Fachhochschule Köln,
Campus Gummersbach

Optimierung eines neuen Logarithmic-Search-Verfahrens zum Image Mosaicing unter Einsatz des CUDA-Frameworks

03.06.2009

Eugen Sewergin, B. Sc.

Erstprüfer: Prof. Dr. Wolfgang Konen
Zweitprüfer: Prof. Dr. Horst Stenzel

Agenda

- Zielsetzung
- Performancemessung
- Optimierungsschritte
- Parameter und Qualität
- Konzept und Portierung in CUDA
- CPU VS GPU Ergebnisse
- Fazit

Zielsetzung

CPU:

- Performanceoptimierung
- Optimierung der algorithmischen Qualität

GPU:

- Beschleunigung der rechenintesiven Teile des Algorithmus mittels CUDA

3

•Optimierung des neuen Logarithmic-Search-Verfahrens mittels CPU und GPU

•Hauptfrage: “Wie gut gelingt es, typische Algorithmen der Bildverarbeitung auf ein paralleles Framework abzubilden?”

Performancemessung

Grobe Messung:

- Eclipse Plugin „TPTP Profiler“
  - Ca 44,9 % per Thread nimmt die Berechnung der Kreuzkorrelationskoeffizienten ein

Feine Messung:

- System.nanoTime Methode

4

•44,9% = 7,72% (Holen der Pixelwerte) + 3,98% (getWidth Methode) + (15,91% (Kreuzkorrelationskoeffizient) + 17,29%(XMean Methode))

Siehe MA S.33 Abbildung 3.1

Optimierungsschritte

Optimierungsschritt 1:

- Integration der Methode xMean in die Methode crossCorrelation

- Mittlere Beschleunigungsfaktoren
  - Pentium M 1,4 GHz: ca 1,35
  - Athlon64 X2 2,5 GHz: ca 1,54

5

•Die Beschleunigung ist abhängig von der Seitenlänge eines Templates und der verwendeten CPU

Siehe MA S.42 Abbildung 3.2 und 3.3

Optimierungsschritte

Optimierungsschritt 2:

- Überführung des Java-Code in C

- Mittlere Beschleunigungsfaktoren
  - Pentium M 1,4 GHz: ca 2,42
  - Athlon64 X2 2,5 GHz: ca 1,97

6

•Die Beschleunigung ist abhängig von der Seitenlänge eines Templates und der verwendeten CPU

•Trotz eines kleinen Overhead, der durch das Schleusen der Daten über JNI entsteht, kann sich das Ergebnis der Sprache C gegenüber Java sehen lassen.

Siehe MA S.42 Abbildung 3.2 und 3.3

Optimierungsschritte

Optimierungsschritt 3:

- Verwendung der schon zuvor berechneten Kreuzkorrelationskoeffizienten

- Mittlere Beschleunigungsfaktoren
  - Testsequenz - Storz parietal2: ca 1,38
  - Testsequenz - Versuch 1 klein: ca 1.61

7

•Die Anzahl der Berechnungen variiert je nach Template-Größe und hängt lediglich von den Eigenschaften einer Testsequenz ab, nicht von der verwendeten CPU.

Siehe MA S.45 Abbildung 3.4 und 3.5

Parameter und Qualität

Parameter „ctresh“ und „rsz“

- Testsequenz „*Storz parietal2*“:
  - Beste Güte bei einem Template von 25x25 Pixel
- Testsequenz „*Versuch 1 klein*“:
  - Bestmögliche Panoramabild bei 31x31 Pixel großem Template
- Hohe Güte des Panoramabildes über verschiedene Template-Seitenlängen hinweg bei ctresh=0,85

8

•Im ImageJ implementiert und benutzt wurde das Vergleichsmaß auf Basis des Kreuzkorrelationskoeffizienten nach Bloemendal (Boundingbox mit 5*5 Templates)

•rsz - stellt die Seitenlänge eines quadratischen Templates dar.

•cthresh - ist ein Grenzwert, der den minimalen zulässigen Wert bei einem Kreuzkorrelationskoeffzient eines Bewegungsvektors angibt. Das heißt, dass alle Bewegungsvektoren mit einem Kreuzkorrelationskoeffzient unter diesem Wert nicht in die Berechnung des Panoramabildes einfließen.

•Im Template-Seitenlängenbereich von 25 Pixel (Testsequenz halbe PAL Auflösung), 31 Pixel (Testsequenz PAL Auflösung) ergeben sich die bestmöglichen Panoramabilder, rechnerisch aber auch vom visuellen Eindruck her.

Parameter und Qualität

Parameter „NPTS“ und „rsz“

- Testsequenz „*Storz parietal2*“:
  - Hohe Güte bei Templates zwischen 19 und 23 Pixel pro Seitenlänge
  - NPTS-Parameterwert 6 liefert eine hohe Güte
- Testsequenz „*Versuch 1 klein*“:
  - Qualitatives Panoramabild bei 31x31 Pixel
  - NPTS-Parameterwert 8 liefert eine hohe Güte

9

•Im ImageJ-Plugin implementiert und benutzt wurde das Vergleichsmaß auf Basis des Kreuzkorrelationskoeffizienten nach Bloemendal (Boundingbox mit 5*5 Templates)

•rsz - stellt die Seitenlänge eines quadratischen Templates dar.

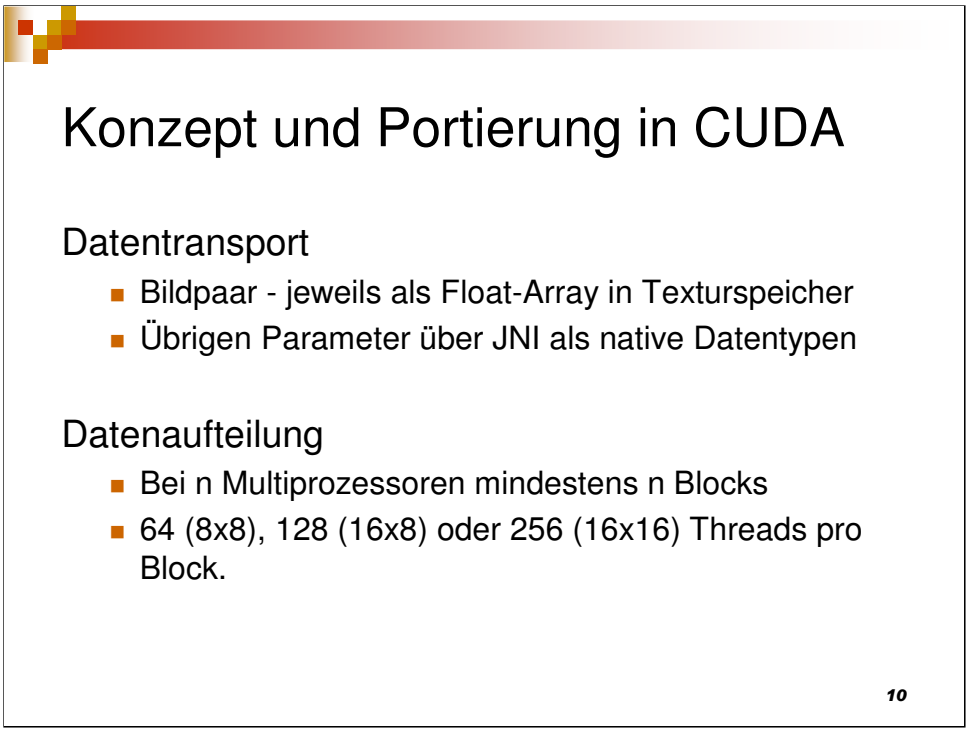
•NPTS - bestimmt die Anzahl der Templates in einem Referenzbild, die im aktuellen Bild mittels Logarithmic-Search gesucht werden. Diese Anzahl wird quadriert

und über das ganze Referenzbild gleichmäßig verteilt.

•Im Template-Seitenlängenbereich von 19 bis 23 Pixel (Testsequenz halbe PAL Auflösung), 31 Pixel (Testsequenz PAL Auflösung) ergeben sich die bestmöglichen Panoramabilder, rechnerisch aber auch vom visuellen Eindruck her.

•Testsequenz halbe PAL Auflösung: NPTS-Parameterwert 6

•Testsequenz PAL Auflösung: NPTS-Parameterwert 6



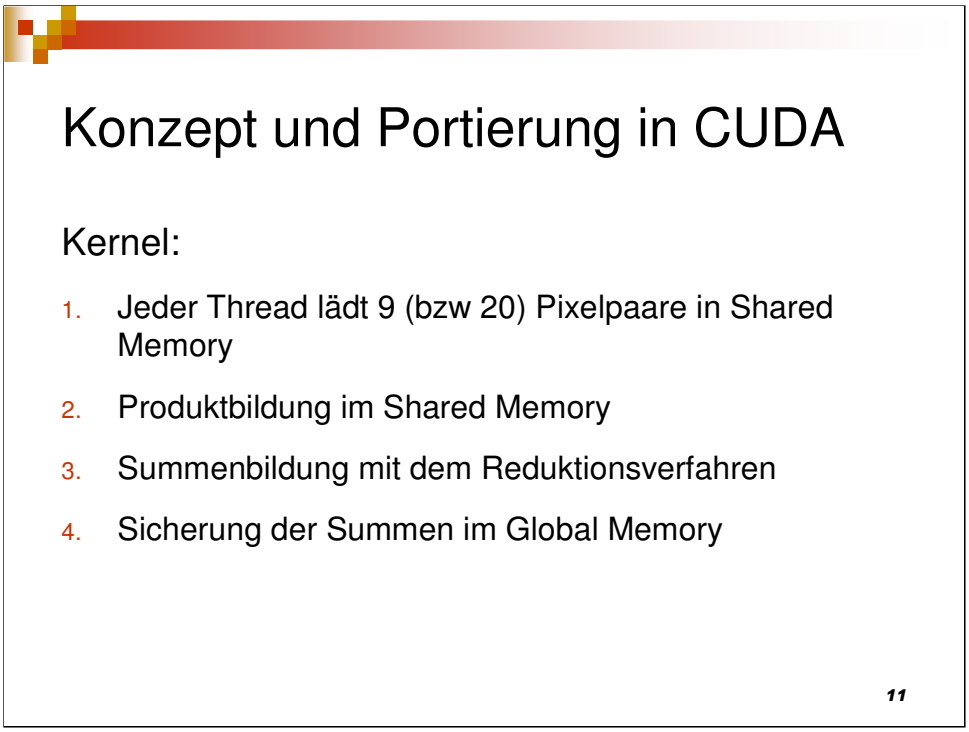
- ImageJ-Plugin Methode „crossCorrelation“, in der ein Kreuzkorrelationskoeffizient berechnet wird, ist am rechenintensivsten

Datentransport:

- Pixelwerte der Bilder werden als eindimensionale Float-Arrays aus den Objekten vom Typ ImageProzessor extrahiert. Die passende Methode hierzu stellt ImageProzessor schon bereit.
- Um hohe Performance auf dem Device zu erzielen, werden neun Kreuzkorrelationskoeffizienten parallel berechnet. Dazu müssen auf der Java-Seite die nötigen Daten gesammelt und an den Kernel weitergeleitet werden.

Datenaufteilung:

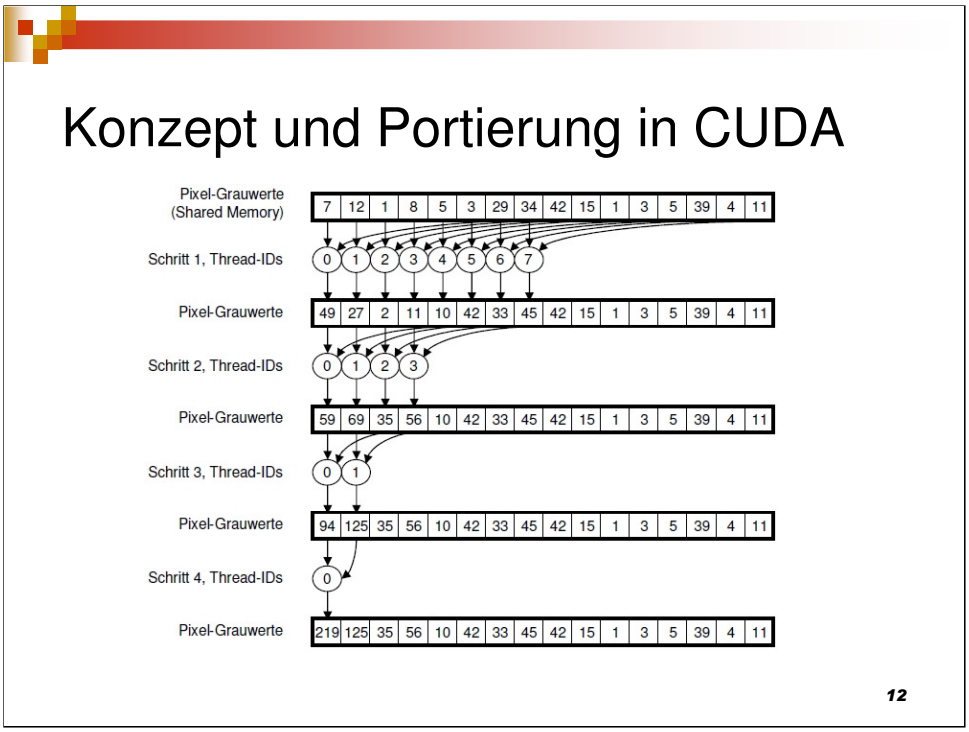
- Eine effektive Nutzung aller zur Verfügung stehenden Multiprozessoren einer Grafikkarte kann nur durch eine optimale Aufteilung der Arbeit in Thread-Blöcke und der Verwendung von mehr Blöcken als MPs erreicht werden. NVIDIA empfiehlt mindestens 64 Threads pro Block zu verwenden, besser jedoch 192 und mehr Threads pro Block, um die Latenzzeit bei Lese- und Schreibzugriffen auf den Grafikkartenspeicher zu verdecken. Nach NVIDIA richtet sich die Wahl der Blockgröße je nach Problemstellung und muss nicht immer die höchste Anzahl an Threads pro Block betragen.
- Hardwaremäßige Grenzen: 512 Threads pro Block, 8192 Register pro Multiprozessor (1024 Register pro Prozessorkern)



- Die schnellste Berechnung der Kreuzkorrelationskoeffizienten war bei 64 Threads pro Block (8x8) zu verzeichnen
- Während der Ausführung eines Kernels arbeitet jeder Thread jedes Blocks im Grid dasselbe Programm ab. Dabei besitzt jeder Thread eine Thread-ID und eine Block-ID, mit dessen Hilfe eine Iterationsvariable erzeugt werden kann, um die doppelte for-Schleife bei der Kreuzkorrelationskoeffizienten-Berechnung zu simulieren und parallel abzuarbeiten.
- Minimierung der Latenzzeit durch Verwendung von Shared Memory

Kernel:

1. 9 (bzw 20) Pixel-Paare benötigt, da gleichzeitig 9 (bzw 20) Kreuzkorrelationskoeffizienten je nach Kernel zu berechnen sind. Pro Thread anstatt 18 nur 10 Ladevorgägne nötig (bzw pro Thread anstatt 40 nur 21 Ladevorgänge nötig)
2. Jeder Thread führt 29 (bzw 62) Produktberechnungen mit den zuvor in Shared Memory gespeicherten Grauwerten durch
3. Insgesamt 29 (bzw 62) Summen zu bilden
4. Abspeicherung der Summen aus dem Shared Memory in dem Global Memory, worauf der Host einen Zugriff besitzt



Ein Beispiel des Reduktionsverfahrens mit 16 Werten (halber Warp)

•Alle arbeitenden Threads sind sequenziell angeordnet und addieren zwei Werte aus nicht benachbarten Bereichen. Dabei entstehen keine Bank-Konikte im Shared Memory. Wie man dem Beispiel entnehmen kann, wird die Summe eines Arrays mit 16 Werten (2 hoch 4) parallel in 4 Schritten berechnet, im Gegensatz dazu wären bei einer sequenzieller Summenbildung 15 Schritte nötig. Daraus entgeht, dass beim Reduktionsverfahren auf einem Array mit 2 hoch n Werten n Schritte bis zum Endergebnis notwendig sind.

CPU VS GPU Ergebnisse

	Rechenzeiten pro Kreuzkorrelationskoeffizient in Nanosekunden					
Template-Seitenlänge in Pixel	Pentium M 1,4 GHz	Athlon64 X2 2,5GHz	8600GT (4 Multiprozessoren)		8800GT (14 Multiprozessoren)	
log.rsz	1,4GHz_JNI C	2,5GHz_JNI C	4MP_9T_pro_Kernel	4MP_20T_pro_Kernel	14MP_9T_pro_Kernel	14MP_20T_pro_Kernel
15	4048	2019	9486	4647	6800	3487
23	6515	3280	12457	10488	6891	3576
31	10112	5127	12742	13431	7083	5652
39	14459	7439	18545	25164	7122	13506
47	19704	9978	41501	30588	26486	15649
63	31865	16153	51431	44111	28891	20078
95	73592	36400	81801	105298	36682	33100
127	128940	63721	123887	187715	46986	50583
159	200105	102383	176789	287928	59826	72704
191	287619	142093	251560	397501	75398	102831
255	528275	260292	421029	749132	115847	174029

13

•In den CPU-Berechnungszeiten spiegelt sich die ca. zweifache Verdoppelung der Rechenleistung in GHz wieder

•Bei der GPU sind die Faktoren relevant: Rechenleistung, Anzahl MPs, Blöcke pro Template, Ressourcen eines Multiprozessors (8 KB Register, 16 KB Shared Memory)

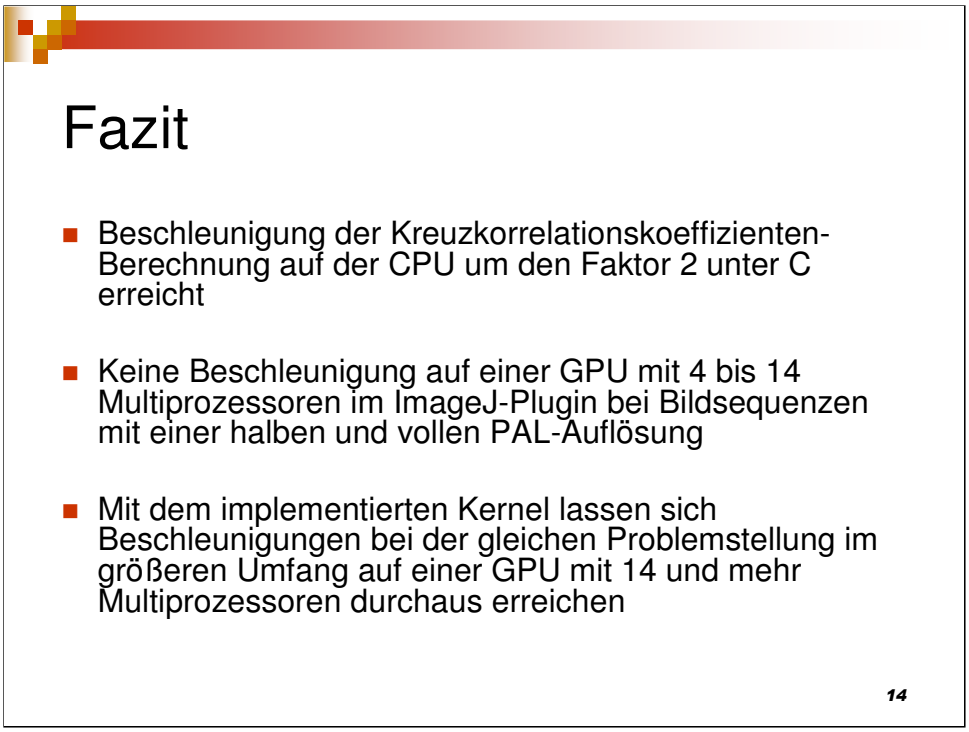
•Kernel mit 20 Kreuzkorrelationskoeffizienten-Berechnungen 15872 Byte an Shared Memory pro Block = 1 Block pro Multiprozessor aktiv (64 Treads)

•Kernel mit 9 Kreuzkorrelationskoeffizienten-Berechnungen 7424 Byte an Shared Memory pro Block = bis zu 2 Blöcke pro Multiprozessor aktiv (128 Treads)

•Ca ab 95x95 Pixel pro Temlate überholt die Grafikkarte mit 14 MPs die 2,5GHz CPU,

•Bei 255x255 Pixel großem Template gibt es bei der GPU mit 14 MPs einen Beschleunigungsfaktor von ca. 2,24 gegenüber einer 2,5 GHz CPU und bei ca. 4,56 gegenüber einer 1,4 GHz CPU

•Neuste Generation von GPUs 30MPs bzw 2x30MPs => theoretisch Verdopplung bzw Vervierfachung der Beschleunigung, müsste allerdings geprüft werden.



•Bei Zuhilfenahme einer GPU konnte auf Grund der hohen Latenzzeiten beim Speicherzugriff und der geringen Ressourcen in einem Multiprozessor keine Beschleunigung für die spezielle Problemstellung im ImageJ-Plugin erzielt werden.

•Die Beschleunigungsfaktoren von Kreuzkorrelationskoeffizienten-Berechnungen auf der GPU fallen im Vergleich zu Matrixberechnungen, die Ähnlichkeiten in der Problemstellung und deren Lösung aufweisen, relativ niedrig aus. Das liegt vor allem daran, dass bei den Kreuzkorrelationskoeffizienten-Berechnungen das Fünffache an Shared Memory verwendet wird und somit der Spielraum für mehr aktive Threads pro Multiprozessor eingeschränkt ist.