Haskell auf der Überholspur mit LLVM

Henning Thielemann

2014-06-20

Motivation

- Haskell-Programmierer:
  „Was nützt schnelle Berechnung, wenn Ergebnis falsch ist?“
- C-Programmierer:
  „Was nützt korrektes Ergebnis,
  wenn man Ende der Berechnung nicht miterlebt.“

Simon Peyton-Jones:

- Haskell: sicher, aber unnütz
- C: nützlich, aber unsicher

Frage: Geht auch sicher und nützlich?

Anwendungen

- Signalverarbeitung
- Bildverarbeitung
- lineare Algebra mit großen Matrizen

Ansätze in Haskell

- `stream-fusion`
- `dph`, `repa`
- `storable-vector`, `vector`

Methoden:

- Auflösen von Funktionsaufrufen („inlining“)
- Erzwingen sofortiger Auswertung („strictness annotations“)
- Vermeide Speicherung von Zwischenergebnissen („fusion“)

Probleme:

- umständlich
- fragil

Ansätze mit weiteren Hilfsmitteln

- externe Bibliotheken: `hmatrix` für LAPACK
- externe Sprachen:
  - `harpy` für x86-Maschinencode
  - `haskelldb` für SQL
  - `csound-expression` für Csound
- externe Server: `hsc3` für SuperCollider
- externe Prozessoren: `accelerate-cuda` für GPU

Probleme:

- Konvertierungen zwischen Haskell und externer Sprache (umständlich)
- gewohnte Funktionen höherer Ordnung (`map`, `foldr`) meist nicht sinnvoll einsetzbar

radikaler Ansatz

- Prozessoren von heute –
  Maschinensprachen im Stile der 70er-Jahre
- passt nicht mehr zu modernen superskalaren Hyperthread-Mehrkern-Architekturen mit Caches, Vektorrecheneinheiten und langen Pipelines

Spezialprozessoren für funktionale Programmierung:

- Lisp machine
- FORTH-Prozessor
- Burroughs large systems für ALGOL 60
- Reduceron, implementiert mit Lava auf FPGA

Probleme:

- Reduceron noch nicht reif für Einsatz
- Itanium-Flop

LLVM-JIT

- effizienter Code zur Laufzeit erstellt
- Kapselung in EDSL
  (eingebettete anwendungsbezogene Sprache)
- läuft auf CPU oder GPU

LLVM

`http://llvm.org/`

- ursprünglich „Low-Level Virtual Machine“
- Komponenten: Optimierer, Assembler, Binder, JIT, Clang
- Erweiterbarkeit: neue Optimierungen, neue Prozessoren
- eigene maschinennahe Zwischensprache
- Extras: Garbage Collection, Ausnahmebehandlung
- nicht enthalten: Syntaxprüfung, Modulsystem, Typprüfung

Sichtweisen:

- LLVM = „Compiler-Backend“
- LLVM = portabler abstrakter Assembler

Just-In-Time compilation

Heute vorgestellt: JIT

- praktisch „Inline“-Assembler
- erzeugt effizienten Code zur Laufzeit
- Optimierungen für verwendeten Prozessor
- Code erzeugen abhängig von Benutzereingaben

Nachteile:

- Laufzeit-Abhängigkeit von LLVM-Bibliothek in passender Version
- Übersetzung benötigt Zeit

Haskell-Schnittstellen

- GHC-LLVM: schreibt ll-Dateien,
  ruft llvm-Kommandozeilenprogramme auf
- `llvm-general`: arbeitet mit Syntaxbaum
  geeignet für Entwicklung von Übersetzern
- `llvm-base`/`llvm`: ruft LLVM-C-Schnittstelle auf
- davon abgeleitet: `llvm-ffi`/`llvm-tf`

Betrachtete Schnittstelle

- `llvm-ffi`: Schnittstelle zu LLVM-C
- `llvm-tf`: typsicherer Überbau mit Typfunktionen
- `llvm-extra`: Zusatzfunktionen für Schleifen, Vektorisierung, prozessorabhängige Instruktionen, Nachbau **`Maybe`**, **`Either`**

Beispiel-Anwendungen:

- `synthesizer-llvm`: Audiosignalverarbeitung
- `knead`: mehrdimensionale Felder – `repa` für LLVM

Komponenten

- Zusammenbau des virtuellen Assemblerprogramms
- Optimierung
- Übersetzung in eine C-Funktion

Zwischenstufen und Übersetzungen

- LLVM-Assembler: `example.ll`
  - `llvm-as example.ll`
- Bitcode: `example.bc`
  - `llvm-dis example.bc`
  - `opt -O3 <example.bc >example-opt.bc`
  - `llc example.bc`
- echter Assembler: `example.s`
  - `as -o example.o example.s`
  - `llc -filetype obj example.bc`
- echter Maschinencode: `example.o`

Hierarchie

Modul gespeichert in

- `example.ll` oder
- `example.bc` oder
- `example.s` oder
- `example.o`

Unterteilung

- Modul: besteht aus Funktionen
- Funktion: besteht aus Blöcken
- Block: Anfang = Sprungmarke, Ende = Sprung

Instruktionen

virtuelle Register

- jedes Register genau einmal beschrieben (initialisiert)
- danach Register unveränderlich
  Single Static Assignment = SSA
- beliebig viele virtuelle Register verwendbar
- LLVM entscheidet, auf welches Maschinenregister ein virtuelles Register gelegt wird, oder ob es im Speicher abgelegt wird
- Phi-Instruktion führt Werte aus verschiedenen Ausführungspfaden zusammen

Anmelden

- Benutzer: `hal9`

Studieren geht vor Probieren

mitgelieferte Beispiele

- `llvm-tf/example/Fibonacci.hs`

Studieren geht vor Probieren

Beispiel: Erzeugung Sägezahn

- `llvm2014-code/src/Tone.hs`
- Erzeugung Sägezahnton studieren
- LLVM-Code studieren
- Assembler-Code studieren
- Optimierer ausprobieren

Übungsaufgaben

- andere Frequenz als 0.01
- Frequenz als Parameter der C-Funktion
- Tonfolge: Aufruf der Funktion mit verschiedenen Frequenzen
- andere Wellenform (Rechteck, Dreieck, Parabel, Sinus)
- exponentiell abfallende Lautstärke
- Stereokanäle mit leicht unterschiedlichen Frequenzen
- mehrere Töne mischen
- Frequenzmodulation
- Vektorisierung

EDSL für Audiosignalverarbeitung

`synthesizer-llvm`