CPU 実験最終発表

3 班 コンパイラ係
深堀 孔明

やったこと

min-caml の最適化

テスト勉強

余興（ CPUExSolver ）

やったこと

min-caml の最適化

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

min-caml の最適化

テスト勉強

余興（**CPUExSolver**）

余興

動機

- 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]

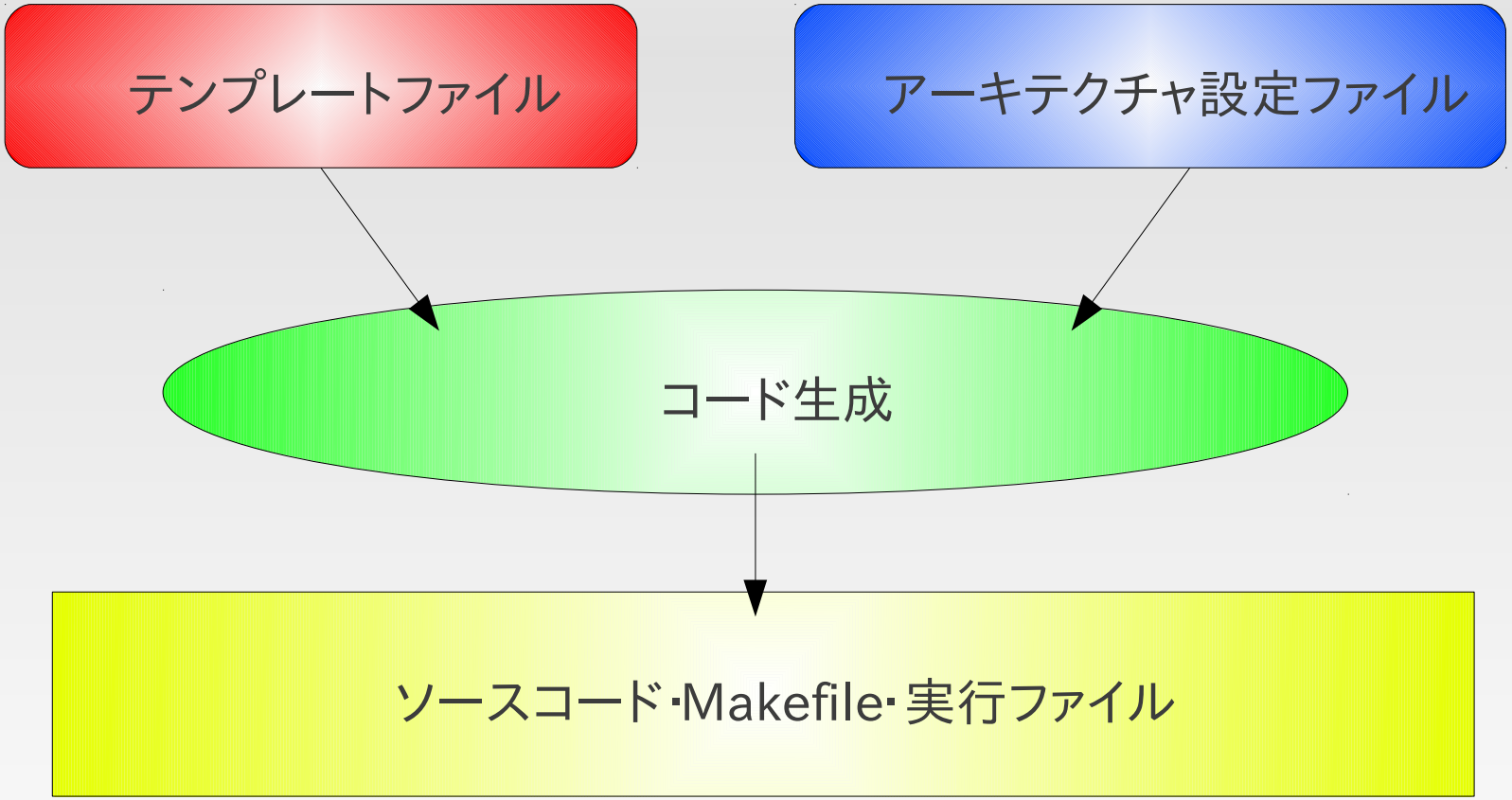
CPUExSolver

- 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？

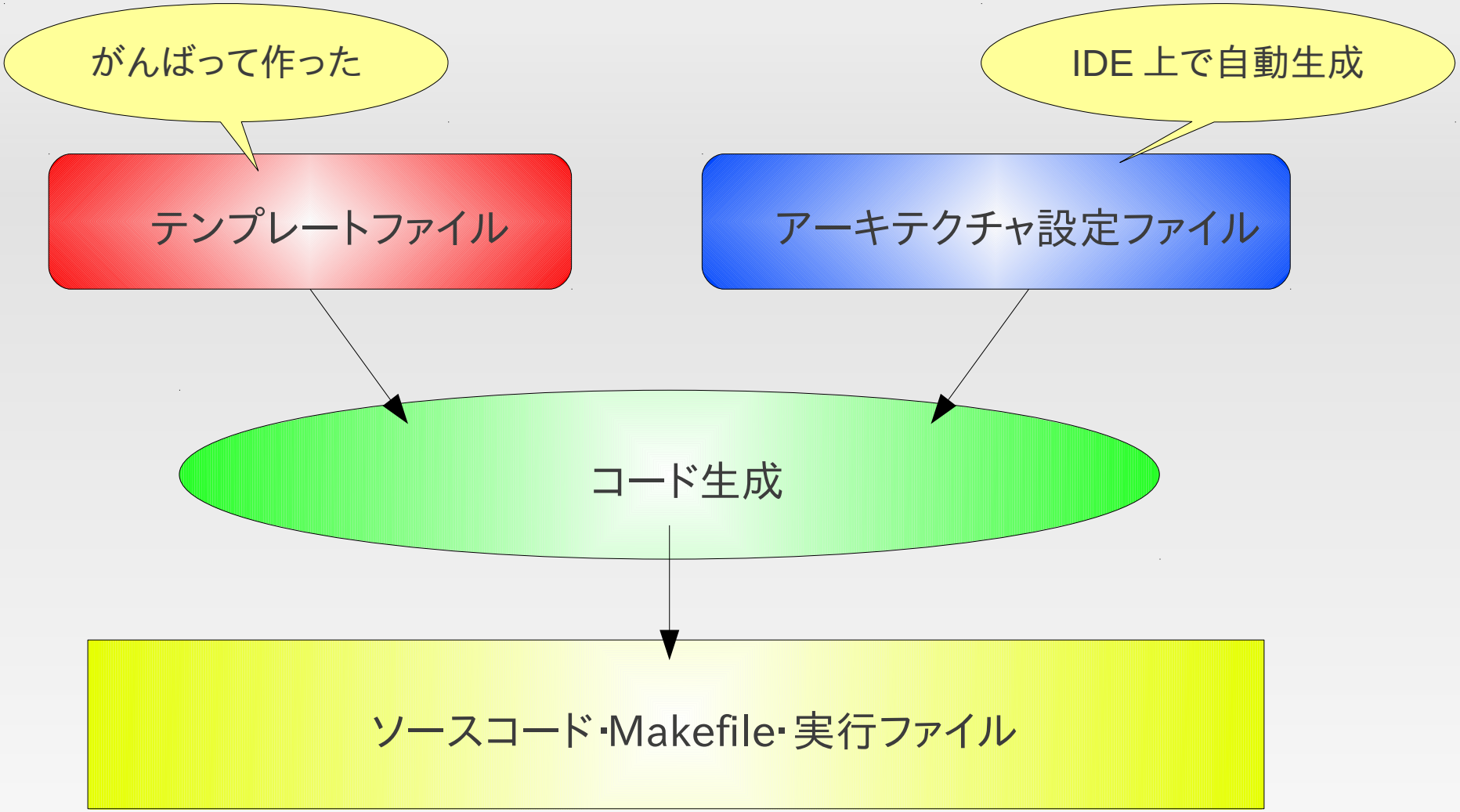
CPUExSolver

- 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）

CPUExSolver　ソルバー機能の構造



CPUExSolver　ソルバー機能の構造



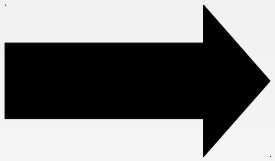
ユーザは GUI でアーキテクチャの設定をするだけ!

CPUExSolver コード生成

- 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

CPUExSolver 問題点

- 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）

CPUExSolver　補足

- 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？

CPUExSolver　　ステマ

- コンパイラ･シミュレータ係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係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

CPUExSolver　補足

- 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ず

（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
  - ー＞ 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