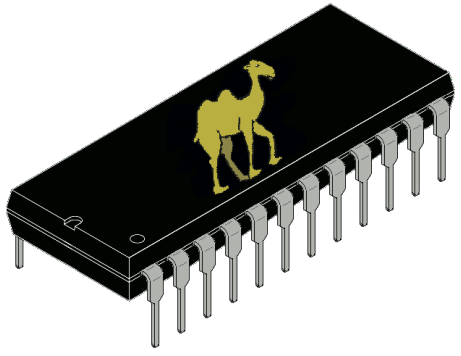
JFLA 2011

Programmer les microcontrôleurs PIC en Objective Caml



Benoît Vaugon

Philippe Wang & Emmanuel Chailloux

Plan

Introduction

Réalisation

Conclusion

Un nouveau *backend* pour OCaml

Pour quelle cible ?

Les microcontrôleurs PIC

Qu'est-ce ?

- Circuit intégré programmable
- Fait pour les applications embarquées

Domaines d'application

- Robotique industrielle, gadgets électroniques, domotique (appareils électroménagers), cartes à puce...

Les PIC : une cible contrainte

Caractéristiques du PIC18F4620 :

1. 3968 registres de **1 octet**
2. Architecture : RISC **8 bits**
3. Mémoire programme : **64 Ko**

Le défi : des ressources « limitées » pour :

- Un langage multi-paradigme : impératif, fonctionnel, objet
- Un système de modules de haut niveau, foncteurs
- Une gestion automatique de la mémoire
- Un mécanisme d'exceptions
- Polymorphisme et comparaisons (`compare`)

Jusqu'où irons-nous ?

La programmation sur PIC (avant)

Assembleur

-
```
EXEMPLE_DE_CODE:
        tblrd*+
        addwf     FSR0L, W
        rcall     SOUS_PRGM
        btfsc     STATUS, C
        incf      FSR0H, F
        return
```

Compilation en natif

- Basic (BASIC84plus)
- C (sdcc, SIXPIC, MCC18, Hi-Tech C)
- Forth (PicForth)

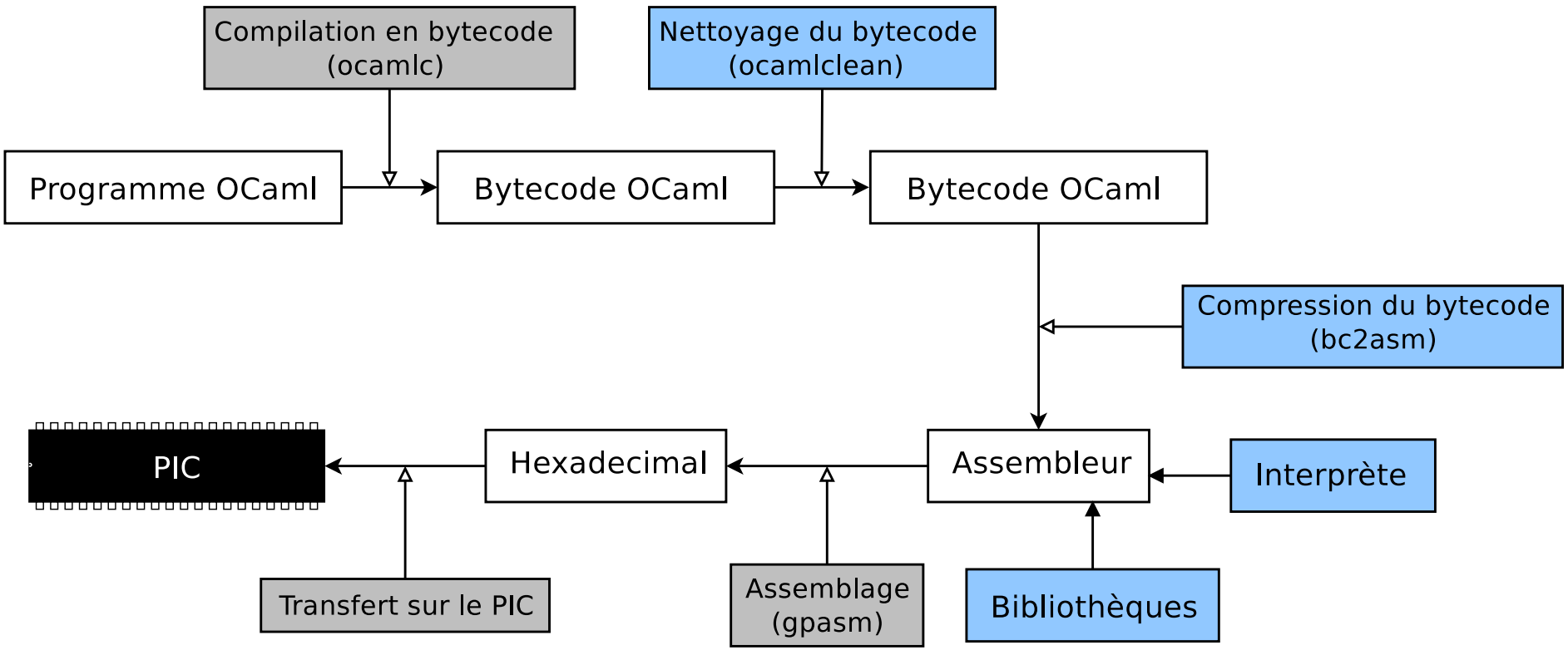
Interprètes

- Basic (BasicStamps)
- Forth (FlashForth)

Utilisation d'une machine virtuelle

- PICOBIT (machine virtuelle Scheme pour PIC)

Chaîne de production



OCamlClean

Une analyse statique originale du code-octet [1]

Buts :

- Réduire l'occupation du tas (mémoire dynamique) : élimination des fermetures non utilisées (bibliothèques)
  - $\Rightarrow$ nettoyer le code-octet (mémoire statique)
  - $\Rightarrow$ accélérer l'initialisation de la machine virtuelle
- Globalisation des fermetures (réduire les indirections)
  - $\Rightarrow$ accélérer l'exécution du code

Un programme vide se compile en...
un programme vide

1. // Jérôme Vouillon

Du code-octet à l'assembleur

Code octet généré par `ocamlc` (aligné sur 32 bits)
$\rightarrow$ mémoire statique du PIC (aligné sur 1 octet)

Compression du code-octet

- Élimination des 0 (de 4 à 1 ou 2 octets)
  - $\Rightarrow$ Gain d'un facteur **3,5**

Analyse statique de l'état initial du tas

$\Rightarrow$ Pas de calcul à l'initialisation, simple copie

Lien avec la bibliothèque d'exécution

- Fonctions externes (codées en assembleur)
  - Élimination des inutiles
  - Accès direct

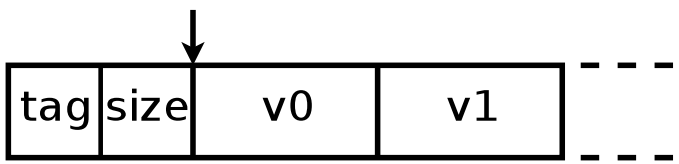
OCamlrun sur PIC

Réécriture de la machine virtuelle OCaml en assembleur PIC

```
wc -l interp.asm runtime.asm
  2556 interp.asm
   368 runtime.asm
  2924 total
```

Représentation uniforme sur 16 bits

- Entiers codés sous la forme $2 \times n + 1$ (bit de tag)
- Blocs :



Gestionnaire mémoire : *Stop & Copy*

- Simple (150 loc assembleur)
- Performant (1 exécution < 1,5ms)
- Division de la taille du tas par 2

Bibliothèque standard

```
wc -l stdlib.asm
  6757 stdlib.asm
```
+ entre 150 et 200 lignes OCaml

Modules copiés	Modules modifiés	Modules non-portés
Array, ArrayLabels, CamlinternalLazy, CamlinternalMod, Char, Hashtbl, Int32, Int64, Lazy, List, ListLabels, MoreLabels, OO, Queue, Set, Sort, Stack, StdLabels, String, StringLabels	Buffer$^+$, CamlinternalOO$^+$, Gc$^+$, Map$^+$, Obj$^+$, Pervasives*, Random, Printf*, Std_exit, Sys*	Arg, Complex, Callback, Digest, Filename, Format, Genlex, Marshal, Nativeint, Parsing, Printexc, Scanf, Stream, Weak

Bibliothèques spécifiques(1)

Entrées-sorties du PIC

```
external read_reg : reg -> int = "caml_pic_read_reg";;
(** [read_reg reg] reads value of the Special Function Register [reg].
    Returns value between 0 and 255. *)

external write_reg : reg -> int -> unit = "caml_pic_write_reg";;
(** [write_reg reg value] writes value in the Special Function Register [reg].
    [value] should be between 0 and 255. *)
```

Interruptions

```
external set_interruption_handler : (bit -> unit) -> unit =
  "caml_set_interruption_handler";;
(** [set_interruption_handler h] registers the interruption handler [h]. *)

external clear_interruption_handler : unit -> unit =
  "caml_set_interruption_handler";;
(** [clear_interruption_handler ()] removes the interruption handler. *)
```

Bibliothèques spécifiques(2)

Gestion des afficheurs LCD

```
module Aff = Lcd.Connect (
  struct
    module Pic = Pic
    let bus_size = Lcd.Four
    let e  = Pic.LATD0
    let rs = Pic.LATD2
    let rw = Pic.LATD1
    let port = Pic.PORTC
  end
);;
Aff.init ();;
Aff.config ();;
Aff.print_string "Hello world";;
```

Gestion des connexions séries

```
let (ic : t in_channel) = Serial.open_in () in
  while true do
    match Serial.receive ic with
      | Clear -> clear ()
      | Moveto (l, c) -> moveto l c
      | Int i -> print_int i
      | Text t -> print_string t
  done
```

Étude de performance

Vitesse d'exécution

- **400 000** instructions VM par seconde
- **25** cycles machine par instruction VM
  par rapport à **18** sur un processeur standard
- Comparaison avec PICOBIT :
  - **37000** instructions VM par seconde pour PICOBIT
  - soit « 11 fois moins rapide » que OCaPIC
    (mais pour des code-octets différents)

Occupation de la mémoire statique

- Taille totale : **64 Ko** pour le PIC18F4620
- Interprète de code-octet : **4954 o** (7,6%)
- Bibliothèque d'exécution : **4254 o** (6,5%)
- Code-octet : jusqu'à **55 Ko** (86%)

Conclusion

Application fonctionnelle

- Distribuée en open-source (licence CeCILL-B)
  `http://www.algo-prog.info/ocaml_for_pic/`

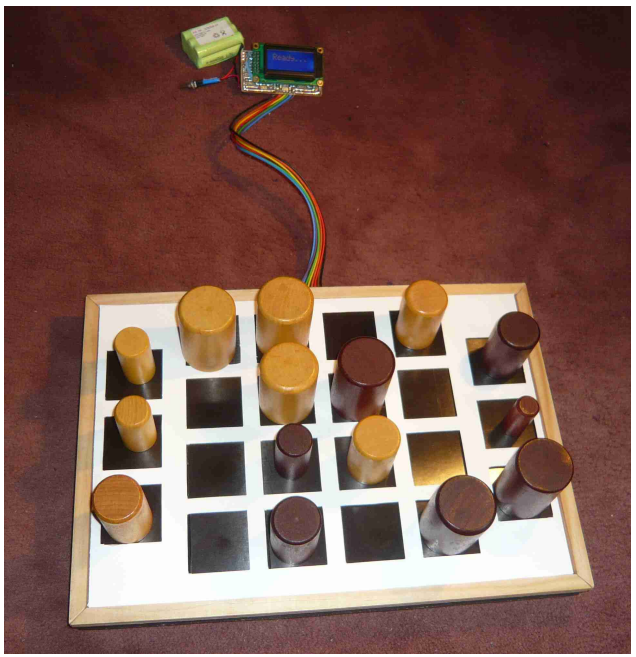
De bonnes performances

Perspectives

- Adapter à d'autres modèles de programmation
- Porter sur d'autres architectures
- Écrire d'autres ramasse-miettes
- Gérer une RAM externe
- ...

Exemple d'application

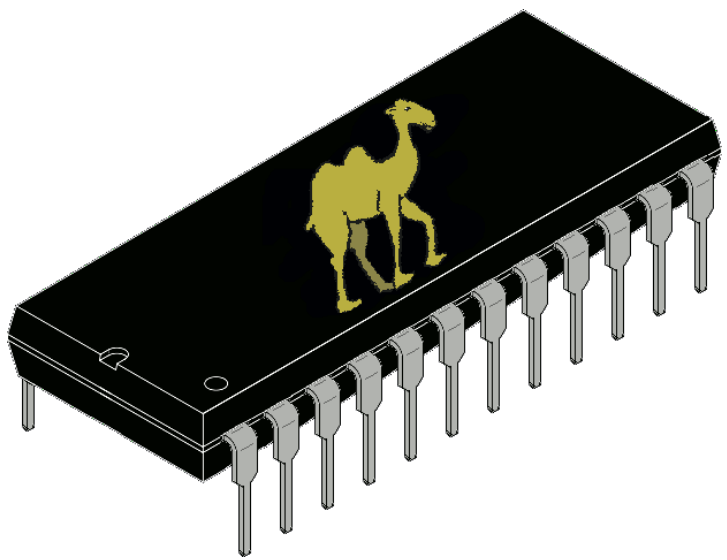
- Un jeu de réflexion se jouant à deux
- Programmation d'une IA sur PIC
- Entièrement codé, réalisé et testé



Questions

?

? ?



? ?

?

Bibliographie

Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon.

The Objective Caml system (release 3.11) : Documentation and user's manual.

Institut National de Recherche en Informatique et en Automatique, 2008.

`http://caml.inria.fr/pub/docs/manual-ocaml/`.

Microchip.

PIC18F2525/2620/4525/4620 Data Sheet, 2008.

`http://ww1.microchip.com/downloads/en/devicedoc/39626b.pdf`.

Vincent St-Amour and Marc Feeley.

Picobit : A compact scheme system for microcontrollers.

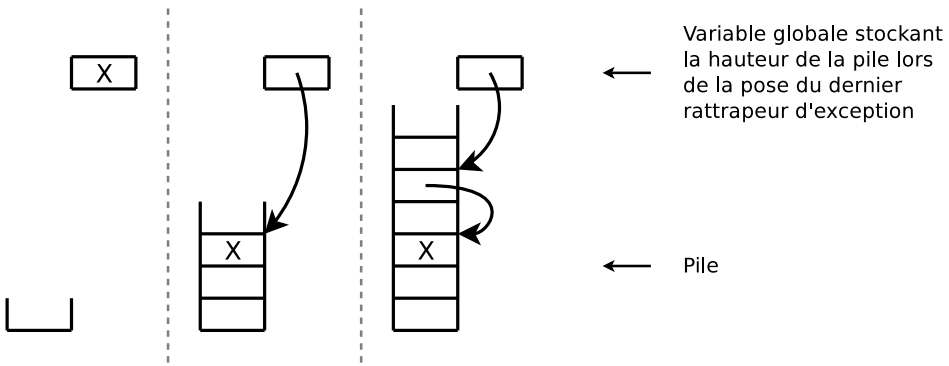
Implementation and Application of Functional Languages (IFL'09), pages 1–11, September 2009.

Gestion des exceptions

Plusieurs techniques

1. Rien lors de la pause des rattrapeurs, tout au lancement (exemple : Java)
2. Stockage des rattrapeurs dans la pile (exemple : OCaml)

Choix 2 : comme en OCaml



Gestion par trois instructions VM

- `pushtrap` / `poptrap`
- `raise`

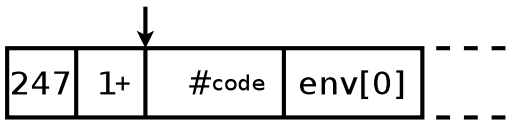
Gestion des fermetures

Qu'est-ce ?

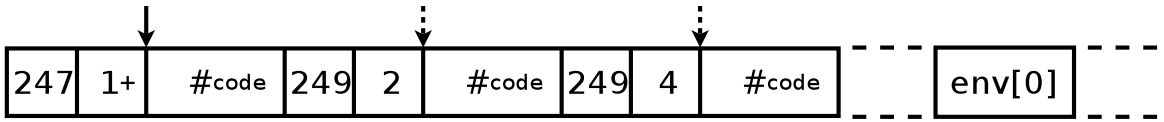
- Permet de représenter les fonctions
- Stocke l'environnement
- Stocke les arguments déjà passés en cas d'appel partiel

Deux types de fermetures

- Fermetures simples



- Fermetures récursives



Géré par 5 instructions de la machine virtuelle

- `apply` / `appterm`
- `return`
- `grab` / `restart`

Gestion automatique de la mémoire

Qu'est ce ?

- Libère la mémoire qui n'est plus utilisée
- Dans certains cas compacte la mémoire

Plusieurs types de ramasse miettes

- *Stop & Copy*
- *Mark & Sweep*
- *Mark & Compact*

Choix : *Stop & Copy*

- Deux phases :
  1. Parcours des racines
  2. Parcours du tas TO
- Optimise l'utilisation des registres
- N'améliore pas la localité spaciale