OCaml 5, une évolution majeure dans la douceur

Florian Angeletti
Inria

29 Septembre 2022

OCaml 5?

- 10 ans après OCaml 4, une nouvelle version majeure
- OCaml 5 =
  - OCaml 4.14
  - + nouveau environnement d'exécution
  - + nouveau module Domain: parallélisme
  - + nouveau module Effects: effets algébriques
  - - des fonctions et modules dépréciés

OCaml 4 et les fils d'exécutions

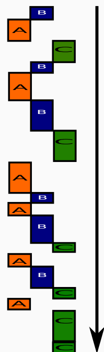
Dans OCaml 4:

```
let t_A = Thread.create task_A ()
let t_B = Thread.create task_B ()
let t_C = Thread.create task_C ()
let () =
  Thread.join t_A;
  Thread.join t_B;
  Thread.join t_C
```

OCaml 4 et les fils d’exécutions

Dans OCaml 4:

```
let t_A = Thread.create task_A ()
let t_B = Thread.create task_B ()
let t_C = Thread.create task_C ()
let () =
  Thread.join t_A;
  Thread.join t_B;
  Thread.join t_C
```



OCaml 4 et les fils d'exécutions

Dans OCaml 4:

```
let t_A = Thread.create task_A ()
let t_B = Thread.create task_B ()
let t_C = Thread.create task_C ()
let () =
  Thread.join t_A;
  Thread.join t_B;
  Thread.join t_C
```

- Concurrence: l'exécution des tâches A, B et C est entremêlée
- Pas de parallélisme: Un verrou global assure qu'un seule tâche OCaml s'exécute à chaque instant

Parallélisme dans OCaml 4

- Plusieurs tâches C en utilisant la FFI
- Plusieurs processus:
  - pas de mémoire partagée (facilement)
  - communication par messages
- Parallélisme à mémoire partagée: un manque réel

La longue quête du multicœur

- Plusieurs tentatives précédentes pour enlever le verrou global
- Enlever un verrou global demande de réécrire l'environnement d'exécution
- Une initiative relancée en 2014
- Plusieurs obstacles majeurs:
  - GC parallèle et complexité
  - Modèle mémoire et contentions de donnée (data races)
  - compatibilité avec le code existant
  - compatibilité avec les performances d'OCaml 4

Un mouvement de fond

- Intégration commencée en interne en 2019
- Accélérée en 2020-2021 (OCaml 4.13)
- 2022: le grand saut dans le monde multicœur

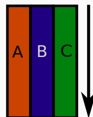
Domaines et parallélisme

Domaines et parallélisme

- Une nouvelle unité de parallélisme, les domaines
- Un domaine: ensemble de fils d'exécutions avec une zone mémoire propre
- Chaque domaine s'exécute en parallèle et coopère sur les phases de GC

```
let d_A = Domain.spawn task_A
let d_B = Domain.spawn task_B
let d_C = Domain.spawn task_C
let () =
  Domain.join d_A;
  Domain.join d_B;
  Domain.join d_C
```

```
let d_A = Domain.spawn task_A
let d_B = Domain.spawn task_B
let d_C = Domain.spawn task_C
let () =
  Domain.join d_A;
  Domain.join d_B;
  Domain.join d_C
```



Utiliser des domaines

- Les domaines sont des constructions bas-niveau plus destiné aux bibliothèques
- Les domaines sont coûteux à initialiser: mieux vaut les initialiser une seule fois
- Des bibliothèques haut niveau construites au dessus
- Un exemple: Domainslib

Utiliser Domainslib

```
let num_domains = Domain.recommended_domain_count - 1
let pool = Domainslib.Task.setup_pool ~num_domains ()
let pfor () = Domainslib.Task.parallel_for
    ~start:0 ~finish:100
    ~body:(fun i -> ... )
let () = Domainslib.Task.run pool pfor
```

Un modèle mémoire pour les cas pathologiques

```
let done' = ref false
let result = ref 0
let task_1 () =
  result := 1;
  done' := true
let task_2 () = while not !done' do Domain.cpu_relax () done;
  !result
let d1 = Domain.spawn task_1
let d2 = Domain.spawn task_2
let final = Domain.join d1; Domain.join d2
```

Quelle valeur pour final?

- 1?
- 0?
- 42??

Quelle valeur pour final?

- 1
- 0
- 42

Synchronisations entre lecture et écriture

```
let done' = Atomic.make false
let result = ref 0
let task_1 () =
  result := 1;
  Atomic.set done' true
let task_2 () = while not (Atomic.get done') do Domain.cpu_relax () done;
  !result
let d1 = Domain.spawn task_1
let d2 = Domain.spawn task_2
let final = Domain.join d1; Domain.join d2
```

Comment comprendre le modèle mémoire?

La solution la plus simple: éviter d'avoir à le comprendre. Deux options:

- Pas de valeur mutable partagée entre domaine
- Pas de data races sur des valeurs mutables partagées

- Data races: lire précautionneusement le manuel

Effets algébriques

Effets algébriques

- Effets algébriques: fonctionnalité expérimentale
- Un système d'effets est en cours de conception

Pourquoi les effets?

- Séparer la définition et l'utilisation des effets de leur interprétation

```
type _ Effect.t += Random: int Effect.t
let rec sum n =
  if n = 0 then 0
  else Effect.perform Random + sum (n-1)
```

Quel sens donner à Random?

```
open Effect.Deep
type ('a,'b) handler = 'b Effect.t -> (('b, 'a) continuation -> 'a) option
let randomc (type a): (_,a) handler = function
| Random -> Some (fun k -> continue k  (Random.int 100))
| _ -> None
let random = { effc = randomc }
let example_0 = try_with sum 100 random
```

Une autre interprétation

```
let not_so_randomc (type a): (_,a) handler = function
  | Random -> Some (fun k -> continue k 0)
  | _ -> None
let not_so_random = { effc = not_so_randomc }
let example_1 = try_with sum 100 not_so_random
```

Effets et concurrence

- Effets algébriques: continuations délimités
- Une manière de manipuler des calculs en cours d'exécution
- Une solution pour permettre d'écrire des schedulers en espace utilisateur
- Un exemple: Eio, bibliothèque d'IO concurrentes en style direct

Compatibilité?

Que faut-il faire pour rendre du code existant compatible avec OCaml 5?

Rien

* Dépréciation

- modules: Stream, Genlex, Pervasives, ThreadUnix
- valeurs:

Array.create, Array.make_float, Array.create_matrix, Bytes.uppercase, Bytes.lowercase, Bytes.capitalize, Bytes.uncapitalize, Char.lowercase, Char.uppercase, Filename.temp_dir_name, Int32.format, Int64.format, Nativeint.format, Format.bprintf, Format.kprintf, Format.set_all_formatter_output_functions, Format.get_all_formatter_output_functions, Format.pp_set_all_formatter_output_functions, Format.pp_get_all_formatter_output_functions, Format.pp_open_tag, Format.pp_close_tag, Format.open_tag, Format.close_tag, Format.formatter_tag_functions, Format.pp_set_formatter_tag_functions, Format.pp_get_formatter_tag_functions, Format.set_formatter_tag_functions, Format.get_formatter_tag_functions, Gc (mutability of the fields of type Gc.control), Lazy.lazy_from_fun, Lazy.lazy_from_val, Lazy.lazy_is_val, Obj.set_tag, Obj.truncate, Obj.final_tag, Obj.extension_constructor, Obj.extension_name, Obj.extension_id, Scanf.stdib, Scanf.fscanf, Scanf.kfscanf, Stdlib.(&), Stdlib.(or), String.set, String.copy, String.fill, String.unsafe_set, String.unsafe_fill, String.uppercase, String.lowercase, String.capitalize, String.uncapitalize, Thread.kill, Thread.wait_write, Thread.wait_read, (.[]<-)

Rien*

† architecture et OS “exotiques”

- Seulement x86-64 et ARM64 supportés
- Seulement Linux, Windows (Mingw64), et MacOS
- (*BSD et OmniOS fonctionnent)
- La branche 4.14 sera supportée jusqu’à la restauration du support des autres architectures et OS

Rien*†

‡ API interne de l’environnement d’exécution

- pointeurs nus
- utilisation des rouages internes de l’environnement d’exécution

Rien*†‡

OCaml 5.0.0 alpha1 Une alpha en progression

- une bonne partie de l'écosystème opam fonctionne déjà sur OCaml 5
- http://check.ocamllabs.io
- Du code OCaml séquentiel valide en OCaml 4 reste valide en OCaml 5
- Même profil de performance

conf-pkg-config.2					3751
base-unix.base					3621
conf-m4.1					3618
base-threads.base					3617
conf-libX11.1					3617
ocaml-secondary-compiler.4.08.1-1					3617
ocamlfind.1.9.5					3617
csexp.1.5.1					3616
dune.3.4.1					3616
dune-configurator.3.4.1					3616
graphics.5.1.2					3616
ocamlfind-secondary.1.9.1					3616
result.1.5					3616
ocamlbuild.0.14.1					3133
base-bytes.base					2673
stdlib-shims.0.3.0					2591
cppo.1.6.9					2519
ppx_derivers.1.2.1					2518
topkg.1.0.5					2511
sexplib0.v0.15.1					2499
conf-which.1					2495
seq.base					2487
ocaml-compiler-libs.v0.12.4					2474
ppxlib.0.27.0					2446
ocaml-migrate-parsetree.2.4.0					2415
base.v0.15.1~5.0preview					2410
num.1.4					2405
stdio.v0.15.0					2385
base-native-int63.0.1					2382
ppx_tools.6.5					2283
cppo_ocamlbuild.1.6.9					2252
octavius.1.2.2					2198
conf-perl.2					2045
ounit.2.2.6					2043
re.1.10.4					2025
cmdliner.1.1.1					2016
ounit2.2.2.6					2008
conf-libpcre.1					2007
pcre.7.5.0					2005
type_conv.113.00.02					1963
conf-glade.2					1933
conf-gnomecanvas.2					1933
conf-gtk2.1					1933
ablgtk.2.18.12					1932
achar.0.0.2					1924
ppx_sexp_conv.v0.15.1					1907
conf-findutils.1					1896
ocaml-src.5.0.dev					1877
sexplib.v0.15.1					1874
minios-xen.0.9					1873
mirage-xen-minios.0.9.3					1872
parsexp.v0.15.0					1872
mirage-xen-posix.3.3.1					1871
ppx_compare.v0.15.0					1865
ppx_inline_test.v0.15.0					1859
s-build-tools.113.33.04					1853
ppx_here.v0.15.0					1853
ppx_hash.v0.15.0					1851
ppx_optcomp.v0.15.0					1850
ppx_cold.v0.15.0					1848
ppx_enumerate.v0.15.0					1846
ppx_assert.v0.15.0					1845
ppx_base.v0.15.0					1845
jane-street-headers.v0.15.0					1842
st-config.v0.15.1					1842
time_now.v0.15.0					1841
conf-libssl.4					1839
ocplib-endian.1.2					1838
ssl.0.5.12					1836
base-bigarray.base					1835
bigarray-compat.1.1.0					1834
ppx_fixed_literal.v0.15.0					1831
conf-libev.4-12					1828
ocaml-syntax-shims.1.0.0					1825
fieldslib.v0.15.0					1819
mmap.1.2.0					1818
ppx_fields_conv.v0.15.0					1818
ppx_sexp_message.v0.15.0					1818
variantslib.v0.15.0					1818
ppx_let.v0.15.0					1817
ppx_variants_conv.v0.15.0					1817
ppx_custom_printf.v0.15.0					1816
react.1.2.2					1816
bin_prot.v0.15.0					1815
ppx_bench.v0.15.0					1815
base_quickcheck.v0.15.0					1814
wt.5.6.1					1814
ppx_sexp_value.v0.15.0					1814
splittable_random.v0.15.0					1814
typerep.v0.15.0					1814
ppx_bin_prot.v0.15.0					1813
ppx_typerep_conv.v0.15.0					1813
ppx_disable_unused_warnings.v0.15.0					1812
ppx_ignore_instrumentation.v0.15.0					1812
ppx_jane.v0.15.0					1812
ppx_log.v0.15.0					1812
ppx_module_timer.v0.15.0					1812
ppx_optional.v0.15.0					1812
ppx_pipebang.v0.15.0					1812
ppx_stable.v0.15.0					1812
ppx_string.v0.15.0					1812
conf-ncurses.1					1811
uutf.1.0.3					1807
ext.0.8.1					1798
int_repr.v0.15.0					1791
base_bigstring.v0.15.0					1790
fileutils.0.6.3					1754
core_kernel.v0.15.0					1749
conf-gmp.4					1744
camlp-streams.5.0.1					1738
menhir.20220210					1737
ocamlify.0.0.1					1735
menhirLib.20220210					1734
menhirSdk.20220210					1734
ocamlmod.0.0.9					1734
gettext.0.4.2					1728

Comment commencer à utiliser OCaml 5?

- Installer la alpha avec

```
opam update
opam switch create 5.0.0~alpha1
opam repo add alpha git+https://github.com/kit-ty-kate/opam-alpha-repositor
```

- Installer Domainslib

```
opam install domainslib
```

Le prix à payer dans la bibliothèque standard

- Structures de données mutables
- État mutable interne

Structure de données mutables

- synchronisation à la charge de l'utilisateur
  - Array
  - Bytes
  - Buffer
  - Stack
  - Queue
  - Hashtbl
  - ...

Erreurs à éviter

```
let g = Stack.create ()
let task n () = Stack.push n g; ignore (Stack.pop g)
let test () =
  let l = List.init 50 Fun.id in
  let ds = List.map (fun n -> Domain.spawn (task n)) l in
  List.iter Domain.join ds
let () = while true do test () done
```

```
Fatal error: exception Stdlib.Stack.Empty
```

Garanties en cas d'erreur

- pas de crash
- pas de valeurs créées ex-nihilo

État mutable interne

- synchronisation assurée par le module
  - Random
  - Dynlink
  - Str
  - Filesystem
  - Hashtbl
  - ...

Exemple

```
let rec sum n () =
  if n = 0 then 0
  else Random.int 50 + sum (n-1) ()
let ds = List.map (fun _ -> Domain.spawn (sum 100)) (List.init 50 (fun _ -> ()))
let r = List.fold_left (fun acc d -> acc + Domain.join d) 0 ds
```

Le futur

Quand OCaml 5.0.0 sera-t-elle disponible?

En fin d’année

OCaml 5.0.0

- OCaml 5.0.0 sera une version expérimentale
- OCaml 4.14 sera maintenu tant qu' OCaml 5 sera expérimentale
- Tester les décisions de conceptions

Restauration du support

- Port Risc-V en cours de relecture
- Port PowerPC et s390x, sur le moyen terme
- Quelle futur pour x86-32?

Système d’effet

- Fonctionnalité instable mais prometteuse
- Intégrer le système d’effet dans le langage
- Quel système de type?
- Quelle compatibilité ascendante?

Conclusion