CPU実験
最終発表会

コンパイラ係の担当分抜粋

コンパイラ係

概要

- フロントエンドはmincamlをそのまま利用
  + 最適化のモジュールを追加, 必要に応じて仕様を一部変更

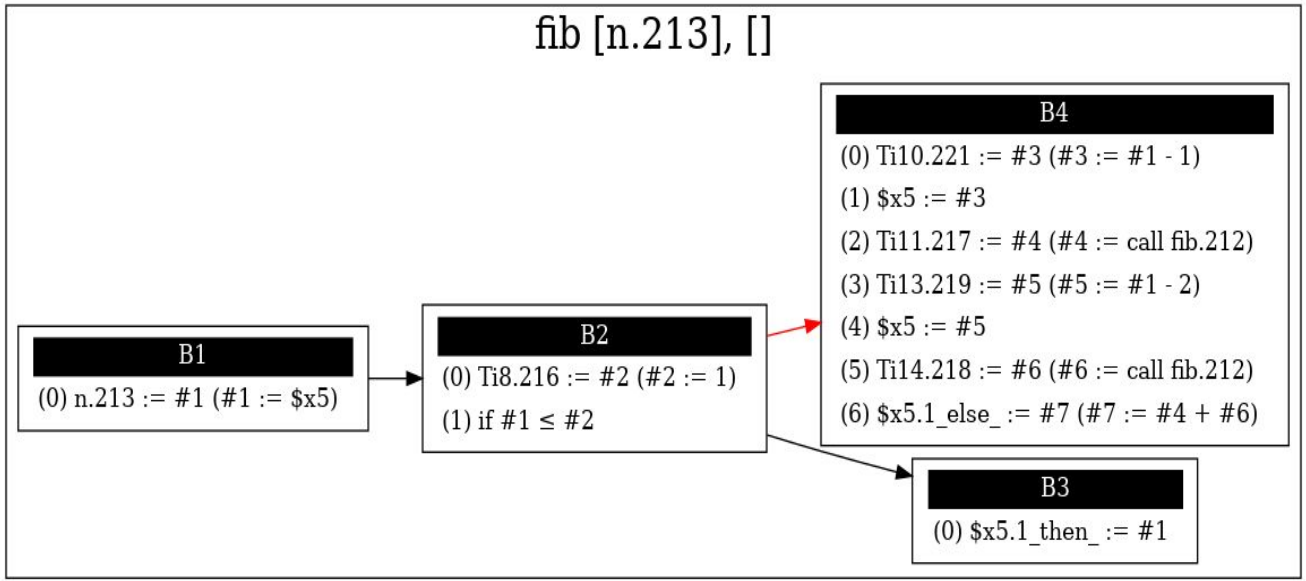
- バックエンドはSSA形式のものをスクラッチ
  - llvm mirを参考にSSAの命令列 + コントロールフローグラフ(CFG)による表現を採用
    - 基本ブロックとの対応をアセンブリにコメント
    - デバッグの際にアセンブリの一部分が, 元のプログラムのどこへ対応しているかわかりやすい (次スライドに出力例あり)
    - 局所的な変更をSSA命令列, 大局的な構造の変更をCFGでそれぞれ独立に行える

出力例 (test/fib.ml)

```
fib.212:
# B1
    sw  x5, 0(x2)
# B2
    li x10, 1
    bgt x5, x10, fib_bleelse_b4
# B3
    ret
fib_bleelse_b4:
# B4
    addi x5, x5, -1
    sw  ra, 1(x2)
    addi    x2, x2, 2
    jal fib.212
    addi    x2, x2, -2
    lw  ra, 1(x2)
    sw  x5, 1(x2)
    lw  x10, 0(x2)
    addi x5, x10, -2
    sw  ra, 2(x2)
    addi    x2, x2, 3
    jal fib.212
    addi    x2, x2, -3
    lw  ra, 2(x2)
    lw  x10, 1(x2)
    add x5, x10, x5
    ret
```

← アセンブリ

デバッグ用のCFG ↓



Aセメスター中の命令数変遷 (256 * 256 レイトレ)

日付	最適化	インライン数	命令数[億]
12/08	グローバル配列導入でクロージャ削除		172
	共通部分式削除	10	98
1/2	SSAレジスタ割当,callee reg導入	10	90
1/6	空ジャンプ省略, 関数ラベルへの条件分岐	10	88
1/12	xor,fabs命令追加, レジスタ数32 → 128	50	78
1/23	再帰関数をインライン展開しない	600	72.5
1/23	while式導入(diffあり)	88	66
2/4	同上	600	51.5

2月中の命令数変遷 (256 * 256 レイトレ)

日付	最適化	インライン数	命令数[億]
2/11	while式のバグを直し, diffを消す	600	53
	実装済みの最適化をwhile式のために修正		52
2/17	分岐の畳み込み		47
2/21	+ 帰納変数の変換		46.5
3/3	+ 配列のCSE - 帰納変数の変換		46.5
	レジスタでオフセット指定するロード , 小数比較分岐の追加命令		41.7

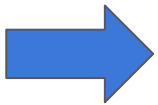
参考記録

- 128 * 128 : 12.9億命令 - 512 * 512 : 145.6億命令

不採用の最適化　帰納変数の変換

- レイトレのプログラムで用いられる末尾再帰関数のなかに配列とオフセットを引数として取るものが多かった.
- lwの際に配列の先頭アドレス + オフセットを計算するのを辞めたかった
  (当初はlwのオフセットとして即値のみが利用可能だったため)

```
la r0, array
mv r1, zero
loop:
bge r1, r3, exit
add r2, r0, r1
lw rd, 0(r2)
addi r1, r1, 1
j loop
exit:
```



```
la r0, array
add r3, r0, r3
mv r1, r0
loop:
bge r1, r3, exit
lw rd, 0(r1)
addi r1, r1, 1
j loop
exit:
```

過去の最適化　ループのブロック出力順変更

- 左の素朴なコンパイルではループ内で returnブロックへの条件分岐と , エントリへの無条件分岐がある
- 右のアセンブリでは fall-throughを利用してループ内は条件分岐 1つだけに
- 最初にジャンプが一つ増えているが , ループが複数回回れば変換後の方がジャンプが少ない

```
label0:
        (entry)
label1:
        (pred)
        bcond x1, x2, label2
        (continue)
        j label0 # 再帰
label2:
        (return)
```

```
        j label0
label3:
        (continue)
label0:
        (entry)
label1:
        (pred)
        ~bcond x1, x2, label3
label2:
        (return)
```

(name) : アセンブリ列
bcond : beq, bleなど条件分岐命令
~bcond : bcondの比較を反転させたもの

最近の最適化

- 分岐の畳み込み
  - インライン展開を進めると, Φ関数に書き込まれる値が定数で,
    次の分岐の方向が静的に定まっている部分コードが出現した
  - 無駄な比較を避けて, 分岐先に直接ジャンプするように
  - Φ関数で書いた値が分岐以降に...

  生存しないとき:
  Φ関数自体を分岐のpredブロックから消せる. これはレジスタ割当前に行える.生存変数も減ってうれしい.

  生存するとき: Φ関数をレジスタコピーに落としてから処理する必要がある

- エイリアス解析によるメモリアクセスのCSE
  - minrtは配列アクセスが多く, 最適化終盤ではlwが全体の3割ほどに
  - 配列アクセスがあったら, 配列名と定数オフセット → 変数名を登録する.
  - サブルーチンで書き換え可能性がある, オフセットが定数でない場合は
    その配列にまつわる登録を消す. (保守的な評価)
  - 定数オフセットで配列書き換えをされたら, 更新後の変数に束縛を更新.

実装した機能一覧

- フロントエンド (KNormal.t)
  - 共通部分式削除CSE
    - 基本的に即値が共通化されるだけ
  - エイリアス解析
  - while式導入
    - ループ不変式を外に出す . 定数だけ .
  - グローバル配列導入
  - 組型引数展開 : インライン展開と干渉したので使われていない
  - +,-演算子のオーバーロード

- バックエンド (SSA形式)
  - レジスタ割当
  - グローバル値番号付けGVN
  - 分岐で共通の引数設定を分岐前に移動
  - 分岐の畳み込み
  - Φ関数のmvと変数の定義を合体させる
  - BB内スケジューリング
  - ループのブロック出力順変更 : ループのジャンプが1つ減る