprozessoren. Möglich wird diese Leistungssteigerung durch einen hoch parallel strukturierten Aufbau der GPU.

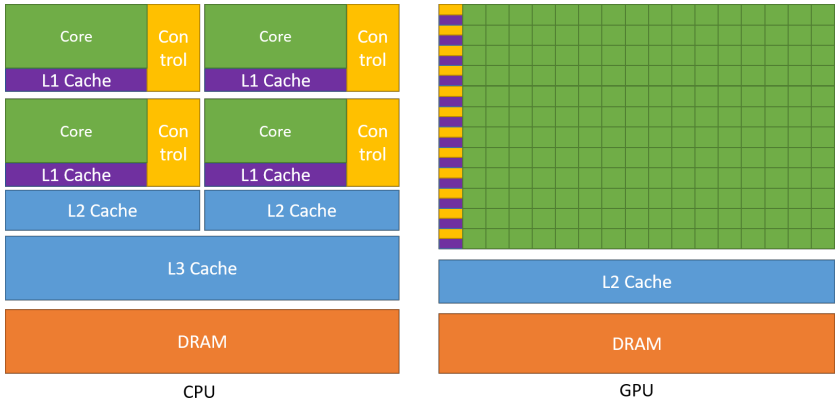

Abbildung 1: CPU vs GPU [5]

Mit Cuda hat NVIDIA eine Computing Plattform und Programmier-Modell für das Computing an GPUs entwickelt. Mittels Cuda können Programme durch GPUs beschleunigt werden [2]. Die Projektaufgabe besteht darin, die Vorteile des parallelen Rechnens auszunutzen und eine Matrix-Vektor Operation mittels Cuda für Nvidia GPUs zu implementieren.

2 Problemstellung und Implementierung

Die Laufzeit einer Matrix-Vektor Operation von der Größe einer m x n dimensionalen Matrix liegt bei der Ordnung O(m*n). Ziel ist es, die Rechenoperationen bestmöglich zu parallelisieren, indem die Arbeit auf verschiedene Threads aufgeteilt wird. CUDA ermöglicht es, Funktionen, auch Kernels genannt, parallel aufzurufen und von verschiedenen Threads bearbeiten zu lassen. Die Matrix-Vektor Operation soll in verschiedenen Kernel-Funktionen umgesetzt werden, die sich durch "Shared-Memory", "Atomic-Operations" und "Intra-Grid Communication" unterscheiden. Des Weiteren werden unterschiedliche Cache-Konfigurationen verwendet. Die grundlegende Idee für die Umsetzung liegt zunächst in der maximalen Parallelisierung der Rechenoperationen. Hierzu berechnet jeder Thread eine Multiplikation. In den folgenden Kernels wurden verschiedene Techniken angewendet um die Teilsummen zu sammeln.

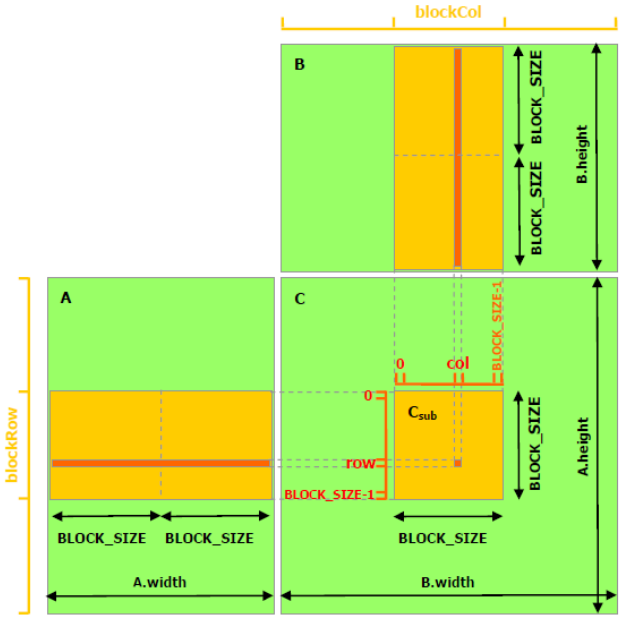

Abbildung 2: Shared Memory - zu beachten ist, dass für die Matrix-Vektor-Operateion die Spaltenanzahl für die B- und C-Matrix bei eins liegt. [4]

Kernel 0 (Shared-Memory Reduktionskernels) In diesem Kernel wurde ausgenutzt, dass Threads, die zu einem Block zusammengefasst sind, auf gemeinsamen Speicher (Shared-Memory) zugreifen können. Dadurch können Speicherzugriffe für die Berechnungen beschleunigt werden. Somit können alle Teilsummen, die im Shared-Memory liegen, aufsummiert werden. Hierfür wurden die Blöcke geschickt angelegt, sodass die Länge der Blöcke in der Höhe, wie in Abbildung 2 zu sehen ist, bei eins liegt und somit alle Teilsummen im Shared-Memory ohne weitere Unterscheidung zusammen gezählt werden können. Die Summen der Blöcke können anschließend im Reduktionskernel zusammengefasst und in den Ergebnisvektor geschrieben werden.

Kernel 1 (Shared-Memory Atomic-Add) Atomic-Operations sind Rechenoperationen, die nicht unterbrochen werden können, um eine Möglichkeit zu bieten, den Speicher konsistent zu halten, da gleichzeitige Schreibvorgänge auf eine Speicherstelle ausgeschlossen sind. Diese Atomic-Add Operation wird ausgenutzt, um die Teilsummen der Blöcke direkt auf den Ergebnisvektor zu addieren.

Kernel 2 (Shared-Memory Intra-Grid-Communication) Mittels "Intra-Grid-Communication" ist es möglich, Blöcke in einem Grid zu synchronisieren. Hier ist die Anzahl der zur Verfügung stehenden Streaming-Multiprozessoren der GPU-Architektur zu beachten, die eine Limitierung der "Intra-Grid-Communication" darstellen.

Kernel 3 (all Atomics) Im letzten Kernel schreiben alle Threads ihre Teilsumme direkt mittels eines Atomic-Add-Befehls in den Ergebnisvektor.

3 Performanceanalyse

Die Zeitmessung für die Performanceanalyse wird über CUDA Events durchgeführt. Mit cudaEventElapsedTime kann die Zeit zwischen zwei Events mit einer Genauigkeit von etwa etwa 0,5 Mikrosekunden gemessen werden. Die Zeitausgabe selbst wird in Millisekunden angegeben [3]. Das Programm wurde auf zwei unterschiedlichen GPU-Architekturen ausgewertet. Die erste GPU ist eine GeForce MX250 vom Architekturtyp Pascal mit einer CUDA Capability von 6.1 und 3 Streaming-Multiprozessoren. Die zweite GPU ist eine RTX 2070 Super vom Architekturtyp Turing mit einer CUDA Capability von 7.5 und 40 Streaming-Multiprozessoren.

Performanceanalyse der Kerne

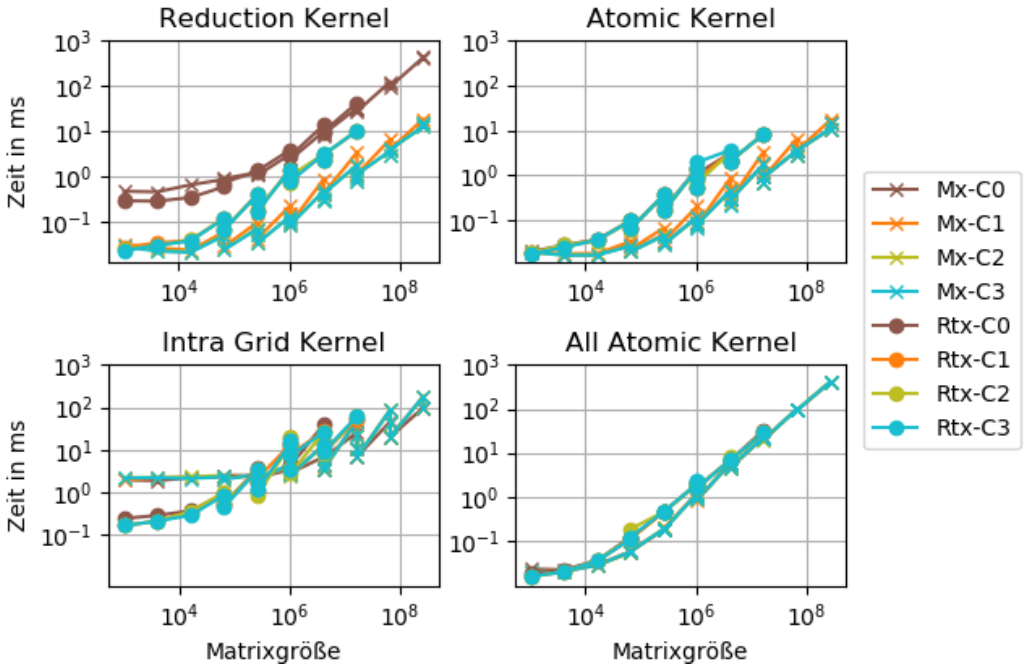

Abbildung 3: Analyse der Kernel in vier Plots; Werte der Grafikkarte MX-250 werden mit einem x-Marker dargestellt und Werte der RTX 2070 Super mit einem Punkt; Cache-Konfigurationen werden in unterschiedlichen Farben dargestellt, C0-Cache-Prefer-None, C1-Cache-Prefer-L1, C2-Cache-Prefer-Shared und C3-Cache-Prefer-Equal gilt

In der Abbildung 3 kann man sich einen Überblick verschaffen, wie lange die Kernels für unterschiedliche Matrixgrößen brauchen, um das Matrix-Vektor-Produkt zu berechnen. Im Plot des "Reduction Kernel" kann man erkennen dass die Cache-Config "Cache-Prefer-None" zu einer längeren Bearbeitungszeit führt. Im Plot des "Intra-Grid Kernel" sieht man, dass für kleine Matrizen die GPU MX-250 länger benötigt als die RTX-2070 Super.

Performanceanalyse der Cache-Config

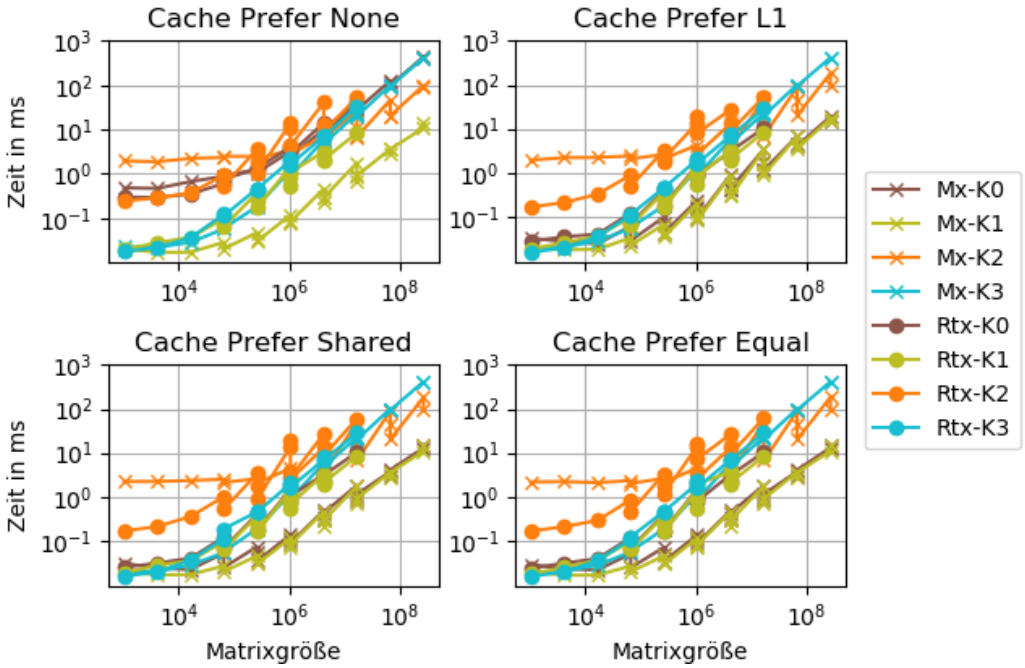

Abbildung 4: Analyse der Cache-Config in vier Plots; Werte der Grafikkarte MX-250 werden mit einem x-Marker dargestellt und Werte der RTX 2070 Super mit einem Punkt; Kernel werden in unterschiedlichen Farben dargestellt, wobei K0-Reduction-Kernel, K1-Atomic-Kernel, K2-Intra-Grid-Kernel und K3-All-Atomic-Kernel gilt

Bei der Abbildung 4 ist die Perfomanceanalyse für unterschiedliche Cache-Konfigurationen dargestellt. Wie in der vorherigen Abbildung kann man erkennen, dass der "Reduction-Kernel" mit der Cache-Config "Prefer None" langsamer ist als mit anderen Cache-Konfigurationen.

Analyse der Blockdimension

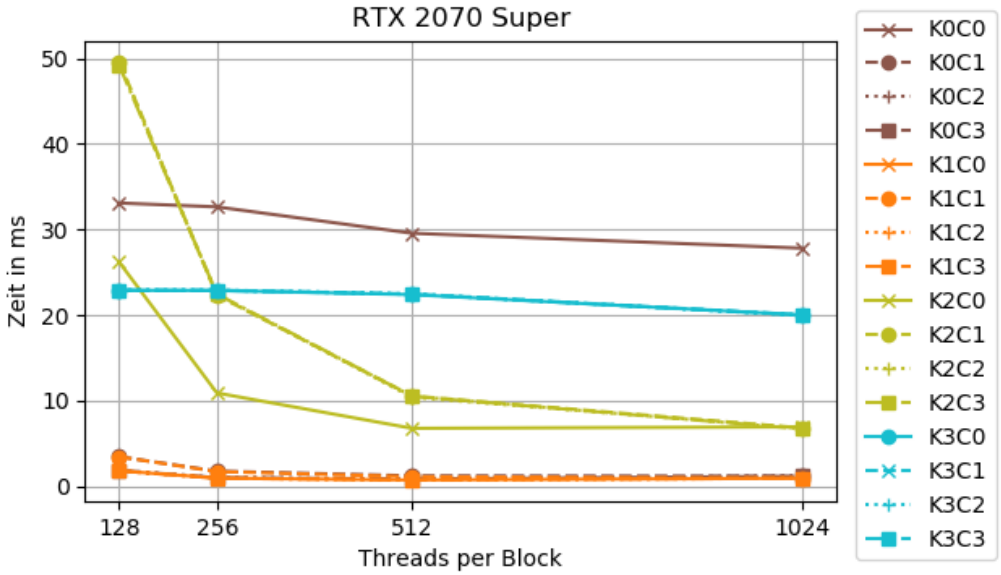

Abbildung 5: Analyse der Blockdimension auf der GPU RTX 2070 Super; Matrixgröße ist 4096 x 4096; Kernel werden in unterschiedlichen Farben dargestellt, wobei K0-Reduction-Kernel, K1-Atomic-Kernel, K2-Intra-Grid-Kernel und K3-All-Atomic-Kernel gilt; Cache-Konfigurationen werden in unterschiedlichen Markern dargestellt, wobei C0-Cache-Prefer-None, C1-Cache-Prefer-L1, C2-Cache-Prefer-Shared und C3-Cache-Prefer-Equal gilt

In der letzten Abbildung 5 erkennt man den Einfluss der Blockgröße auf die Berechnungszeit der Kernels. Der “Atomic-Kernel” (Orange) ist der schnellste Kernel und weist kaum eine Änderung in den unterschiedlichen Blockgrößen auf. Die Kernel “All-Atomic-Kernel” und “Reduction-Kernel” haben eine höhere Bearbeitungszeit, die sich ebenfalls nur leicht in bei den Blöckgrößen ändert. Bei dem “Intra-Grid Kernel” wurde bei niedriger Blockgröße von 128 Thread eine hohe Bearbeitungsdauer gemessen. Diese Bearbeitungzeit fällt schon bei der nächsten Threadgröße stark ab und liegt am Ende nur leicht höher als bei dem “Atomic-Kernel”.

4 Zusammenfassung

Ziel war es, die Berechnung der Matrix-Vektor Operation durch Anwendung von Cuda-Kernels zu beschleunigen und auf verschiedenen Nvidia GPU-Architekturen zu vergleichen. Man stellt fest, dass der “Atomic-Kernel” häufig die besten Ergebnisse liefert. Vom “All-Atomic-Kernel” ist abzuraten, da er für alle Cache-Konfigurationen den steilsten Anstieg der Bearbeitungsdauer aufweist. Auffallend sind auch die Unterschiede der GPUs, welche durch eine Anpassung des Programmcodes an die Architektur der GPU die Ergebnisse stark beeinflusst.

Literatur

[1] Nvidia. *About Nvidia*. URL: https://www.nvidia.com/de-de/about-nvidia/ai-computing/ (besucht am 25. 06. 2020).

[2] Nvidia. *Developer Cuda-Zone*. URL: https://developer.nvidia.com/cuda-zone (besucht am 25. 06. 2020).

[3] Nvidia. *EventManagement*. URL: http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/online/group__CUDART__EVENT_g14c387cc57ce2e328f6669854e6020a5.html (besucht am 25. 06. 2020).

[4] Nvidia. *Shrard Memory*. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#shared-memory (besucht am 25. 06. 2020).

[5] Nvidia. *The Benefits of using GPUs*. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#from-graphics-processing-to-general-purpose-parallel-computing (besucht am 25. 06. 2020).

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Jena, den 02.07.2020

_____