Aufgabenblatt zum Praktikum Optimierende Compiler

Prof. Dr. Andreas Koch
Jens Huthmann, Florian Stock

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 12
Aufgabenblatt 4 — Abgabedatum: 01.07.2012 23:59 CEST

Einleitung

Dieses Aufgabenblatt behandelt zwei weitere Datenflussanalysen (LIVE und REACH). LIVE benötigen Sie für das nächste Aufgabenblatt. Mit Hilfe von REACH sollen Sie ein weiteres Optimierungsverfahren implementieren.

Aufgabe 4.1 Interaktive Oberfläche

Die von Ihnen in Aufgabe 1 erstellte interaktive Oberfläche des Bantam-Compilers soll um vier Kommandos erweitert werden.

`live` soll die Live-Variables Analyse, welche Sie in Aufgabe 4.2 implementieren, für alle CFGs des Quellprogramms durchführen.

`reach` soll die Reaching Definitions Analyse, welche Sie in Aufgabe 4.3 implementieren, für alle CFGs des Quellprogramms durchführen.

`dumpdataflow` soll die Ergebnisse aller Datenflussanalysen in textueller Form ausgeben. Genaueres ist in Aufgabe 4.4 beschrieben.

`licm` soll den Loop-Invariante-Code Motion Algorithmus welchen Sie in Aufgabe 4.5 implementieren auf alle CFGs des Quellprogramms anwenden.

Aufgabe 4.2 Live-Variables Analyse

Implementieren Sie das in der Vorlesung vorgestellte Verfahren zur Live-Variables Analyse unter Verwendung Ihres Datenflussframeworks. Das Ergebnis Ihrer Analyse werden Sie in Ihrer Rückwandlung aus der SSA Form mit dem Briggs Algorithmus weiterverwenden.

Aufgabe 4.3 Reaching-Definitions Analyse

Implementieren Sie das in der Vorlesung vorgestellte Verfahren zur Reaching-Definitions Analyse unter Verwendung Ihres Datenflussframeworks.

Aufgabe 4.4 Dump Dataflow

Implementieren Sie eine textuelle Ausgabe der Ergebnisse der Datenflussanalyse(n) mit dem in Listing 1 beschrieben Format. Durchlaufen Sie alle Basisblöcke beginnend mit dem Startblock des aktuellen CFGs in Pre-Order. Schreiben Sie das Ergebniss für jede Methode jeder eigene Datei welche Sie mit `<Methodenname>.dfdump` bennenen. Für `Analysename` verwenden Sie die in Tabelle 1 gegebenen Bezeichner.

Listing 1: Ausgabeformat für Dump Dataflow

```
<Methodenname>
<TAB>BB<Basisblocknummer>
<TAB><TAB><Analysename>
<TAB><TAB><TAB>ENTRY:<EntrySet>
<TAB><TAB><TAB>EXIT:<ExitSet>
```

Tabelle 1: Analysenamen

Analysename	Analyse
LIVE	Live-Variables
REACH	Reaching-Definitions

Aufgabe 4.5 Loop-Invariant-Code Motion

(Teil)Ausdrücke einer Schleife, die innerhalb der Schleife nur Variablen verwenden, die außerhalb der Schleife definiert worden sind, können aus der Schleife herausbewegt werden ohne die Korrektheit des Programms zu beeinflussen. Um diese Ausdrücke zu erkennen verwenden Sie das Ergebnis ihrer Reaching-Definitions Analyse.

Ihre Aufgabe ist es Loop-Invariant-Code Motion (LICM) zu implementieren. Sie sollen sich bei den bearbeiteten Ausdrücken auf primitive Integer Operationen beschränken.

Listing 2: Beispiel vor LICM

```
int foo(int x) {
  sum = 0;
  for(i=0;i<n;i++) {
    for(j=0;j<n;j++) {
      sum = sum + x + i;
    }
  }
  return sum;
}
```

Listing 3: Beispiel nach LICM

```
int foo(int x) {
  sum = 0;
  for(i=0;i<n;i++) {
    tmp1 = x + i;
    for(j=0;j<n;j++) {
      sum = sum + tmp1;
    }
  }
  return sum;
}
```

Programmierstil

Die von Ihnen erstellten Programme werden in der Endfassung erfahrungsgemäß einige Tausend Zeilen Java umfassen. Um dem Betreuer das Verständnis und Ihnen die Wartung zu erleichtern, sollen Sie von Anfang an einen sauberen und disziplinierten Programmierstil praktizieren.

Bei der Implementierung sind die Konventionen aus *Writing Robust Java Code* weitgehend einzuhalten. Dieses Dokument liegt als PDF auch auf der Web-Seite der Vorlesung. Ergänzend soll folgendes beachtet werden:

- Achten Sie darauf, dass Klassen nicht zu komplex werden (zu viele Instanzvariablen, zu viele Methoden). Bei deutlich mehr als 20 dieser Konstrukte sollten Sie die Klasse aufteilen.
- Analoges gilt für die Komplexität von einzelnen Methoden. Auch hier sollten Sie bei mehr als 100 Programmzeilen Länge die Methode aufteilen.
- Verwenden Sie statt Abfragen von `instanceof` echte objekt-orientierte Konstrukte (z.B. polymorphe Methoden).

Der Test und die Abnahme Ihrer Programme wird vom Betreuer auf Linux mit dem SUN Java Development Kit (JDK) Version 1.7 erfolgen.

Dokumentation

Die Lösungen werden nur durch das unten beschriebene `README` und die in das Java-Programm eingebetteten JavaDoc-Direktiven und Kommentare dokumentiert. Achten Sie daher darauf, dass Sie von diesen beiden Möglichkeiten ausreichend und aussagekräftig Gebrauch machen!

Kommentare sollen am Anfang jeder von Ihnen modifizierten oder neu erstellten Quelldatei, pro Klasse und pro Instanzvariable und Methode verfasst werden. Bei Verwendung relativ kurzer Methoden und aussagekräftiger Bezeichner können sich Kommentare innerhalb von Methoden auf wenige wirklich wichtige Stellen beschränken. Bei komplizierteren Methoden soll der Ablauf aber durch eine größere Anzahl an aussagekräftige Kommentaren im Methodenrumpf verdeutlicht werden.

Der Dateikopfkommentar muss neben einer allgemeinen Beschreibung auch eine Historie von Änderungen enthalten. Jeder Eintrag in dieser Historie beschreibt unter Angabe von Datum/Uhrzeit und Namen des Autors auf 1-2 Textzeilen die Natur der Änderungen. Alternativ kann hier auch das Log Ihres Versionskontrollsystems (z.B. CVS oder besser SVN) verwendet werden.

Diese Angaben sind für den Betreuer wichtig, damit im Kolloquium die für ein Thema passenden Ansprechpartner gefunden werden!

Abgabe

Grundsätzlich schickt jede Gruppe spätestens zum Abgabetermin in einem `.jar`- oder `.zip`-Archiv die Quelldateien Ihrer Version des Bantam-Compilers an

`oc@esa.informatik.tu-darmstadt.de`

mit dem Subject `Abgabe 4 Gruppe` *N*, wobei Ihnen *N* bereits in der Vorlesung mitgeteilt wurde. In dem Archiv sollen nicht nur die eigenen, sondern *alle* (auch unmodifizierten) Quellen des Bantam-Compilers enthalten sein. Ebenso legen Sie eventuell verwendete zusätzliche externe Bibliotheken in Form ihrer jeweiligen `.jar`-Dateien bei (aber siehe Abschnitt Plagiarismus).

Wichtig: Um unseren Testaufwand überschaubar zu halten, lesen Sie bitte die letzten drei Absätze nocheinmal. Sie sollen abgeben:

- Die *vollständigen* Quellen (also die `.java`-Dateien!)
- Aber *nur diese*, nicht noch irgendwelche Altdaten Ihrer eigenen Testläufe (soweit nicht explizit in der Aufgabenstellung angefordert). Räumen Sie also Ihre Verzeichnisse auf, bevor Sie sie in ein Archiv packen!
- Die eventuell benötigten Fremdbibliotheken
- Alles zusammen in einem `.jar` oder `.zip`-Archiv. Also kein `.7z`, `.tar.gz` oder `.tbz`

Daneben enthält das Abgabearchiv eine Datei `README.txt`, die enthält

- die Namen der Gruppenmitglieder und die jeweils bearbeiteten Themen
- eine Übersicht über die neuen und geänderten Dateien mit jeweils einer kurzen (eine Zeile reicht) Beschreibung ihrer Funktion.
- Hinweise zur Compilierung der Quellen. Geben Sie eine `javac`-Kommandozeile an bzw. verweisen Sie auf mitgelieferte Makefiles oder ANT Build-Dateien. *Nicht* ausreichend ist ein Hinweis auf eine von Ihnen verwendete IDE (wie Eclipse, NetBeans etc.).
- Angaben über weitere Bibliotheken (beispielsweise JSAP, log4j, JUnit etc.), die Sie eventuell verwendet haben. Diese Bibliotheken legen Sie bitte dann auch als `.jar` Dateien in das abgegebene Archiv.

Gruppenarbeit

Bei den Tests von Optimierungen ist es häufig so, dass auf den ersten Blick alles funktioniert. Aber dann in einem Quelltext ein “Sonderfall” auftaucht, der vom bisherigen Optimierungscode noch nicht oder nur fehlerhaft bearbeitet wird. Unterschätzen Sie also den Testaufwand nicht!

Falls in Ihrer Gruppe eine Situation entstehen sollte, in der einzelne Mitglieder deutlich zu wenig (oder zu viel!) der anfallenden Arbeitslast bewältigen, sprechen Sie den Betreuer bitte *frühzeitig* auf die Problematik an. Nur so kann durch

geeignete Maßnahmen in Ihrem Interesse gegengesteuert werden. Nach der Abgabe ist es dafür **zu spät** und Sie tragen die Konsequenzen selber (z.B. wenn sich eines Ihrer Team-Mitglieder wegen seiner Verpflichtungen beim Wasser-Polo nur stark eingeschränkt den Mühen der Programmierung widmen konnte, und Sie daher eine unvollständige Lösung abgeben mussten).

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe einer Lösung zu den Programmierprojekten bestätigen Sie, dass Ihre Gruppe die alleinigen Autoren des neuen Materials bzw. der Änderungen des zur Verfügung gestellten Codes sind. Im Rahmen dieser Veranstaltung dürfen Sie den Code des Bantam-Compilers von der FG ESA Web-Seite zu Optimierende Compiler sowie Code-Bibliotheken für nebensächliche Programmfunktionen (Beispiele siehe oben) frei verwenden. Mit anderen Gruppen dürfen Sie sich über grundlegenden Fragen zur Aufgabenstellung austauschen. Detaillierte Lösungsideen dürfen dagegen *nicht vor Abgabe*, Artefakte wie Programm-Code oder Dokumentationsteile *überhaupt nicht* ausgetauscht werden. Bei Unklarheiten zu diesem Thema (z.B. der Verwendung weiterer Software-Tools oder Bibliotheken) sprechen Sie bitte Ihren Betreuer gezielt an.

Weitere Infos unter `www.informatik.tu-darmstadt.de/plagiarism`