Parallelisierung eines NVIDIA GPGPU Client Server Services basierend auf TCP-Sockets

Til Koke
Institut für Betriebssysteme und Rechnernetze
TU Braunschweig
Mühlenpfordstr.23, 38106 Braunschweig
Email: til.koke@tu-bs.de

Maximilian Wesche
Institut für Betriebssysteme und Rechnernetze
TU Braunschweig
Mühlenpfordstr.23, 38106 Braunschweig
Email: maximilian.wesche@tu-bs.de

David Winterland
Institut für Betriebssysteme und Rechnernetze
TU Braunschweig
Mühlenpfordstr.23, 38106 Braunschweig
Email: david.winterland@tu-bs.de

***Abstract*—Die Ausarbeitung befasst sich mit einer verteilten Anwendung für das Auswerten von Stereobildern. Ein vorhandener Algorithmus wird verteilt auf einer Grafikkarte, und auf mehreren Grafikkarten in einem Rechnernetz ausgeführt. Dabei wird eine neue Methode zur Datenübertragung durch TCP-Sockets durchgesetzt. Die Ausführungsgeschwindigkeit wird mit einer SOAP Kommunikationschnittstelle und einer lokalen Ausführung ohne Rechnernetz verglichen.**

I. Einleitung

Verteilte Systeme helfen bei der Berechnung von komplexen mathematischen Problemen. Durch eine verteilte Berechnung kann das Ergebnis schneller berechnet werden, somit ergeben sich zum Teil große Performance Verbesserungen. Ein allgemein bekanntes Beispiel für parallele Berechnungen ist das Multi-Threading in gängigen Prozessoren von PCs. Hier werden große Aufgaben, wie bspw. Berechnungen, auf mehrere CPU Kerne aufgeteilt, typischerweise besteht eine CPU aus 2 bis 6 Kernen. Bei der Berechnung von grafikintensiven Aufgaben, bspw. die Berechnung einer Disparitätenmatrix aus Stereobildern, bieten GPUs auf Grund der großen Anzahl an Kernen und der Möglichkeit diese parallel zu nutzen enorme Performance Vorteile gegenüber CPUs. Diesen hohen Grad an Parallelität für Probleme abseits des Renderings zu nutzen, nennt sich "General Purpose" Berechnungen auf GPUs (GPGPU). Da aber auch die Anzahl der Kerne einer GPU begrenzt ist, ist es möglich für sehr große Probleme die Rechenleistung mehrerer GPUs, verteilt über das Netzwerk, zu verwenden. Für die Verteilung der Aufgaben im Netzwerk gibt es bereits eine Anwendung, die Performance Optimierung dieser Anwendung mit einem Ansatz der auf der Serialisierung mit nativen C-Client-Server-Sockets basiert ist Hauptbestandteil dieser Arbeit. Der Abschluss der Arbeit besteht aus einem Performance Vergleich der beiden Anwendungen bei der verteilten Berechnung von Disparitätenmatrizen aus Stereobildern.

II. Ansatz

Compute Unified Device Architecture (CUDA) ist eine Bibliothek die es erlaubt, auf NVIDIA GPUs eigenen, C ähnlichen Code auszuführen. Um Rechenaufgaben auf mehrere GPUs im Netzwerk zu verteilen und für den Zeitpunkt der Berechnung ein lokales Rechencluster zu bilden, gibt es bereits eine Anwendungen die einen GSoap Webservice verwendet, um die Berechnung der Disparitätenmatrix auf die einzelnen GPUs zu verteilen. Dadurch dass jedem Bildpunkt ein xml-tag angefügt wird, ist die Übertragung sehr langsam. Im Vergleich zu einer lokalen GPU Berechnung ist die verteilte Berechnung durch den Overhead der Serialisierung langsamer. Der Ansatz dieser Arbeit ist es die Verteilung basierend auf C-Client-Server Stream-Sockets zu realisieren.

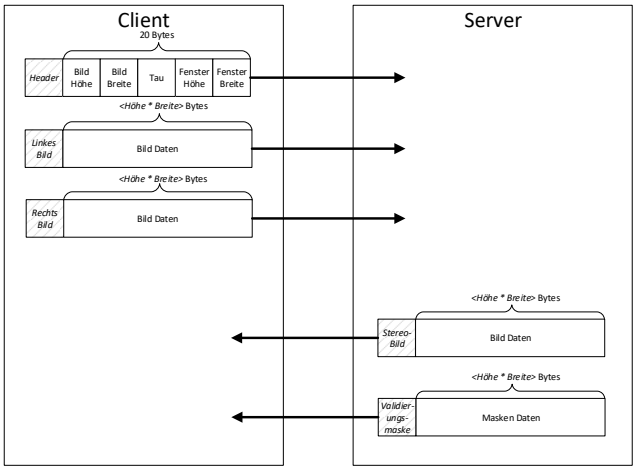

Fig. 1. Paketaufbau Client-Server-Sockets.

III. Implementierung

Die verteilte GPGPU Berechnung der Disparitätenmatrixauf wird auf 6 identischen Alienware Aurora PCs ausgeführt, diese besitzen folgende Hardware:

- Intel i7 6400 Prozessor mit 4 GHz
- 8 GB Arbeitsspeicher
- NVidia GTX1080 Grafikkarte mit 2560 CUDA Kernen und 8 GB VRAM
- Alle Rechner sind mit Gigabit Ethernet verbunden

Bei der verteilten Anwendung werden die Bilder von der Anwendung Stereo-Lab zeilenweise aufgeteilt, auch der CUDA Algorithmus wird bereits von der Stereo-Lab Anwendung bereitgestellt. Die Verteilung der Bilddaten für

die Berechnung der Disparitätenmatrix basiert auf Client-Server Sockets. Hierbei werden TCP-Stream Sockets verwendet. Als Implementierungsgrundlage werden POSIX Sockets verwendet, die unter den meisten Linux-Distributionen zur Verfügung stehen. Zuerst wird ein Datenpaket konstanter Größer versendet, welches Metainformation über das zu lösende CUDA-Problem enthält. Es wird die Größe des Teilbildes und die Berechnungsparameter festgelegt.

- **Bild Höhe** in absoluten Pixeln.
- **Bild Breite** in absoluten Pixeln.
- **Tau** Suchbereich des Berechnungsfensters.
- **Fenster Höhe** des Berechnungsfenster in absoluten Pixeln.
- **Fenster Breite** des Berechnungsfenster in absoluten Pixeln.

Der genaue Paketaufbau und das Übertragungsprotokoll ist in Abbildung 1 symbolisiert. Es folgt die Übertragung der eigentlichen Bilddaten. Sie werden auf zwei Pakete aufgeteilt. Diese beinhalten jeweils den Teil des linken bzw. rechten Bildes, welches die Empfänger für die Berechnung nutzen soll. Die Größe der Pakete wird aus der Bildhöhe und -breite errechnet, die in den Metadaten des ersten Datenpaketes enthalten sind. Die Bilder werden unkomprimiert als 256-stufige Grauwerte übertragen. Pro Pixel wird ein Byte Speicher verwendet. Wichtig ist hierbei, dass die Zeilen immer vollständig übertragen werden, da sonst relevante Information für die Berechnung fehlen. Da der Algorithmus der StereoLab-Software zeilenweise arbeitet, werden die Bildzeilen immer vollständig übertragen. Auf dem Server wird nun der CUDA-Algorithmus für das Teilbild ausgeführt. Als Ergebnis wird ein Stereobild und eine Validierungsmaske geliefert. Sie besitzen die selbe Höhe und Breite wie die Eingabebilder. Das Stereobild ist ebenfalls ein Grauwertbild, wobei ein hoher Grauwert eine geringere Distanz zum Betrachter repräsentiert. Die Validierungsmaske enthält zusätzlich Informationen darüber, für welche Pixel im Stereobild die Disparitätsberechnung keine gültigen Ergebnisse liefert. In beiden Fällen wird, wie bei den empfangenen Bildfragmenten, ein Byte pro Pixel verwendet. Insgesamt benötigen die Ergebnisse der Bereichnung also genauso viel Speicher wie beide Teilbilder. Da die Validierungsmaske nur Boolsche-Werte enthält, wäre hier eine weitere Optimierung Übertragungsformates nötig. Das berechnete Stereobild sowie die Validierungsmatrix werden nach der Berechnung zum Client zurückgesendet. Dort werden die Teilbilder zusammengesetzt. Diese Vorgehensweise erlaubt es, die Bilder auf eine beliebige Anzahl an Rechnern zu verteilen. Die Anzahl ist maximal beschränkt durch die Zeilen des Ursprungsbildes durch die Höhe des gewählten Fenster. (So wird sichergestellt, dass die Teilbilder in jedem Rechner auch gültige Eingangsdaten für den Disparitätsalgorithmus sind.)

IV. Validierung

Im weiteren Verlauf dieser Arbeit erfolgt ein Vergleich zwischen den beiden Implementierungen: Beide Implementierung, sowohl die vorliegende, als auch die in dieser Arbeit

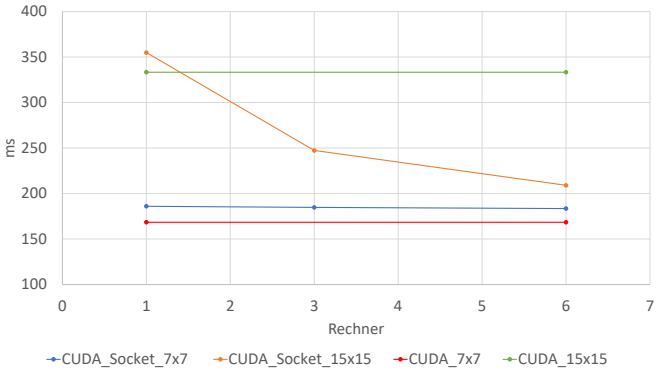

Fig. 2. Vergleich der Socket Implementation gegenüber dem CUDA-Algorithmus auf einer Grafikkarte.

vorgestellte, wurde beide jeweils lokal, verteilt auf drei und auf sechs Rechnern getestet. Dabei wurde für jede Testreihe der gleiche Algorithmus 3 mal ausgeführt und der Mittelwert gebildet um eine bessere Vergleichbarkeit zu erreichen. Zum Vergleich wurde zusätzlich der CUDA Algorithmus ohne Serialisierung auf der lokalen Grafikkarte getestet.

Es wurde für alle Tests das Stereobild eines Elefanten verwendet mit der Fenstergröße 7x7 bzw. 15x15 Pixel.

Für die kleinere Fenstergröße ist die Socket-Implementierung langsamer, als die lokale Ausführung (Siehe Abbildung 2). Das liegt daran, dass die Teilprobleme durch die Anzahl der Kerne einer Grafikkarte überdeckt werden. Das sieht man auch daran, dass mit steigender Rechner Anzahl, die benötigte Zeit nicht sinkt. Somit kann ermittelt werden, dass die Serialisierung und Deserialisierung für dieses Bild ca. 20 ms benötigt.

Für die größere Fenstergröße kann man mit steigender Rechneranzahl einen Performance-Gewinn gegenüber der lokalen Ausführung beobachten. So ist die Socketimplementierung mit sechs parallelen Rechnern um mehr als 100 ms schneller als die lokale Alternative.

Der Vergleich zur CUDA Implemtierung mit GSoap und XML fällt noch deutlicher aus. Die Ergebnisse sind in Abbildung 3 aufgetragen. Diese Variante profitiert zwar sichtbar von der Verwendung multipler Grafikarten, die Serialisierung mit Hilfe von XML benötigt ca. die 10-fache Zeit des eigentlichen CUDA-Algorithmus.

V. Zusammenfassung und weitere Arbeiten

In der Evaluation wird deutlich, dass die verteilte Disparitätsberechnung schneller ausgeführt wird, wenn die Komplexität der Teilprobleme (hier: die Fenstergröße) groß genug ist. Außerdem ist die Serialisierung mit Sockets wesentlich effizienter als die einer SOAP Schnittstelle.

Damit hat die Arbeit gezeigt, dass das zusammenschließen mehrerer Grafikkarten für die Berechnung großer Probleme wesentlich schneller sein kann. Gleichzeitig liefert diese Arbeit eine Lösung für die verteilte Berechnung von Stereobilder mit Hilfe von Stream-Sockets die wesentlich schneller ist, als die Konkurrenz.

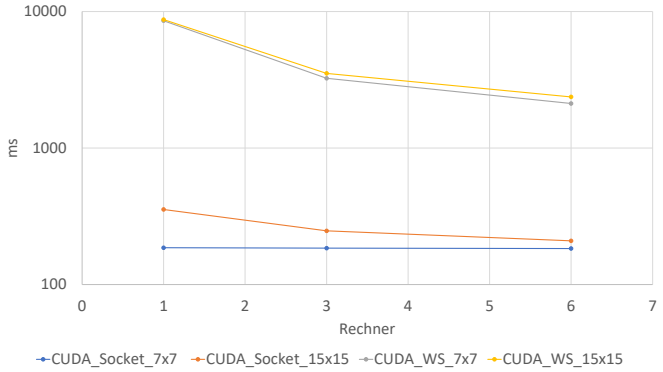

Fig. 3. Vergleich der Socket Implementation gegenüber der Webservice Variante.

Die hier ausgewerte Implementierung befasst sich nur mit der Übertragung durch das TCP-Protokoll. Ebenso könnte sich die Verwendung von UDP anbieten, da das Protokoll weniger Overhead hat. Jedoch müsste dabei zusätzlich sichergestellt werden, dass die Bilddaten auch in der richtigen Reihenfolge übermittelt werden. Neben den direkten Sockets wären auch Middleware-Übertragungsschichten interessant, die weniger Serialisierungsaufwand betreiben als SOAP. Ein Kandidat ist das Java-RMI Framework, dass die Übertragung großer Datenmengen auf Byteebene der JVM ermöglicht. Ein weiterer möglicher Ansatz ist, die Bilddaten vor der Übermittlung zu komprimieren. Dabei müsste man sich auf Verfahren beschränken, vollständige Rekonstruktion der Originaldaten ermöglicht, um die Ergebnisse der CUDA-Berechnung nicht einzuschränken.

References

[1] GSOAP, https://www.genivia.com/doc/soapdoc2.html#tth_sEc11.11, 6.11.2018.