Bericht
KU Rechnerorganisation
2016

(Graz)

Gruppe „ulbel22“

Gerald Birngruber (birngruber@student.tugraz.at) 1530438

Thomas Finsterbusch (finsterbusch@student.tugraz.at) 1331968

Volker Seiser (v.seiser@student.tugraz.at) 1430356

1 Inhalt

2 Einleitung und Aufgabestellung

In diesem Report werden die Ergebnisse der Aufgabe 4 für Rechnerorganisation (VO:705.036 und KU: 705.037) zusammengefasst. Die Aufgabe bestand aus mehrere Teilaufgaben. Diese beschäftigten sich mit dem Stack und der Umsetzung eines solchen für die Toy-CPU und die Erweiterung und Verbesserung des Codes und des Simulationsmodells.

Die Aufgaben (Posch, 2016):

- Entwicklung eines Quicksortalgorithmus mit Stack für die Toy - CPU welche Werte aus dem Standard Input einliest und diese sortiert wieder ausgibt.
- Vergleich der Laufzeiten des Quicksortalgorithmus und des Bubblesortalgorithmus gemessen in Taktzyklen.
- Erstellen eines Bash - Scrips welches den erstellten Quicksort Code und den mit ATOY optimierten Code testet. Bestimmung der maximalen Anzahl an Werten die sortiert werden können.
- Implementierung eines neuen Adressierungsmodus für die STOY CPU anhand des Verilog - Modelles welche die ursprünglichen Befehlen JL und JR ersetzen. Durch diese Implementierung soll der Quicksort Algorithmus verbessert und optimiert werden.

3 Zusammenfassung

- Das erstellte Quicksortprogram für S-Toy funktioniert. Es ist in der Lage bis zu 29 Werte zu sortieren. Im schlechteste Fall können immer noch bis zu 13 Werte sortiert werden.
- Die erstellten Bash-Scripts sind in der Lage die Quicksortprogramme zu testen. Dabei wird die maximale Länge im Worstcasefall bestimmt und die maximale Länge bei Zufallswerten. Es wird mit 1 Wert begonnen und immer um 1 Wert mehr an den Standardinput gegeben, bis ein überlauf produziert wird und das Programm abstürzt.
- Im Verilogmodell wurden die Befehle JL und JR gegen die Befehle STX und LDX ausgetauscht. Dadurch ist es möglich die Länge des Maschinencodes zu verringern, da aus 2 Instruktionen eine gemacht wurde.
- Durch die Verbesserungen im Verilogmodell wurde der Maschinencode angepasst und optimiert. Dadurch war es möglich die Anzahl an sortierbaren Werten zu erhöhen. Nach den Verbesserungen konnten im Worstcase Fall 15 Werte sortiert werden. Mit Randomwerten konnten zwischen 20 und 30 Werte sortiert werden.
- Die Laufzeit der Programme ist wie aus der Theorie bekannt bei Quicksort geringer als bei Bubblesort ausgefallen. Bis zu 10 Werten ist der Unterschied sehr gering und kaum messbar. Bei der Maximalen Anzahl an Werten ist bereits eine um ca. 25% verkürzte Laufzeit festzsutellen.

4 Ziel der Aufgabe

Ziel der Aufgabe ist es die Verwendung des Stacks zu verstehen indem dieser selbst programmiert wird. Dabei werden Aufbau und Abbau des Stacks durch Prolog und Epilog erlernt. Im Weiteren wird durch erstellen des Bash – Skripts der Umgang mit der Shell geübt und vertieft. Im letzten Schritt wird das Verständnis für die CPU durch das Hinzufügen eines neuen Befehles verbessert. Da dieser neuen Befehle jedoch nicht vom herkömmlichen Compiler erkannt werden muss der ASM Code selbstständig in Maschinensprache umgewandelt werden.

5 Abgegebene Dateien

- ulbel22_STOY.asm: Assemblercode des implementierten Quicksort Codes.
- ulbel22_STOY.v: Verilogmodell der TOY-CPU mit Stack(STOY) auf der getestet wurde.
- ulbel22_STOYcode.toy: Quicksort Maschinencode für das STOY-Verilogmodell.
- ulbel22_STOY.sh: Bash Script zum Testen des Quicksortcodes mit Zahlenfolgen im Worstcase und im Randomcase.
- ulbel22_ATOY.asm: Assemblercode des mit LDX und STX optimierten Quicksort Codes.

- ulbel22_ATOY.v: Verilogmodell der Advanced TOY-CPU(ATOY) in der LDX und STX implementiert wurden.
- ulbel22_ATOYcode.toy: Optimierter Quicksort Maschinencode für das ATOY-Verilogmodell.
- ulbel22_ATOY.sh: Bash Script zum Testen des mit LDX und STX optimierten Quicksortcodes mit Zahlenfolgen im Worstcase und im Randomcase.
- ulbel22_DOK.pdf: Report

6 Bash-Script

Die Bash ist eine Unix Shell. Sie ist in vielen UNIX Systemen die Standard Shell (Wikipedia_Bash, 2016). In einem Bash - Skript können wie in der Linux Konsole Befehle ausgeführt werden. Des Weiteren besteht auch die Möglichkeit Variablen zu definieren und mit diesen zu arbeiten. Für Variablen gibt es jedoch anders als in C o.ä. keine Typisierung, was bedeutet, dass ein gespeicherter Wert als alles interpretiert werden kann, er hat keinen festen Datentype. Um ein Bash – Skript ausführbar zu machen muss mit dem Befehl chmod ausführbar gemacht werden.
Übersicht der verwendeten Befehle:

- #! /bin/bash: Die Shebang wird in der ersten Zeile des Scripts geschrieben. Damit wird sichergestellt das die folgenden Befehle mit der richtigen Shell aufgerufen werde.
- Echo: Ausgabe von Text auf der Konsole
- If then fi: Wie in anderen Programmiersprachen kann auch in einer Skriptsprach eine if Bedingung formuliert werden. Die ein der if – Bedingung stehenden Befehle werden ausgeführt, wenn die Condition wahr ist.
- Schleifen: Im Skript können auch for und while – Schleifen verwendet werden. Um diese besser steuern zu können gibt es auch die Befehlen break und continue mit ihren bekannten Bedeutungen.
- rm: Remove: der Removebefehl löscht die Datei die angegeben wurde.
- Printf: Für eine formatierte Ausgabe im C- Style wird das printf Statement verwendet.
- > und >>: Mit dem einfachen und dem doppelten Pfeil nach rechts werden Ausgeben in ein File dargestellt. Mit einem Pfeil wird das angegeben File ersetzt falls vorhanden. Werden 2 Pfeile geschrieben so wird die Ausgaben in die letzte Zeile der vorhandenen Datei angehängt.
- $RANDOM: Mit dem Random Befehl wird ein zufälliger Integer Wert erzeugt im Bereich von 0 – 32767.

Das erstellte Bash-Script ist in der Lage alle Daten die nicht benötig werden zu löschen um eine störfreie Testumgebung zu schaffen. Der Sortieralgorithmus wird danach auf die maximale Eingabelänge getestet. Dafür wird die Anzahl an Werten in einer Schleife erhöht bis ein Überlauf produziert wird. Danach wird die maximale Läng bei Zufallszahlen ermittelt. Dies geschieht ebenfalls so lange, bis ein überlauf produziert wird. Das Bash-Script dokumentiert den gesamten Vorgang und gibt einen kleinen Report aus in welchem die Anzahl für Worstcase und Randomcase beschrieben sind. Des Weiteren Wird eine Datei mit Taktzyklen abhängig von der Eingabelänge erstellt.

7 Quicksort/Bubblesort

7.1 Quicksort

Quicksort ist ein schneller rekursiver Algorithmus der nach dem Prinzip: Teile und Herrsche funktioniert. (Wikipedia_Quicksort, 2016) Der Algorithmus teilt eine bestehende Liste in 2 Listen auf. Dafür wird ein Pivotelement verwendet. Alle Elemente die Kleine sind werden in die eine alle andern

Elemente in die andere Liste gegeben. Danach sind in der ersten List nur kleiner und in der 2. Liste nur Größere Elemente zu finden. Danach ruft sich der Quicksortalgorithmus wieder auf und sortiert die beiden entstandenen Listen nach demselben Prinzip. Die Abbruchbedingung der Rekursion ist wenn eine Unterliste nur mehr 1 oder kein Element mehr enthält. Dann gilt der Listenzweig als sortiert und die Rekursion wird abgebrochen und returned. Vorteil dieses Algorithmus ist das er schneller als andere vergleichbare Algorithmen arbeitet. Im durchschnittsfall benötigt er eine Laufzeit von O (n log(n)) und im worst case Szenario ist er genau so schnell wie Bubblesort mit einer Laufzeit von O(n^2)

7.2 Bubblesort

Bei Bubblesort (Wikipedia_Bubblesort, 2016) wir eine bestehende Liste immer von links nach rechts durchgearbeitet und dabei immer 2 Elemente miteinander verglichen. Ist das rechte Element kleiner als das linke, werden die beiden Vertauscht, wenn dies nicht der Fall ist passiert kein Tausch. Danach wird das vorher rechte Element zum linken und das rechte Element ist das nächste in der Liste. So wird die gesamte Liste durchgegangen. Ist der Vergleich am Ende der Liste angekommen wird wieder von vorne begonnen. An Ende des ersten Durchgangs steht das größte Element ganz am Ende der Liste. Die Laufzeit beträgt im Average und im worst case Fall immer O(n^2)

8 LDX und STX

8.1 Funktion

LDX und STX sollen das Laden und Speicher aus dem Speicher durch Verringerung der ansonsten nötigen Maschinebefehle beschleunigen.

8.2 Implementierung

Die beiden neuen Befehle LDX und STX werden anstelle der Befehle JR und JL in das Verilogmodell der TOY –CPU eingebaut. Da die Funktion und deren Verbesserungen gleichermaßen für Laden und Speichern gilt wird nur das Speicher genau beschrieben. Es gilt jedoch gleichermaßen für das Laden Durch die Implementierung ist es möglich den Speichervorgang in den Hauptspeicher zu beschleunigen. In der normalen Version musste zuerst der Speicherort bestimmt werden an dem das Element abgelegt werden sollte indem dieses durch Addition eines Offsets zum Beginn des Arrays addiert werden musst. Danach konnte das Element an den Berechneten Platz gespeichert werden. Die neue Funktion STX macht aus diesen beiden Schritten einen indem es die Berechnung des Offsets und den Speichervorgang kombiniert. Nach einem Speichervorgang muss lediglich der Offset neu berechnet werden. Dies musste jedoch bei der normalen Version auch gemacht werden und bringt daher keine Verbesserung oder einen negativen Effekt.

8.3 Optimierung des Quicksortcodes

Durch die Verwendung der Funktionen LDX und STX kann der Code modifiziert werden. Aus zu vor 2 Anweisungen wird danach Eine. Dies spart Platz im Speicher da weniger Maschinencode gespeichert werden muss. Diese Speicher kann wiederum für den Aufbau des Stacks benutz werden. Des Weiteren wird die Anzahl an Taktzyklen reduziert.

9 Ergebnisse

9.1 Worstcase Scenario STOY

Das worst case Szenario für den Quicksortalgorithmus ist, wenn die zu sortierenden Werte bereits sortiert in der Liste vorliegen oder genau einer verkehrt herum sortierten Liste vorliegt. Dabei beträgt die Laufzeit O(n^2). Das worst case Szenario erklärt sich dadurch, dass immer das erste Element in der Liste als Pivotelement gewählt wird. Da dies jedoch das kleinste Element der Liste ist wird die

gesamte restliche Liste nicht weiter aufgeteilt und im nächsten Aufruf des Quicksortalgorithmus wird wieder das kleinste Element als Pivotelement gewählt. Daher wird die Liste nie in 2 Unterlisten aufgeteilt und jedes Element wird einzeln sortiert.
Bei der Testung wird zuerst begonnen mit einem Wert. Nach jeder Runde die der Algorithmus bestanden hat wird die Anzahl der Inputparameter um eins erhöht. Der von uns erstellte STOY – Code ist in der Lage mit dem beschriebenen Testverfahren **13 Werte** die in einer geordneten Liste sind zu sortieren.

9.2 Random Scenario STOY

Bei diesem Test wird der Input nach jeder erfolgreich absolvierten Runde neu generiert. Dafür wurde die Anzahl an Werten die zu sortieren waren variiert. Begonnen mit 2 Werte wurde nach jedem durchlauf die Anzahl an Werten um 1 erhöht. Dies wurde so lange durchgeführt, bis bei Quicksort ein überlauf Produziert wurde. Dies geschieht, da der erzeugte Stack nach jedem rekursiven Aufruf der Sortierfunktion wächst, da der Stackpointer, Basepointer und Variablen auf dem Stack gelegt werden. Da Stack, Maschinecode und Array im gleichen Speicher sind, kann der Stack in das eingelesene Array wachsen und so Informationen zerstören. Daher wurde nach jedem Sortierdurchlauf geprüft, ob ein Überlauf passierte. War dies nicht der Fall wurde die Anzahl der Werte am Standardinput erhöht und erneut getestet. Wenn ein überlauf entdeck wurde, wurde der Versuch beendet und die max. Anzahl bestimmt.

Die Werte werden durch die RANDOM Funktion der Bash erzeugt. Es werden Zahlen von 0 bis 32767 generiert. (0x0000 bis 0x7FFF). Um einen statistische Verteilung zu erhalten wurde 40 Testläufe gestartet. Dabei war es möglich folgende Anzahl an Werten zu sortieren:

- Minimale Anzahl: 17 Werte
- Maximale Anzahl: 29 Werte
- Durchschnittliche Anzahl: 21 Werte ± 3 Werte

Die Schwankung der Werte lässt sich durch die zufällige Verteilung der Werte erklären. Da die Anzahl an Rekursionen bei Quicksort abhängig ist von der Art wie die Werte zu Beginn vorliegen. Wie bereits erwähnt ist der Worst Case eine Sortierte Liste. Die minimale Anzahl an Werten die sortiert werden kann befindet sich im Bereich dieser Sortierung.
Bei der maximalen Anzahl an Werten ist die Liste so gestaltet, das als Pivotelement immer das Medianelement verwendet wird. Dieses Teilt die Liste in 2 ca. gleich große Unterlisten und daher werden die wenigsten Rekursionen aufgerufen und der Stack benötig weniger Speicher.

9.3 Laufzeitvergleich Bubblesort, Quicksort in STOY und Quicksort in ATOY

Die Laufzeiten der beiden Programme wurde durch die Messung der Laufzeiten (gemessen in Taktzyklen) bestimmt. Es wurde beiden Programmen eine ansteigende Anzahl an zufälligen Werten übergeben und danach die von den beiden Simulatoren angegebenen Taktzykeln notiert und in einem Diagramm dargestellt.

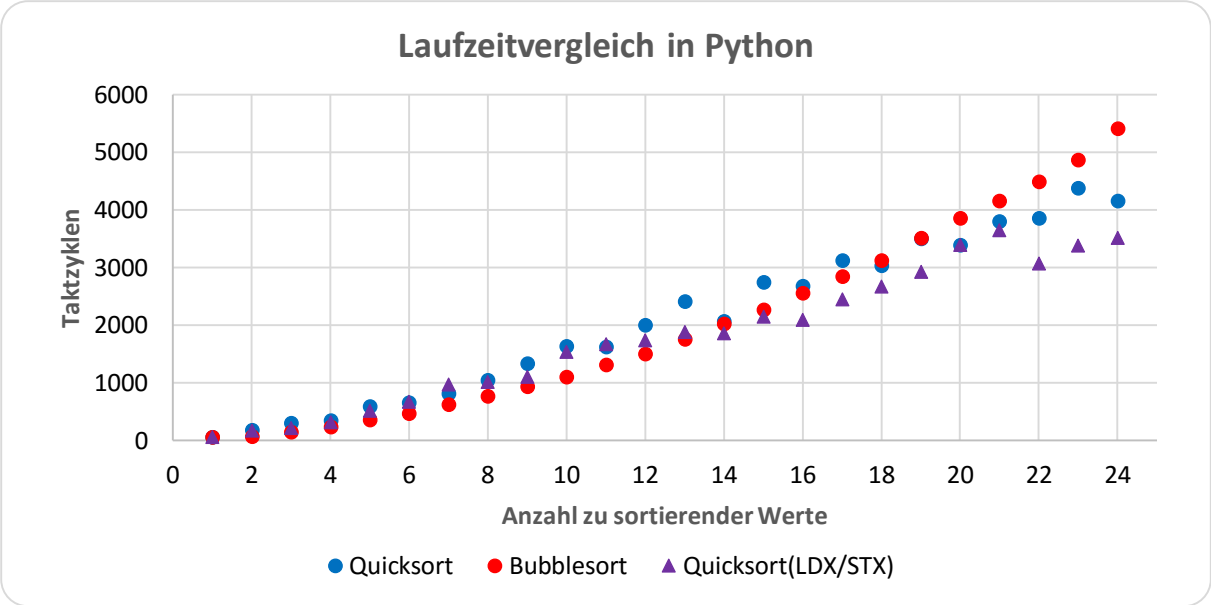

Figure 1: Diagramm mit dem Vergleich der Laufzeiten in Python

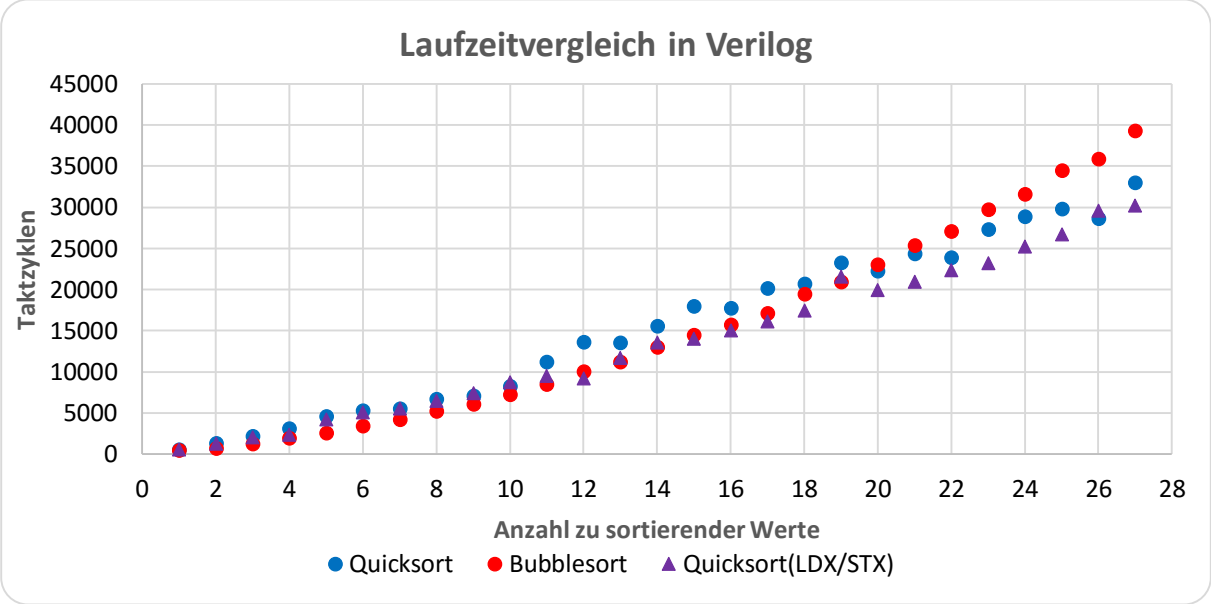

Figure 2: Diagramm mit dem Vergleich der Laufzeiten in Verilog

Die Taktzyklen werden in Verilog und Python verschieden gemessen. Verilog gibt die Dauer in Clockcycles an während das Pythonmodell die Anzahl an Instruktionen angibt. Daher ist der Wert für die Taktzyklen unterschiedlich. Jedoch sind in beiden Diagrammen ein eindeutiger und gleicher Trend zu sehen. Die Laufzeit von Bubblesort steigt gleichmäßig mit einer exponentiellen Funktion an. Bei Quicksort ist zu erkennen, dass das der Verlauf der Kurve flacher ist. Alle 2 Werte ist ein Sprung zu erkennen. Dieser beruht darauf, dass jeweils nach 2 Werten wieder um eine Rekursion mehr gemacht werden muss und diese benötigt wieder mehr Zeit. Dafür ist die darauffolgende Anzahl an Werte ca. gleichschnell sortiert.

9.4 Optimierter Quicksortcode mit LDX und STX

Durch die Optimierung mit LDX und STX konnten einigen Verbesserungen erzielt werden:

- Im Worstcase Szenario könne bis zu 15 Werte sortiert werde.

- Die Anzahl der Taktzyklen ist im Worstcase Szenario um 12% geringer.
- Im Random Szenario ist es möglich zwischen 20 und 30 Werte zu sortieren.
- Im Durchschnitte können bis zu 25 ± 3 Werte sortiert werden.
- Die Anzahl der Taktzyklen ist im Random Szenario um 12% geringer.

10 Literaturverzeichnis

Graz, T. (kein Datum). *Tu Graz.* Von http://portal.tugraz.at/ abgerufen

Posch, K. C. (20. 05 2016). Rechnerorganisation. Graz, Steiermark, Österreich. Von https://seafile.iaik.tugraz.at/f/dbfbd6a5e1/ abgerufen

Wikipedia_Bash. (29. 05 2016). *Wikipeia*. Abgerufen am 07. 06 2016 von https://de.wikipedia.org/wiki/Bash_(Shell)

Wikipedia_Bubblesort. (19. 03 2016). *Wikipedia*. Abgerufen am 08. 05 2016 von https://de.wikipedia.org/wiki/Bubblesort

Wikipedia_Quicksort. (26. 05 2016). *Wikipedia*. Abgerufen am 07. 06 2016 von https://de.wikipedia.org/wiki/Quicksort

11 Abbildungsverzeichnis