AUFGABE 3: AUSFÜHRUNGSZEIT

In dieser Übungsaufgaben werden Sie sich mit verschiedenen Möglichkeiten befassen, maximale Ausführungszeiten abzuschätzen. Ziel dieser Aufgabe ist es, ein Gefühl für die Grenzen dieser Herangehensweisen zu bekommen.

1 Aufgabenstellung

Nach dem erfolgreichen Aufsetzen des `build`*-Verzeichnisses können Sie sich mittels* `make doc` *eine Übersicht über alle in* `libEZS` *bereitgestellten Funktionen inklusive deren Dokumentation erzeugen.*

Denken Sie daran, Ihren Quellcode nach vollständiger Bearbeitung noch vor dem Beginn der Rechnerübung abzugeben. Rufen Sie hierzu in Ihrem `build`*-Verzeichnis* `make submit` *auf.*

1.1 Zeitmessung mit der libEZS:

Wir haben für Sie bereits die Funktionen `checksum()` und `bubblesort()` implementiert.

Aufgabe 1

Von welchen Faktoren ist die Ausführungszeit dieser Funktionen abhängig? Unterscheiden sich diese Faktoren?

Antwort:

Aufgabe 2

Für welche Eingaben weist `bubblesort()` die größtmögliche Laufzeit auf?

Antwort:

Verwenden Sie die von Ihnen implementierte Stoppuhr um die *Worst Case Execution Time (WCET)* beider Funktionen abzuschätzen. Implementieren Sie eine

automatische Messung mit einem Stichprobenumfang von mindestens $N = 100$. Versuchen Sie dabei Störungen der Ausführungsumgebung so gut wie möglich zu eliminieren bzw. zu berücksichtigen [1].

Visualisieren Sie sich hierbei auch das Histogramm der gemessenen Zeitwerte (für zufällige Eingaben fester Länge). Die Zeitwerte können Sie hierzu auf die serielle Konsole ausgeben. Wenn Sie sich diese beispielsweise mit Hilfe von `cutecom` ausgeben lassen, so können Sie die Messwerte direkt in eine Datei speichern.

☞ `ezs_print_measurement()`

Diese Daten können Sie dann z. B. mit `labplot2` visualisieren. Importieren Sie hierzu die Datei mit den Messdaten („New“; „File“, „Import“, „Import from File“, „ASCII data“ mit „Data container: New Workbook“). Ihr Messwerte sollten nun als Spalte importiert sein. Stellen Sie die Daten anschließend als Histogram dar (rechte Maustaste auf den Kopf der Spalte mit den Werten, Menüpunkt „Plot Data“, „Histogram“).

Aufgabe 3

Welche Anforderungen sind an die Stichprobe zu stellen? Wieso fordern wir in dieser Aufgabe einen Stichprobenumfang von mindestens 100?

Antwort:

Aufgabe 4

Wie hoch ist die Auflösung Ihrer Messung? In welcher Größenordnung liegt Ihre Messgröße? Ist Ihre Messung damit gültig? Falls nicht, wie können Sie die Messung zumindest statistisch durchführen?

Antwort:

Aufgabe 5

Machen Sie sich mit den Begriffen Mittelwert, Standardabweichung, Standardfehler, Minimum und Maximum vertraut und berechnen Sie diese Werte für Ihre Messung.

[1] `http://ecos.sourceware.org/ecos/docs-2.0/ref/kernel-interrupts.html`

Welche Aussagen machen diese Kenngrößen über die Datenpunkte Ihrer Messung? Welche Aussagekraft haben diese Werte in Bezug auf die WCET? *Hinweis:* `labplot2` kann Sie bei der Berechnung dieser Werte unterstützen (rechte Maustaste auf den Kopf der Spalte mit den Werten, „Column Statistics“).

Antwort:

1.2 Zeitmessung mit dem Oszilloskop:

In dieser Teilaufgabe lernen Sie Ausführungszeiten mit Hilfe des Oszilloskops zu ermitteln. Die Funktion `ezs_gpio_set()` erlaubt es Ihnen den auf der Experimentierplatine herausgeführten Pin PD12 als GPIO zu verwenden.

Aufgabe 6

Wie können Sie mit Hilfe des GPIO-Pins Ausführungszeiten messen?

Antwort:

Aufgabe 7

Führen Sie nun die Messungen der vorangegangenen Teilaufgabe noch einmal mit Hilfe von GPIO-Pin und Oszilloskop durch.

Aufgabe 8

Wie genau stimmen diese mit den Messungen Ihrer Stoppuhr überein?

Antwort:

Aufgabe 9

Wie viele Stichproben schaffen Sie pro Minute mittels Oszilloskop?

Antwort:

Aufgabe 10

Welche der beiden Messarten würden Sie bevorzugen? Warum?

Antwort:

Aufgabe 11

Welche Probleme sehen Sie sowohl bei der WCET-Abschätzung durch den Zeitgeber als auch bei der durch das Oszilloskop?

Antwort:

1.3 Werkzeuggestützte WCET-Ermittlung:

In dieser Teilaufgabe sollen Sie die WCET mit Hilfe eines Werkzeugs zur Codeanalyse bestimmen. Im Rahmen dieser Veranstaltung kommt hierbei der *aiT*, das in der Tafelübung vorgestellte Werkzeug, der Firma *AbsInt* zum Einsatz. Laden Sie Ihr Programm, wie in den Übungsfolien dargestellt, in den aiT und analysieren Sie die WCET der Funktionen `bubblesort()` und `bubblesort_job()`. Falls aiT Sie nach einer Lizenzdatei fragt, melden Sie sich bitte mit denen in der Übung ausgegebenen Gruppenanmeldedaten bei unserem Lizenzserver an („Connect to license server...“, Host: `i4alm.cs.fau.de`, Port: 42426). Pro Gruppe kann jeweils nur ein Client zeitgleich gestartet werden.

☞ make aiT

Aufgabe 12

Worin liegt aus Sicht von aiT der Unterschied zwischen `bubblesort()` und `bubblesort_job()`? Welche ist besser geeignet um eine realistische WCET-Analyse für die vorliegende Anwendung zu erhalten? Wieso?

Antwort:

Aufgabe 13

Während der Analyse treten Warnungen auf. Weshalb dürfen Sie diese nicht ignorieren?

Antwort:

Stellen Sie aiT mithilfe der in der Übung vorgestellten Annotationen so ein, dass die Analyse ohne Warnung gelingt.

Aufgabe 14

Wie unterscheiden sich die Ergebnisse zwischen dem ersten Durchlauf und dem ohne Warnung?

Antwort:

Aufgabe 15

Aus der Vorlesung sind Ihnen ereignis- und zeitgesteuerte Echtzeitsysteme in verschiedenen Ausprägungen für die Verbindlichkeiten von Termine bekannt. Für welche Arten ist die werkzeuggestützte WCET-Bestimmung gut geeignet? Für welche weniger gut?

Antwort:

1.4 WCET-Analyse-freundliche Entwurfsmuster:

Aufgabe 16

Betrachten Sie nun die Funktion `sample_job()`. Wieso ist das in dieser Funktion verwendete Entwurfsmuster für eine WCET-Analyse schwierig? Wo könnten sich infolgedesen im Systementwurf Probleme ergeben?

Antwort:

Aufgabe 17

Wieso ist die werkzeuggestützte WCET-Bestimmung eines eCos Faden schwierig? Haben Sie eine Idee, wie ein günstigerer Ansatz aussehen könnte?

Antwort:

2 Erweiterte Aufgabe

Die Erweiterten Übungsaufgaben sind nur für Teilnehmer verpflichtend, die das 7,5-ECTS-Modul belegen. Wir werden Sie natürlich auch dann bei der Bearbeitung unterstützen, wenn Sie diese Teilaufgaben freiwillig bearbeiten.

2.1 Analyse von `bubblesort()`

Aufgabe 18

Betrachten Sie erneut die Funktion `bubblesort()`, diesmal mit Hinblick auf Compileroptimierungen. Bisher wurde ihre Anwendung ohne Optimierungen (`-O0`) kompiliert. Passen Sie die `COMPILE_FLAGS` in der `CMakelists.txt` an, um mit `-O2` zu kompilieren.

Führen Sie Messungen mit verschiedenen Eingabewerten (Welche Eingabewerte sind für `bubblesort()` interessant?) sowohl mit `-O0` als auch `-O2` durch. Versuchen Sie beobachtete Unterschiede durch einen Vergleich des tatsächlich generierten Objektcodes von `bubblesort()` sowie der dazugehörigen Kontrollflussgraphen zu

☞ arm-none-eabi-objdump -d

erklären. Diese werden für Sie automatisch während des Bauprozesses erzeugt. Eine genaue Beschreibung des Instruktionssatzes des Cortex-M4-Prozessors finden Sie im entsprechenden Handbuch von ARM in Kapitel 3[2].

☞ bubblesort-arm{-pseudo}-O{0,2}.png

Hinweis: Gehen Sie bei der Betrachtung des Objektcodes zielorientiert vor: Versuchen Sie zunächst, Kontrollflusselemente[3] des C-Codes im Kontrollflussgraphen, und somit im Objektcode, zu identifizieren. Ordnen Sie nun im Quellprogramm eingabedaten-abhängige Konstrukte dem Kompilat zu und betrachten Sie Instruktionen in deren Umgebung.

Welche Unterschiede in der Laufzeit können Sie zwischen den Optimierungsleveln feststellen und wie lassen sich diese anhand des vom Compiler generierten Codes erklären?

Antwort:

2.2 Analyse von heapsort

Aufgabe 19

Betrachten Sie nun die Funktion `heapsort_job()`. Bestimmen Sie Obergrenzen für die maximale Anzahl an Iterationen pro Schleife. Hinweis: Abhängig von den gewählten Parametern kann die Analyse mittels aiT bis zu 20 Minuten dauern. Es ist deshalb in dieser Aufgabe nicht erforderlich, eine WCET-Schranke zu ermitteln.

2.3 Wort-Case Laufzeit von heapsort

Aufgabe 20

Versuchen Sie eine möglichst lange Laufzeit der Funktion `heapsort_job()` für ein Eingabearray der Länge 500 zu erzwingen (mit -O0). Wie können Sie eine Eingabe mit möglichst hoher Laufzeit (jenseits von reinem Bruteforce) konstruieren? Versuchen Sie dabei mindestens die Laufzeitschranke von 164150 Zyklen ($\approx$ *3.908 μs*) überschreiten. Die Gruppe, welche die längste Laufzeit demonstrieren kann, erhält eine Belohnung. Wenn Sie am Wettbewerb teilnehmen wollen, dann senden Sie bitte das zugehörige Eingabearray und Ihren Messwert an *i4ezs-owner@lists.informatik.uni-erlangen.de*.

[2] `http://infocenter.arm.com/help/topic/com.arm.doc.dui0553b/DUI0553.pdf`
[3] Schleifen, Verzweigungen, Aufrufe, Arrayzuweisungen

Hinweis: Wie Sie aus der Tafelübung wissen, hängt die Laufzeit eines Algorithmus sowohl von Eingabedaten als auch Betriebssystemkontext und Hardware ab. Während Sie den Einfluss des Betriebssystems durch geeigneten Messaufbau in dieser Aufgabe eliminieren, so sind u.A. auch Hardwareeffekte (vor allem Speicherplatzierung) für diese Aufgabe durchaus relevant. Testen Sie deshalb neben Eingabedaten unbedingt auch verschiedene Speicherstellen des Arrays, da durch Linkereffekte Eingabedaten (insb. global sichtbar vs. modullokal vs. stapellokal) in unterschiedlichen Speicherbänken abgelegt sein könnten. Können Sie hier Effekte feststellen? Bitte kontaktieren Sie uns, sollten Sie nachhaltige Probleme dabei feststellen die oben geforderte Zeitschranke zu erfüllen!

Hinweise

- Bearbeitung: Gruppe mit je drei Teilnehmern.
- Abgabefrist: 03.06.2024 (23:59) ☞ `make submit`
- Fragen bitte an `i4ezs@lists.cs.fau.de`