Zeit- und bewegungsabhängige Rauschreduktion

Elmar Bührle[1], Benjamin Keck[1,2], Stefan Böhm[3], Joachim Hornegger[1]

[1]Lehrstuhl für Mustererkennung, Martensstraße 3, 91058 Erlangen
[2]Siemens Healthcare, CV, Medical Electronics & Imaging Solutions,
Postfach 3260, 91050 Erlangen
[3]Siemens Healthcare, Imaging & IT, Angiography, Siemensstr. 1, 91301 Forchheim
elmar@buehrle.org

Kurzfassung. Dieser Beitrag beschäftigt sich mit der Echtzeit 2D-Bildverarbeitung mittels NVIDIAs CUDA-Schnittstelle anhand des gewählten komplexen Filters. Es werden verschiedene Optimierungsdetails hevorgehoben und erläutert, die für CUDA-Implementierungen verallgemeinert werden können. Dabei erzielt unsere Umsetzung eine Beschleunigung um den Faktor 45 gegenüber der Referenzimplementierung auf CPU-Basis und ist damit zu aktueller Spezialhardware konkurrenzfähig.

1 Einleitung

Die Bildgebung interventionell genutzter Angiographieanlagen mittels Röntgentechnologie stellt heute sehr hohe Anforderungen an die Bildqualität. Die Bildqualität wird durch verschiedenste Faktoren beeinflusst und ist für den Echtzeitbetrieb sicherzustellen, d.h. typischerweise sind die Algorithmen zur Bildverbesserung für einen maximalen Datenstrom von 480-600MBits/s (monoplan-System) bei einer maximalen Latenz von ca. 100 ms auszuführen. Für unsere Untersuchung haben wir einen mehrstufigen zeit- und bewegungsabhängigen Rauschreduktionsalgorithmus gewählt, der in kommerziellen Angiographiegeräten verwendet wird. Die Berechnungsgeschwindigkeit von Bildverarbeitungsalgorithmen ist sowohl von der Komplexität der Algorithmen als auch von der verwendeten Hardware abhängig. Um die Echtzeitfähigkeit des Systems bei der Verarbeitung der akquirierten Bildfolge zu garantieren, muss bei vielen Produkten auf Spezialhardware, wie z.B. DSPs oder FPGAs, zurückgegriffen werden. Die Konsequenz ist ein erhöhter Aufwand für die Hardware-Entwicklung und für die Implementierung der Algorithmen auf der Spezialhardware.

Im Jahre 2007 stellte der Grafikprozessorhersteller NVIDIA gemeinsam mit einer neuen Generation von Grafikprozessoren (GPUs) eine Programmierschnittstelle namens “Common Unified Device Architecture”(CUDA) vor. Mithilfe einfacher Erweiterungen der Programmiersprache C kann eine CUDA-fähige Grafikkarte zur Berechnung verschiedenster Algorithmen verwendet werden. Die erste Generation entsprechender GPUs (G80) erreichte durch die extreme Parallelität bereits eine Leistung von mehr als 300 Gigaflops.

In unserer Untersuchung gehen wir auf die Eignung der im Vergleich zu Spezialhardware günstigeren, sowohl leicht integrierbaren als auch vergleichsweise einfach programmierbaren Hardware am Beispiel des gewählten Algorithmus ein.

2 Stand der Forschung und Beitrag

Die Eignung von Grafikkarten zur Beschleunigung von Berechnungen im Bereich der Medizinischen Bildverarbeitung wird bereits seit über einem Jahrzehnt erforscht [1]. Mack et. al. zeigten 2004 [2] die Verwendbarkeit von Grafikkarten im Bereich der "Echtzeit-Röntgenbildverarbeitung", wobei die verwendeten Algorithmen mittels Direct3D auf die bisherige Rendering Pipeline abgebildet wurden. Algorithmen werden dabei mittels Shader-Programmiersprachen wie GLSL oder HLSL in die spezialisierte Grafikwelt abgebildet. Die effiziente Umsetzung von Algorithmen auf GPUs mittels CUDA konnte bereits 2007 im Bereich der 3D-Rekonstruktion gezeigt werden [3]. In unserem Beitrag möchten wir die Eignung von GPUs zur Echtzeit 2D-Bildverarbeitung mittels CUDA und die dabei zu beachtenden Programmiertechniken anhand eines Beispielalgorithmus erläutern.

3 Methoden und Algorithmen

Bei dem beschriebenen Filter handelt es sich um ein rekursives zeitliches Filter erster Ordnung, das die Filterwirkung an die Bewegung im Bild adaptiert. Die Bewegungserkennung und Filterung erfolgt in einer Auflösungshierarchie. Im Folgenden wird das Filter als multiskalares Rauschglättungsfilter mit Bewegungsdetektion bezeichnet. Die wesentlichen Schritte der Filterung sind die Dekomposition in eine Laplace-Auflösungshierarchie [4] und eine kantenerhaltende Rauschglättung auf variablen Stufen der Laplace-Pyramide.

3.1 Laplace-Auflösungshierarchie

Die Laplace-Pyramide ist ein wesentliches Konzept in der Bildverarbeitung [4] und wird im beschriebenen Algorithmus zur Dekomposition verwendet. Die anschließende Ortsfilterung dient einerseits als Vorverarbeitungsschritt für die Bewegunsdetektion und andererseits als Ergänzung zum Temporalfilter. Bei der Verwendung von GPUs zur Beschleunigung rechenintensiver Anwendungen muss Folgendes beachtet werden: Teure Zugriffe auf den globalen Grafikspeicher, die jeweils 400 bis 600 Taktzyklen (vgl. [5], S.49) benötigen, können durch den Einsatz von Texturen (Texturcache) vermieden werden. Zugriffe auf den Texturcache erreichen dabei eine ähnliche Latenz wie auf Register. NVIDIAs Speicherverwaltungseinheit unterstützt das Lesen von 32-, 64- oder 128-bit Werten, wobei die maximale Datenrate beim Lesen von 32-bit Werten erreicht wird. In unserem Fall jedoch liefert das Akquisitionsgerät 16-bit breite Daten. Die Geschwindigkeit des Reduktions- und des Expansionsschrittes ist auf Grund der geringen Anzahl an arithmetischen Befehlen durch die Speicherbandbreite limitiert. Dies entspricht einem niedrigen Verhältnis von Rechenoperationen pro Speicherzugriff.

Beide Probleme werden durch die folgende Strategie gelöst: Durch das Zusammenfassen von zwei oder vier, in x-Richtung nebeneinanderliegenden Bildpunkten werden Speicheroperationen optimiert. Dies führt im Fall von 16-bit

Werten zu Speicherzugriffen von nunmehr mindestens 32-bit, wodurch in einem Thread die optimale Speicherbandbreite ausgenutzt und das Verhältnis von Rechenoperationen pro Speicherzugriff gesteigert wird.

3.2 Varianzgesteuertes Glättungsfilter

Das varianzgesteuerte Glättungsfilter beschreibt die Tiefpassfilterung eines Bildpunktes entlang der Richtung, die bezüglich einer lokalen Nachbarschaft die minimale Varianz aufweist. Dadurch wird eine kantenerhaltende Rauschreduktion erreicht. Obwohl die Glättungsrichtung und die der minimalen Varianz übereinstimmen, muss die lokale Nachbarschaft des Glättungskerns nicht notwendigerweise der Nachbarschaft der Varianzberechnung entsprechen.

Analog zur Implementierung der Laplace-Pyramide werden hier auf Grund der lokalen Speicherzugriffe Texturen eingesetzt. Die relativ zum Mittelpunkt definierten Richtungen können aus Effizienzgründen im Constant Memory abgelegt werden, wodurch Mehrfachzugriffe pro Richtung effizient den Constant Cache nutzen.

Eine weitere Optimierung ist das Entrollen von Programmschleifen (loop unrolling). Obwohl CUDA hierfür die Compileranweisung *#pragma unroll* zur Verfügung stellt, wurden die vorkommenden verschachtelten Schleifen in unserem Fall nicht vollständig entrollt. Durch den Einsatz von (nicht offiziell unterstützter) Template-Metaprogrammierung konnten wir schleifenähnliche Strukturen nachbilden, die in jedem Fall entrollt werden.

Diese definitive Entrollung der Schleifen ermöglicht einen weiteren Optimierungsschritt: Die relativ zum Mittelpunkt definierten Richtungen der Varianzberechnung können nunmehr im Code als *switch*-Block realisiert werden, wodurch der Zugriff auf den Constant Memory für die Berechnung entfällt.

3.3 Multiskalares Rauschglättungsfilter mit Bewegungsdetektion

Ein Bewegungsdetektor kann durch den Vergleich zwischen den verschiedenen Stufen der Laplace-Hierarchie von $n-1$ bereits gefilterten Bildern ($s_{\text{filtered}}^{(n-1,2,..)}$) und dem aktuellen Bild effizient realisiert werden. Durch die Kombination einer Laplace-Pyramide, eines varianzgesteuerten Glättungsfilters und eines Bewegungsdetektors erhält man das multiskalare Rauschglättungsfilter mit Bewegungsdetektion (vgl. Abbildung 1). Dieses ermöglicht eine Rauschreduktion durch die Filterung der oberen Bandpassstufen mit dem varianzgesteuerten Filter, zusätzlich kombiniert mit der zeitlichen Änderung der Bildfolge.

Da der Registerverbrauch dem der optimierten Laplace-Pyramide ähnlich ist, können effizient mehrere Werte auf einmal verarbeitet werden.

4 Evaluierung und Ergebnisse

Aufgrund der Komplexität wurde die Optimierung des varianzgesteuerten Glättungsfilters priorisiert. Durch den Einsatz von Fließkommaarithmetik entstehen

geringe Werteabweichungen zwischen der Referenz und der auf der GPU berechneten Bilder. Insgesamt entsteht dabei im Beispielbild eine durchschnittliche Abweichung aller Werte von lediglich $7.2 \cdot 10^{-6}$ für einen Wertebereich von 0 bis 4095. Die fehlerhaften Werte weichen im Schnitt um 1.27% vom Ergebnis der Referenzimplementierung ab.

Das multiskalare Filter hingegen zeigt mit einem durchschnittlichen Fehler von etwa 2.78% stärkere Abweichungen, was wiederum auf die rekursive Struktur des Filters zurückzuführen ist. Zusätzlich entsteht durch die Berechnung mit Fließkommazahlen auf der GPU ein numerischer Unterschied, der jedoch aufgrund der höheren Genauigkeit näher am theoretischen Ergebnis liegt.

Die unterschiedlichen Berechnungszeiten der einzelnen Stufen des multiskalaren Filters sind in Tabelle 1 aufgezeigt. Hierbei verdeutlicht der Vergleich zwischen einer optimierten CPU-Implementierung und der von uns vorgestellten CUDA Implementierung, gemessen auf der für unsere Tests verwendeten Grafikkarte NVIDIA GeForce 8800 GTX, die deutlich höhere Rechenleistung. Darüberhinaus liegt die Gesamtperformance mit 7.24 ms in einem Bereich, der einen Echtzeitbetrieb ermöglicht!

5 Zusammenfassung und Ausblick

Mit der Einführung von mehrkernbasierten GPUs, wie z.B. NVIDIAs G80, ist es möglich die Anforderungen für Hochleistungsfluoroskopiesysteme in Echtzeit zu erfüllen. Dabei haben wir bezüglich des implementierten Filters einen Geschwindigkeitsvorteil von Faktor 10 über den gegebenen Anforderungen erreicht.

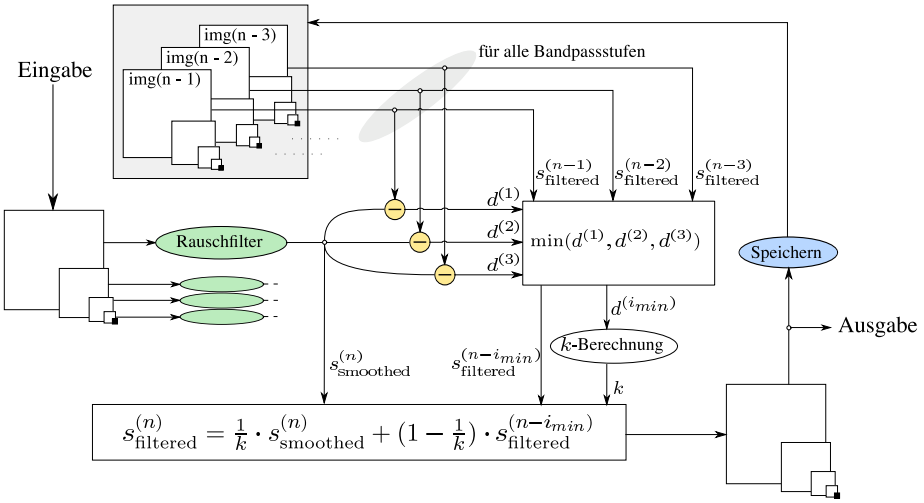

Abb. 1. Verarbeitung der Bandpassbilder mit dem multiskalaren Rauschglättungsfilter; k-Faktor steuert den temporalen Einfluss.

Tabelle 1. Gemessene Zeiten jedes Filterschrittes in ms bzw. Beschleunigungsfaktor zwischen Singlecore und GPU Implementierung.

	Glättungsfilter	Konstruktion Laplace	Rekonstruktion Laplace	Multiskalarer Filter	Kopieren CPU → GPU	Kopieren GPU → CPU	Gesamt
Xeon 2.66 1-Kern	181.59	18.38	15.99	115.35			**331.31**
Xeon 2.66 2-Kern	91.57	9.88	7.91	58.94			**168.30**
GeForce 8800 GTX	3.90	0.86	0.48	0.74	0.60	0.66	**7.24**
Speedup	×46.6	×21.4	×33.3	×155.9			×**45.8**

Ein weiteres wichtiges Ergebnis erzielten wir durch die Technik der Schleifenentrollung in der Optimierung. Die Verwendung des erweiterten Programmierkonzeptes für Mehrkernarchitekturen (CUDA) und deren 32-bit Gleitkommaverarbeitung kann die zukünftige Entwicklung von medizinischen Echtzeitanwendungen stark vereinfachen. Allerdings müssen interventionelle Röntgensysteme zusätzlich Netzwerkübertragungen, die Archivierung und verschiedene DICOM-Dienste parallel mit einer festen Latenz verarbeiten. Die stabile Hochleistungsverarbeitung eines Bildstroms in Kombination mit einem stark belasteten Prozessor und diesbezüglichen Betriebssystemanforderungen muss daher noch gezeigt werden.

Literaturverzeichnis

1. Mueller K, Yagel R. Rapid 3D cone-beam reconstruction with the algebraic reconstruction technique (ART) by utilizing texture mapping graphics hardware. IEEE Nuclear Science Symposium. 1998;3:1552–1559.
2. Mack M, Hornegger J, Paulus D, et al. Echtzeit-Röntgenbildverarbeitung mit Standardhardware. Proc BVM. 2004; p. 395–399.
3. Scherl H, Keck B, Kowarschik M, et al. Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA). In: Frey EC, editor. Nuclear Science Symposium, Medical Imaging Conference 2007; 2007. p. 4464–4466.
4. Burt PJ, Adelson EH. The Laplacian pyramid as a compact image code. IEEE Trans Comm. 1983;31:532–540.
5. NVIDIA. NVIDIA CUDA - Compute Unified Device Architecture - Programming Guide. 2701 San Tomas Expressway, Santa Clara, CA 95050 (USA); 2007.