lea.schoenberger [☺] tu-dortmund.de
nils.hoelscher [☺] tu-dortmund.de
nick.pietrass [☺] tu-dortmund.de
jan.pomplun [☺] tu-dortmund.de

Übung zur Vorlesung
Eingebettete Systeme
Wintersemester 18/19

Aufgabenblatt 10 (Praxis)

(20 Punkte)

Hinweis: Für dieses Aufgabenblatt ist keine Abgabe erforderlich. Besprechung: 09.-11.01.2019.

Da unsere Lizenz für eines der verwendeten Tools auf bestimmte Rechner beschränkt ist, müssen Sie sich aus dieser VM heraus per SSH zum Zielrechner verbinden. Verwenden Sie dazu den Befehl **ssh ACCOUNTNAME@ls12pc5.cs.tu-dortmund.de -X** in einem Terminalfenster mit Ihrem jeweiligen Accountnamen und geben Sie das zugehörige Passwort ein.

Vorbereitung

Wenn Sie eine neue Session starten, müssen Sie einige Komponenten neu registrieren. Geben Sie dazu die folgenden Befehle in ein Terminal ein:

- **cd wcet**
- **./env.sh**
- **m+ tricore-gcc**
- **m+ ait**

In Ihrem Homeverzeichnis auf dem Zielserver sind bereits die Dateien entpackt, die für die Übung notwendig sind. Sie befinden sich im Verzeichnis `wcet` – bitte beachten Sie, dass Sie wegen der Verbindung auf einen anderen Rechner nicht einfach den Dateibrowser der VM verwenden können, sondern auf die Kommandozeile angewiesen sind!

1 Schritt 1: Einfache Analyse (5 Punkte)

Verwenden Sie den Befehl `cd ~/wcet/step1`, um in das Verzeichnis für diese Teilaufgabe zu wechseln.

Kompilieren Sie das Beispielprogramm mit dem Befehl **tricore-gcc -g -T ../tc1796.lds test.c** und schauen Sie sich zudem den Quellcode an (z.B. via `kate test.c`, `gedit test.c`, `gvim test.c` oder `cat test.c`).

Was macht das Programm?

Verwenden Sie nun den Befehl `a3tricore`, um den Tricore-Analyzer zu starten. Beim ersten Starten muss eine Lizenzdatei geöffnet werden, diese befindet sich unter:
`/app/unido-i12/i486-suse-linux/aiT/a3/license_v10_2019.dat`
Wählen Sie unter Configuration→Files die Datei `a.out` als Executable aus und `a.ais` als "AIS file". Klicken Sie danach auf das Stiftsymbol in der "AIS file"-Zeile, um den AIS-Editor zu starten. **Wie werden die Loop Bounds mit dieser Datei annotiert?**

Wählen Sie den Punkt "Analyses→Create" aus der Navigationsbox links aus und klicken Sie auf die Option "aiT" (Safe WCET Analysis). In dem daraufhin geöffneten Bildschirm für die Analyseparameter muss nun noch `main` als Symbol für den Analyse-Start ausgewählt werden, bevor Sie die Analyse mit Klick auf den "Play"-Knopf (der rechte nur mit Dreieck und ohne Punkt) in der Symbolleiste starten können.

Was ist die WCET des Programms? Sie können zusätzliche Details aufrufen, indem Sie auf den Punkt "WCET contributions" in der Navigationsbox links klicken.

Schauen Sie sich zudem den Kontrollflussgraphen an (Analysis→Control-Flow Graph).

Wie wird das Programm im Kontrollflussgraphen repräsentiert? Beachten Sie insbesondere, wie Schleifen im CFG dargestellt werden.

Starten Sie die Analyse erneut, aber verwenden Sie dieses Mal die interaktive Analyse (Play-Symbol mit Punkt an der Spitze). Schauen Sie sich an, welche zusätzlichen Informationen in den Boxen dargestellt werden, und öffnen Sie einige der Boxen durch Doppelklick. **Was wird dort dargestellt?**

2 Schritt 2: Scratchpad-Allokation (5 Punkte)

Ein Scratchpad (SPM, ScratchPad Memory) ist ein besonders schneller, interner Speicher, der für die temporäre Zwischenspeicherung von Rechenergebnissen, Daten oder Programmteilen verwendet wird. Im Gegensatz zu einem Cache wird ein Scratchpad-Speicher vom Programmierer verwaltet und nicht automatisch von der Hardware.

Welche Teile eines Programms sollten in ein SPM ausgelagert werden?

Was unterscheidet ein "normales" System von einem Echtzeitsystem, wenn es um den Einsatz eines SPMs geht?

Wechseln Sie in das Verzeichnis `step2` (`cd ~/wcet/step2`). Hier befindet sich eine weitere Kopie des Programms aus Aufgabe 1, in dem zwei Funktionen in das SPM verschoben wurden. Kompilieren Sie es analog zum `step1`-Programm und lassen Sie es von aiT analysieren.

Wie hat die SPM-Nutzung die Laufzeit des Programms beeinflusst?

Wenn Sie eine weitere Funktion ins SPM legen könnten, welche würden Sie wählen? Warum?

3 Schritt 3: Scratchpad-Allokation und Function Outlining (5 Punkte)

Verwenden Sie den Befehl `cd ~/wcet/step3`, um in das Verzeichnis für diese Teilaufgabe zu wechseln. In der Datei `test.c` wurden einige Vorbereitungen getroffen, um die inneren Schleifen der Funktionen `Initialize` und `Sum` per Function Outlining in getrennte Funktionen abzuspalten, damit selektiv nur diese inneren Funktionen in das Scratchpad Memory (SPM) verschoben werden können.

Nehmen Sie das Function Outlining für die inneren Schleifen dieser beiden Funktionen vor.

Kompilieren Sie danach das Beispielprogramm mit dem Befehl **tricore-gcc -o test.elf -g -T ../tc1796.lds test.c** und laden Sie es wie beim vorherigen Übungsblatt in den `a3tricore`-Analyzer. Wenn Sie nun eine aiT-Analyse starten, werden bei der Analyse einige Fehler gemeldet.

Korrigieren Sie die Fehler in der Datei `a.ais`, damit die Analyse erfolgreich durchgeführt werden kann.

Wie hat sich die WCET im Vergleich zur vorherigen Fassung (komplette Funktionen im SPM) verändert?

Hinweis: Dort hatte der Analyzer eine WCET von 85117 Zyklen berechnet.

4 Schritt 4: Ganzzahlige Lineare Programmierung (5 Punkte)

aiT analysiert die WCET eines Programmes, indem es jeden Basisblock einzeln analysiert, mit den Ergebnissen ein ILP (ganzzahliges lineares Programm) aufstellt und dieses löst.

Im Verzeichnis `step4` befindet sich eine Beispielformulierung eines ILPs in der Datei `example.lp`. Schauen Sie sich den Inhalt dieser Datei mit einem beliebigen Texteditor an und lassen Sie es mittels `lp_solve example.lp` von einem ILP-Solver lösen.

Als weiteres Beispiel befindet sich im Verzeichnis die Datei `example2.lp`, die den Kontrollflussgraphen eines sehr einfachen Programms modelliert. Auch diese Datei können Sie mit `lp_solve example2.lp` lösen lassen.

Erläutern Sie das Ergebnis.

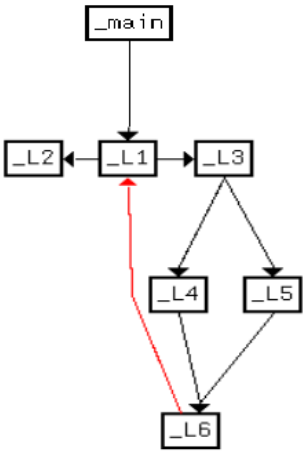
Wie kann man im ILP modellieren, dass ein spezifischer Basisblock eine Laufzeit von mehr als einem Zyklus hat?

Gegeben sei nun folgendes kleines Programm mit Schleife:

```
int main()
{
  int i, j = 0;

  _Pragma( "loopbound min
                100 max 100" );
  for ( i = 0; i < 100; i++ ) {
    if ( i < 50 )
      j += i;
    else
      j += ( i * 13 ) % 42;
  }

  return j;
}
```



Block	Zyklen
main	21
L1	27
L2	20
L3	2
L4	2
L5	20
L6	13

Abbildung 1: Ein Beispielprogramm mit Schleife.

Die grundsätzliche Struktur dieses Programms ist bereits in der Datei `ipet.lp` modelliert.

Ergänzen Sie in der Datei die Anzahl der Zyklen für jeden Basisblock.

Wenn Sie nun versuchen, das ILP per `lp_solve ipet.lp` lösen zu lassen, wird der Solver mit der Fehlermeldung *This problem is unbounded* abbrechen.

Warum kann der ILP-Solver keine Lösung finden?

Ergänzen Sie den fehlenden Constraint für Kante h.

Lassen Sie die korrigierte Datei nochmals von lp_solve lösen – Sie sollten nun ein Ergebnis erhalten.
Warum berechnet der Solver eine Lösung, in der L4 nie ausgeführt wird?