Praktische Übungen zu Computertechnik 2

Versuchsprotokoll

Versuch:

C2 – Parallelrechner

Versuchsdatum und -zeit:

Donnerstag, 03. Juni 2010, 10-13 Uhr

Betreuer:

Adrian Knoth

Name, Studiengang, Mat.-Nr.:

Ralf Wondratschek, B. Sc. Informatik, 112626

Email:

ralf.wondratschek@uni-jena.de

Name, Studiengang, Mat.-Nr.:

Kerstin Gößner, B. Sc. Informatik, 114656

Email:

kerstin.goessner@uni-jena.de

Vom Betreuer auszufüllen:

Vorbereitung/Kolloquium:

Durchführung:

Protokoll:

Gesamtbewertung:

Gliederung

1. Vorbereitung

Um serielle Berechnungen von Computern zu beschleunigen, ging man in den letzten Jahren in der Entwicklung mehrere Wege. Zum einen beschleunigen andere Geräte, wie die Grafikkarte, die Rechenleistung und zum anderen gehören MultiCore-Prozessoren zum Alltag. In der Klassifikation von Michael J. Flynn gehören diese mehrkernigen Prozessoren zur MIMD Architektur, das heißt mehrere Befehle arbeiten zeitgleich auf verschieden Daten. Bei einer weiteren Entwicklung handelt es sich um zusätzliche Vektor-Einheiten im Prozessor. Sie fassen bei einer gleichen Operation mehrere Daten zu einem Vektor zusammen und berechnen das Ergebnis. Bei AltiVec handelt es sich zum Beispiel um eine solche SIMD Einheit von Motorola und IBM. Mit den Streaming SIMD Extensions führte Intel beim Pentium III eine Befehlssatzerweiterung ein um Programme durch Parallelisieren zu beschleunigen.

Besonders effektiv sind die Vektoreinheiten in der Bild- und Videoverarbeitung, da hier pro Pixel in der Regel dieselbe Operation durchgeführt wird. Ziel dieses Versuches ist eine Videoschnittsoftware in C zu erstellen, die auf SIMD Befehle zurückgreift und somit die Videos schneller rendert.

2. Vorgehensweise

Bei dem Versuch sind zwei Videoszenen zur Verarbeitung gegeben, um die entwickelte Schnittsoftware zu testen. Dabei soll mit einem schwarzen Bild für zwei Sekunden begonnen werden, welches über weitere 2 Sekunden in das Video A überblendet. Nachdem Video A volle zwei Sekunden lief, soll eine weitere Überblendung in Video B innerhalb von vier Sekunden erfolgen. Video B läuft auch zwei Sekunden und wird danach wieder in ein schwarzes Bild überblendet. Nach noch einer Sekunde wird das Video beendet.

Für die Software sind im Vorbereitungsmaterial bereits einige Hilfsfunktionen und Konstanten gegeben, die das Programmieren erleichtern sollen.

3. Erprobung

Die erste Aufgabe, ein zwei Sekunden langes schwarzes Video zu erstellen, war mit den Hilfsmethoden leicht lösbar. Mit der Methode black_image(outputframe); bekommt die Variable outputframe ein schwarzes Bild zugewiesen. Durch die darauffolgende Methode wird outputframe für zwei Sekunden lang gespeichert.

Um den Übergang von schwarz zu Video A zu erzeugen, sind 2 Schleifen notwendig. Die erste dient für jeden Frame (insgesamt 50 Frames, da der Übergang zwei Sekunden lang sein soll). Die zweite innere ist nötig, um die einzelnen Pixel von dem jeweiligen Frame zu berechnen. Bei der skalar Variablen handelt es sich um den Faktor für die Sichtbarkeit von Video A. Dieser erhöht sich innerhalb der 50 Frames von 0 zu 1 in 0,02er Schritten. Das Ergebnis von dem jeweiligen Frame berechnet sich dann aus skalar mal Video A. Das Besondere bei der Berechnung ist die Benutzung der SSE-Befehle. Deshalb müssen auch die Variablen skalar, floatvektor und ergebnis im __m128 Format deklariert sein. Nachdem jeder Frame berechnet wurde, wird er noch gespeichert.

Die nächste Aufgabe, Video A zwei Sekunden laufen zu lassen, war wieder relativ einfach zu lösen. Wie bei dem Übergang haben wir zwei for-to-do Schleifen verwendet über die Länge von 50 Frames. Der nächste Frame aus Video wird geladen, das Ergebnis wird berechnet, was im dem Fall einfach der Frame von Video A ist, und mit dem outputframe gespeichert. Sehr wahrscheinlich hätte man auch die zweite for-to-do Schleife weglassen können, da der Frame von Video A nicht verändert bzw. manipuliert wird, diese Variante funktionierte allerdings auch, kostet nur mehr Performance.

Nun kommt der schwierigste Teil des Programmes: der vier Sekunden lange Übergang von Video A zu Video B. Mit der Formel aus den Vorbereitungsunterlagen war allerdings eine gute Hilfe gegeben.

$$\text{Outputframe} = \alpha * \text{inputframeA} + \beta * \text{inputframeB}$$

Folglich brauchten wir auch zwei Variablen für die skalar Werte. Über die Dauer von 100 Frames muss skalarA von 1 auf 0 in 0,01 Schritten fallen und skalarB von 0 auf 1 in 0,01 Schritten steigen. Das bedeutet die Deckkraft von Video A nimmt umgekehrt proportional zur Deckkraft von Video B ab. Den restlichen Teil kann man aus der Formel ableiten: die Frames aus beiden Videos werden geladen, mit dem entsprechenden skalar multipliziert, die Ergebnisse von beiden Multiplikationen addiert und dieser berechnete Frame wird gespeichert.

Video B über zwei Sekunden zu zeigen, ist äquivalent zu betrachten wie bei der vorhergehenden Teilaufgabe Video A zwei Sekunden zu zeigen.

Dasselbe gilt für den Übergang von Video B zu einem schwarzen Bild. Es handelt sich um das gleiche Prinzip wie bei den Übergang von schwarz zu Video A. Der einzige Unterschied besteht im skalar, es steigt nicht von 0 auf 1 an, sondern fällt von 1 auf 0 (also die Deckkraft fällt von 100% auf 0%).

Nun galt es noch ein schwarzes Bild für eine Sekunde zu zeigen, was mit den beiden Hilfsmethoden wieder einfach lösbar war.

Den kompletten Quellcode findet man auch im Anhang auf Seite A1 und A2.

Unsere verwendeten SSE-Befehle findet man auch im Assemblercode wieder (siehe Seite A3 bis A5). So sind die Funktionen __m128 _mm_mul_ps(__m128 a, __m128 b)) (entspricht mulps) und __m128 _mm_add_ps(__m128 a, __m128 b) (entspricht addps) eindeutig identifizierbar. Aus dem Zusammenhang lässt sich auch auf Seite A4 bei .L6: erkennen, dass movaps für die Funktionen __m128 _mm_load_ps(float * p) und void _mm_store_ps(float * p, __m128 a) stehen. Bei .L6: wird in der 3. und 4. Zeile die Multiplikation und danach gleich die Addition aufgerufen, was im Quelltext „Aufgabe 4: Crossfade nach Video B über 4 Sekunden“ (in der 2. for-to-do Schleife) entspricht. Davor werden die Ladefunktion und danach die Speicherfunktion aus den SSE-Befehlen aufgerufen, wodurch dieser Schluss möglich ist. Eine Änderung der arithmetischen SSE-Befehle durch binäre Operationen (Seite A6 und A7) bewirkt keine Änderung am Assemblercode.

4. Schlussfolgerungen

Der Versuch hat gezeigt, dass das Einbinden von SSE-Befehlen keine großen Umstände bedeutet. Der Nutzen, der daraus für die Performance gezogen werden kann, könnte vielen Programmen einen Leistungsschub bringen. So findet man sogar in Textverarbeitungsprogrammen Möglichkeiten, den Kontrast und die Helligkeit bei den eingefügten Bildern zu ändern. Für professionelle Video- und Bildbearbungssoftware sind diese Befehle ein Muss, da die Standards der Bilder und Videos mit der Zeit sich ständig erhöhen. Zum einen wird nach immer höheren Auflösungen getrachtet und zum anderen sollen sich die Farben für jeden Pixel verbessern, zum Beispiel durch mehr Farbkanäle oder mehr Speicher pro Farbkanal. Der Nutzen von parallelen Berechnungen wird daher immer weiter steigen.