OUD - 2012

OCamlICC

Raising Low-Level Bytecode to High-Level C

Michel Mauny, **Benoît Vaugon**

Ensta-ParisTech

Introduction

What we want to do:

- Translate OCaml bytecode into C code.

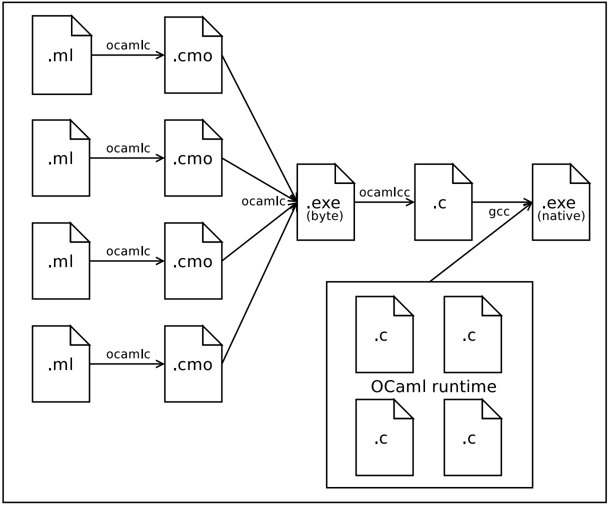
Constraint:

- Use the standard OCaml runtime.

Goals and side effects:

- Native code performances with bytecode portability.
- Post-compilation of bytecode for end users.

Compilation chain



Generating C From Bytecode (1)

Translation in 3 steps:

- Parsing bytecode executables.
- Performing some code transformations.
- Generating one C source file.

Decompilation:

- Translate each λ-abstraction into one C function

Optimizations:

- Whole program analysis and optimizations.
- Do not optimize code directly, generate optimizable C code.
- Static analysis based on *abstract interpretation*.
- Main optimizations performed:
    - Forward code pointers.
    - Remove creation of some unused closures.
    - Reduce sizes of closure environments.
    - Move values from the OCaml bytecode stack to C stack.

Generating C From Bytecode (2)

Move values from the OCaml bytecode stack to the C stack

- Transform OCaml stack cells into C local variables.
- Warning: OCaml copying GC may move memory blocks.
- A stack cell can be extracted from the stack if:

  It is never read as a heap pointer.
  or It is never written as a heap pointer.
  or No garbage collection may occur during its lifetime.
- Note that some heap pointers are safely extracted from the stack.
- Effectiveness:
  - `ocamlc` bootstrap: extraction of 85% of stack cells.

Issues (1)

Exceptions

- Use C setjmp/longjmp.
- C++ try-catch available as an option.

Tail calls

- Correct implementation of tail calls.
- GCC does not implements correctly tail calls when:

  there is a call to setjmp in then same scope
  or the callee receives more arguments than the caller.

- We have two implementations:
  - Architecture and GCC specific solution: assembly code.
  - Pure C solution:
    - Sub-scoping setjmp calls in local functions.
    - Using globals to pass arguments.

Issues (2)

Signal handling

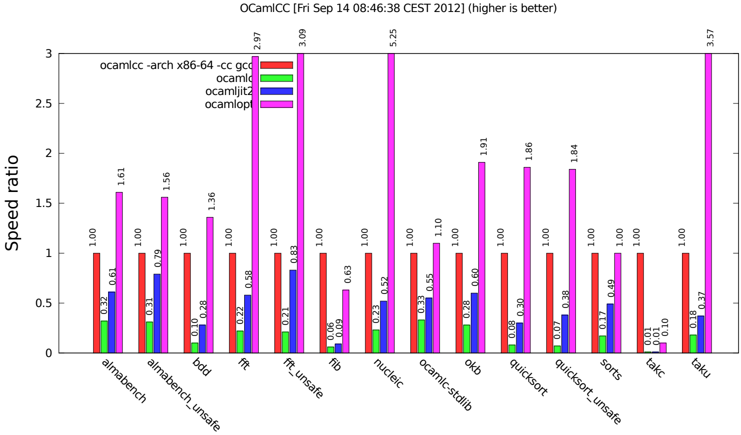
- Must preserve memory consistency when calling a handler.
- Principle: polling a global flag.
- Compilation option to choose between reactivity and performance.

C compilation resources

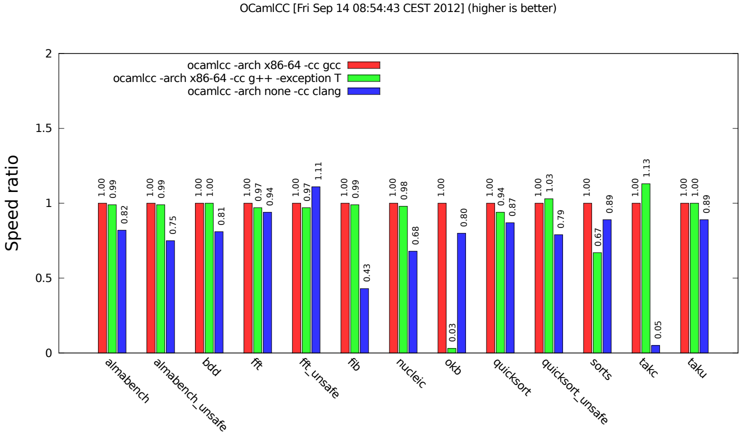
- Generation of a single C file that #includes the OCaml runtime.

  ⇒ Huge C file: more than 10^6 C instructions for `ocamlc` bootstrap.
- Managable, so far.
- Separate compilation may be available as an option in future versions.

Performances (1)



Performances (2)



OCamlCC [Fri Sep 14 08:54:43 CEST 2012] (higher is better)

Conclusion

Distribution

- `https://github.com/ocaml-bytes/ocamlcc`

Supported C compilers

- `gcc` (default)
- `g++`
- `clang`

As portable as the OCaml bytecode

Good performances

Future work

- Peephole optimizations:
  - Floating point arithmetics.
  - Other standard bytecode patterns.
- Other backends: icc, . . .