

CPU 実験最終発表

3 班 コンパイラ係
深堀 孔明

やったこと

10月

11月

12月

min-caml の最適化

1月

2月

3月

テスト勉強

余興 (CPUExSolver)

やったこと

10月

11月

12月

min-caml の最適化

1月

2月

3月

テスト勉強

余興 (CPUExSolver)

min-caml の最適化

- いろいろやった
 - インライン展開
 - 定数畳み込み
 - 大域的番号付け
 - 1, -1 用専用レジスタを用意
 - グラフ彩色によるレジスタ割り当て
 - グローバル変数の仕組み
 - その他パラメータ調整など
- min-rt の発行命令数: 14.4 億命令

やったこと

10月

11月

12月

min-caml の最適化

1月

2月

3月

テスト勉強

余興 (**CPUExSolver**)

- CPU 実験で作成されるアーキテクチャはどの班も大体同じ
- ジェネレータ作れそう
- 作ってみた

CPUExSolver[つくってみた]

The screenshot displays the CPUExSolver application interface. On the left, the ProjectView shows a tree structure of files including 'lib', 'cls-bug', 'cls-bug2', 'cls-rec', 'even-odd', 'fib', 'float', 'fmul', 'funcomp', 'gcd', 'inprod', 'inprod-loop', 'inprod-rec', 'join-reg', and 'join-reg2'. The main editor window shows the source code for 'even-odd.ml':

```
let t = 123 in
let f = 456 in
let rec even x =
  let rec odd x =
    if x > 0 then even
    if x < 0 then even
  f in
  if x > 0 then odd
  if x < 0 then odd
  t in
print_int (even 789)
```

The 'アーキテクチャの生成' (Architecture Generation) dialog is open, showing configuration options for instruction sets. The '命令セット1' (Command Set 1) tab is selected. The dialog is divided into three columns: '算術演算' (Arithmetic Operations), '算術演算(即値)' (Arithmetic Operations (Immediate)), and '浮動小数演算' (Floating Point Operations). Each column contains a list of operations with checkboxes and input fields for their names.

算術演算	算術演算(即値)	浮動小数演算
<input type="checkbox"/> 値の複製 <input type="text"/> mov	(必須) 上位16bitへの即値代入(整数) <input type="text"/> mvhi	(必須) 値の複製 <input type="text"/> fmov
(必須) 加算 <input type="text"/> add	(必須) 下位16bitへの即値代入(整数) <input type="text"/> mvlo	(必須) 符号反転 <input type="text"/> fneg
(必須) 減算 <input type="text"/> sub	(必須) 即値加算 <input type="text"/> addi	<input checked="" type="checkbox"/> 上位16bitへの即値代入 <input type="text"/> hi
<input checked="" type="checkbox"/> 乗算 <input type="text"/> mul	(必須) 即値減算 <input type="text"/> subi	<input checked="" type="checkbox"/> 下位16bitへの即値代入 <input type="text"/> lo
<input type="checkbox"/> 除算 <input type="text"/> div	<input checked="" type="checkbox"/> 即値乗算 <input type="text"/> muli	(必須) 加算 <input type="text"/> fadd
<input type="checkbox"/> 論理左シフト <input type="text"/> sll	<input type="checkbox"/> 即値除算 <input type="text"/> divi	(必須) 減算 <input type="text"/> fsub
<input type="checkbox"/> 論理右シフト <input type="text"/> srl	(必須) 即値論理左シフト <input type="text"/> slli	(必須) 乗算 <input type="text"/> fmuln
<input type="checkbox"/> 算術左シフト <input type="text"/> sla	<input type="checkbox"/> 即値論理右シフト <input type="text"/> srli	<input checked="" type="checkbox"/> 乗算 + 符号反転 <input type="text"/> fmuln
<input type="checkbox"/> 算術右シフト <input type="text"/> sra	<input type="checkbox"/> 即値算術左シフト <input type="text"/> slai	<input checked="" type="checkbox"/> 除算 <input type="text"/> fdiv
<input type="checkbox"/> 論理シフト(値に応じて左右シフトを切り替える)	(必須) 即値算術右シフト <input type="text"/> srai	<input type="checkbox"/> 逆数 <input type="text"/> finv
<input type="text"/> shift	<input type="checkbox"/> 即値論理シフト(値に応じて左右シフトを切り替える)	<input type="checkbox"/> 逆数 + 符号反転 <input type="text"/> finvn
<input checked="" type="checkbox"/> ビット積 <input type="text"/> and	<input type="text"/> shifti	<input type="checkbox"/> 絶対値 <input type="text"/> fabs
<input checked="" type="checkbox"/> ビット和 <input type="text"/> or	<input checked="" type="checkbox"/> 即値ビット積 <input type="text"/> andi	(必須) 平方根 <input type="text"/> fsqrt
<input checked="" type="checkbox"/> ビット否定論理和 <input type="text"/> nor	<input checked="" type="checkbox"/> 即値ビット和 <input type="text"/> ori	<input type="checkbox"/> 切り捨て <input type="text"/> floor
<input checked="" type="checkbox"/> ビット排他的論理和 <input type="text"/> xor	<input checked="" type="checkbox"/> 即値ビット否定論理和 <input type="text"/> nori	<input type="checkbox"/> 正弦 <input type="text"/> fsin
	<input checked="" type="checkbox"/> 即値ビット排他的論理和 <input type="text"/> xori	<input type="checkbox"/> 余弦 <input type="text"/> fcos

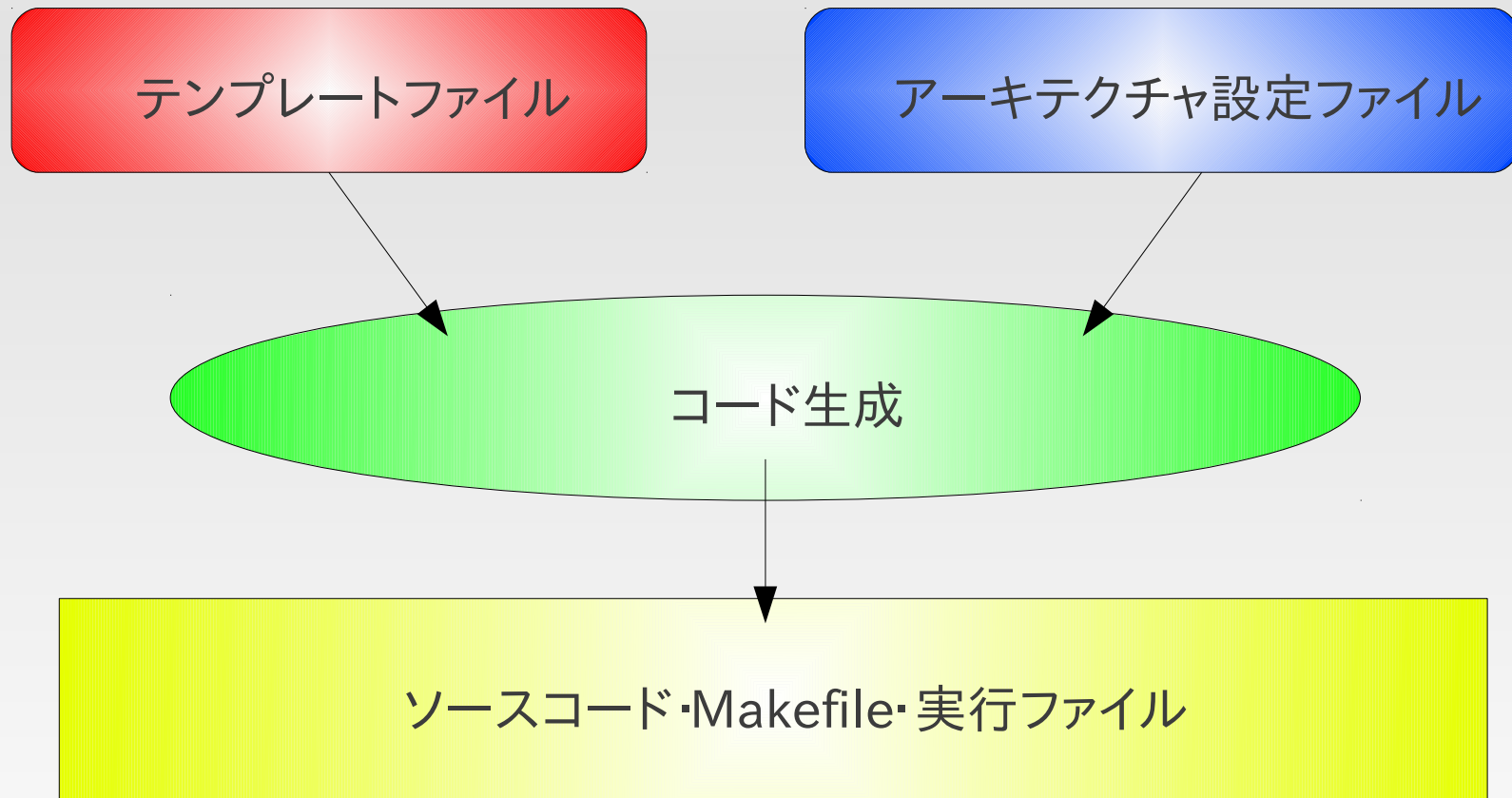
At the bottom of the dialog, there are three buttons: '標準出力' (Standard Output), '標準エラー' (Standard Error), and '入' (Input). A checkbox at the bottom left is checked, labeled '標準エラーもここに表示' (Display standard error here).

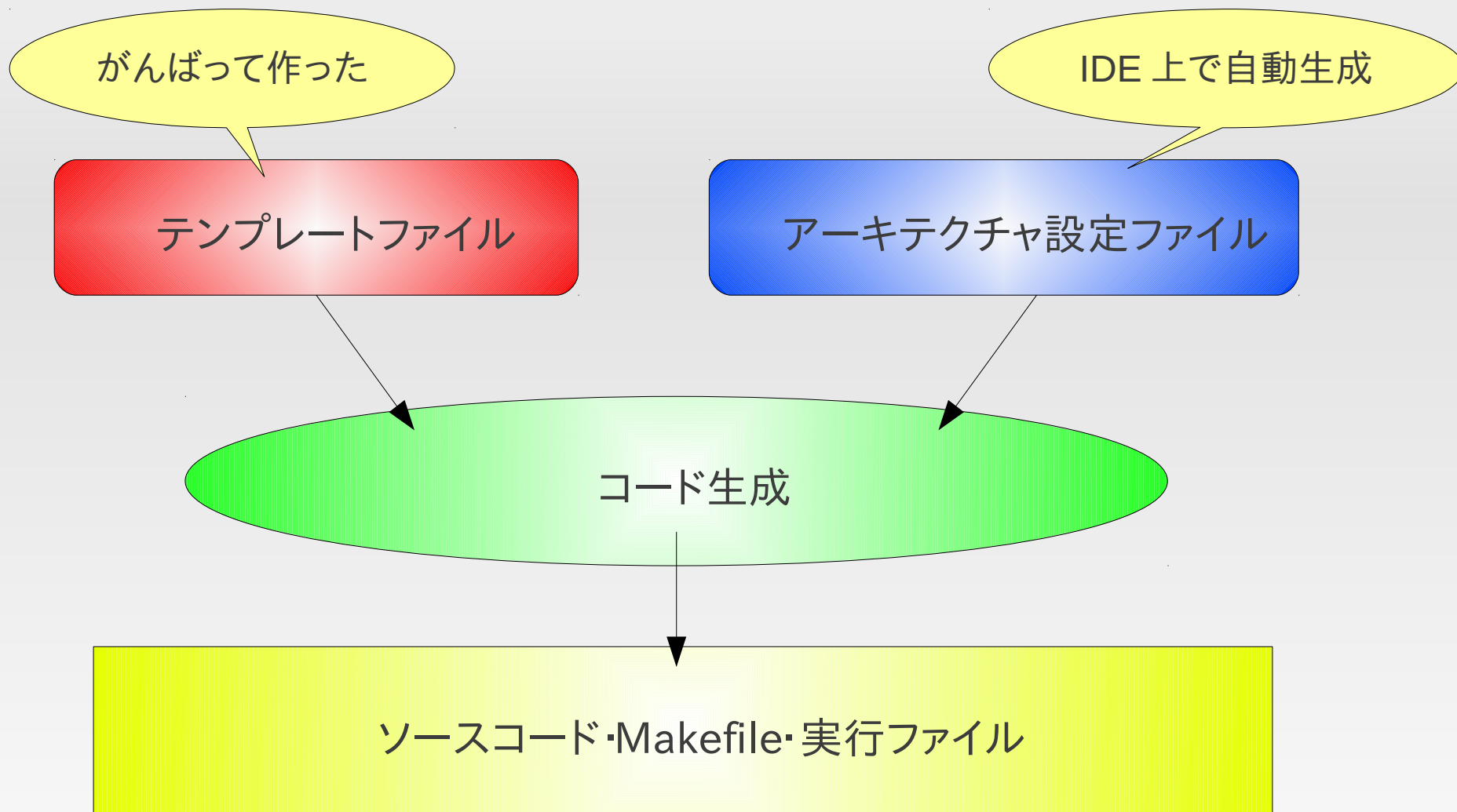
CPUExSolver

- **アーキテクチャ自動生成機能**付き min-caml 用 (なんちゃって) 統合開発環境
 - 設定に応じて以下を自動生成
 - コンパイラ
 - リンカ
 - アセンブラ
 - シミュレータ
 - 各種ライブラリ
 - 命令セット、アドレッシング、レジスタ、etc...
 - アーキテクチャ作成からシミュレートまで GUI 上で完結
- CPU 実験を根底から覆す画期的ツール？

CPUExSolver

- 使用言語など
 - OCaml (コンパイラ)
 - C++ (アセンブラ・シミュレータ)
 - Java (リンカ)
 - Qt4 (GUI)
 - Python + Mako Library (テンプレートファイル)





ユーザは GUI でアーキテクチャの設定をするだけ!

- コード生成には **Mako** を使用
- **Mako**
 - Python 製のテンプレートエンジン
 - Python の機能を使ってテンプレートが書ける

```
% for i in range(0, 5)  
まこっ!  
% endfor
```

sample.txt.tmpl



```
まこっ!  
まこっ!  
まこっ!  
まこっ!  
まこっ!
```

sample.txt

デモ

- アーキテクチャの制約が大きい
 - 32bit の固定長。R、I、J 形式のみサポート
 - 条件レジスタによる分岐はできない
- コアは生成できない
- デバッグが不十分
 - 設定によってはバグるかも
- アセンブラ・シミュレータが低機能
 - ~~デバッグ機能は甘え~~ ただの手抜き
- UI 部分は未完成 (洗練されていない)

- 他班のアーキテクチャで動かすことは考えてない
 - 命令セットが特殊だったり一貫性がなかったりする
- 非効率な設定はあえてできないようにした
(例) 条件レジスタ(命令数が倍になる?)
- CPUExSolver を使えば一貫性がある効率的なアーキテクチャが作れる?

- コンパイラ・シミュレータ系の仕事を**全部**やってくれる
- コンパイラはそこそこ高性能
 - min-rt : 14 億 ~ 20 億命令
 - 今年度のコンパイラ系向け課題も実装済み!
 - グラフ彩色によるレジスタ割付
- レベル80くらいから CPU 実験を始められる?
- **ぜひ使ってください! > 12er**

CPUExSolver

- ソースコードは github に公開中
 - <https://github.com/furaga/CPUExSolver>
- 夏休みの CPU 実験を語る会までには完成版をリリースします

ご清聴ありがとうございました

- 各班のアーキテクチャへの対応状況

	コンパイル可能	バイナリの仕様に対応	備考
1班	○ (15.0億命令)	×	RAM・バイナリ仕様
2班	×	×	32bit 長でない
3班	○ (14.3億命令)	○	完全にサポート
4班	?	?	未確認
5班	×	×	条件レジスタを使用
6班	○ (17.6億命令)	△	6班のシミュレータで動かす

(補足) min-caml の最適化

コンパイラの最適化について

- %g28 を 1, %g29 を -1 で固定 (if 文を最適化)
- コンパイル時間を短縮。それによってインライン展開のパラメータ引き上げ
 - Asm.fv を高速化

関数呼出

- Callee で殺されないレジスタは退避しないようにしました。
 - 各関数内で使用されるレジスタを記憶
- グラフ彩色によるレジスタ割り当て(やや簡略)
 - 後退辺無視(再帰関数特有の最適化はされない)
 - クロージャが存在しないときのみ行う
 - 存在するならオリジナルの `regAlloc.f` を使用

コンパイラの最適化について

- グローバル変数の仕組みを用意
- トップレベルの配列・タプルのアドレスはコンパイル時に決まる
 - 配列・タプルのアドレスをメモリに退避する必要ない
 - \rightarrow クロージャを削除
- min-rt からクロージャを完全に駆逐

その他

- %g28 を 1, %g29 を -1 で固定 (if 文を最適化)
- コンパイル時間を短縮。それによってインライン展開のパラメータ引き上げ
 - Asm.fv を高速化