lea.schoenberger [☺] tu-dortmund.de
christian.erdmann [☺] tu-dortmund.de
nils.hoelscher [☺] tu-dortmund.de
jan.pham [☺] tu-dortmund.de

Übung zur Vorlesung
Eingebettete Systeme
Wintersemester 19/20

Aufgabenblatt 10 (Praxis)

(10 Punkte)

Hinweis: Für dieses Aufgabenblatt ist keine Abgabe erforderlich. Bearbeitung: 06.-10.01.2020.

Öffnen Sie ein Terminal und geben Sie den Befehl **ssh -X youraccount@ls12pc5.cs.tu-dortmund.de** ein, wobei Sie statt *youraccount* Ihren Benutzernamen einsetzen. Laden Sie mit **wget** das Archiv `wcet2` von der Veranstaltungswebseite herunter und entpacken Sie es. Falls Sie im Umgang mit der Kommandozeile nicht versiert sind, hilft Ihnen Google an dieser Stelle.

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**

1 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.

2 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.

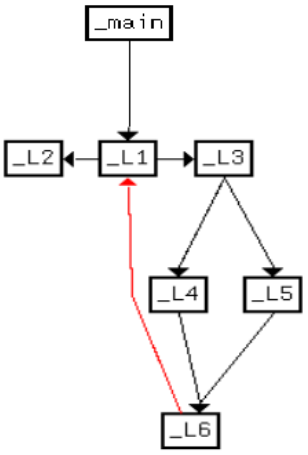
Wie kann man im ILP modellieren, dass ein spezifischer Basisblock eine Laufzeit von mehr als einem Zyklus hat?

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?