Multicore OCaml

Stephen Dolan
Leo White
Anil Madhavapeddy

University of Cambridge

OCaml 2014
September 5, 2014
Concurrency and Parallelism

- Concurrency is for *writing programs*
  “my program handles 10,000 connections at once”

- Parallelism is for *performance*
  “my program makes use of 8 cores”
Concurrency in OCaml

• Good monadic concurrency libraries
  LWT and Async
  but monads are a bit awkward

• Direct-style threading library
  vmthreads and systhreads
  but neither is terribly efficient with many threads
Parallelism in OCaml

• Multiple processes with manual copying
  parmap, netmulticore, Async_parallel, ...

• Not much else.
  .... tumbleweed ...
Unify concurrency and parallelism?

• Concurrent programs are easily parallelised
  Should we use the same primitives?

• Java, C#, and others do
  But it's a bad idea :(

• At scale, death by context-switching
Fibers & Domains

• Fibers for concurrency
  Cheap to create, can have millions

• Domains for parallelism
  Expensive to create, have only a few
Concurrent Primitives

val spawn : (unit -> unit) -> unit

module MVar : sig
  type 'a t
  val create : unit -> 'a t
  val take : 'a t -> 'a
  val put : 'a t -> 'a -> unit
end
let async (f : unit -> 'a) : unit -> 'a =
  let m = MVar.create () in
  spawn (fun _ -> MVar.put m (f ()))
  fun _ -> MVar.take m
Parallelism primitives

• Multiple “domains” run in parallel
  Fibers are balanced between domains

• Creating domains is heavyweight
  Generally done once at startup and left alone

• Per-domain minor heap, shared major heap
  Differences not visible to OCaml code
Implementing the system

- OCaml is very fast for immutable data and functional programs
  
  We try to keep its behaviour unchanged

- Mutability is more complicated in a multithreaded system

- We use a descendant of the Doligez-Leroy collector from Concurrent Caml Light
Private minor heaps

- Each domain has a private minor heap
  - No pointers between minor heaps
Private minor heaps

- Each domain has a private minor heap
  - No pointers between minor heaps

Shared major heap

Domain 1 minor

Domain 2 minor
Barriers and faults

Upon access to a foreign heap, a fault occurs

Creates a shared copy of reachable objects
Collecting the minor heaps

- Completely independent minor collections
  Almost unmodified minor collector
Collecting the shared heap

- Mostly-concurrent parallel collector (VCGC)
  Requires synchronising domains once per cycle
Mostly-concurrent collection

• Domains race to mark reachable objects
  Some objects can be marked more than once

  ![Diagram showing UNMARKED and MARKED states]

• Domains sweep separate parts of the heap
  Sweeping need not precede marking

  ![Diagram showing GARBAGE and FREE states]
Shared collector cycles

- Domains verify marking and remap GC bits

Requires synchronising all domains, but there aren't many

Domains in `caml_enter_blocking_section` do not participate
C API

• Some minor changes (`sed` can fix most)
  
  `caml_modify(&Field(x,i), y)`
  becomes
  
  `caml_modify_field(x, i, y)`

• Same atomicity guarantees as current GC

• `caml_{enter,leave}_blocking_section`: does something different, has the same usage
Current Status

• Bootstraps, plays nicely with OPAM
  Branched off slightly before 4.02, no camlp4 right now

• GC needs testing, tuning and benchmarking
  GC policy is atrocious at the moment

• Bytecode only, and some features broken
  Weak references, finalisers and lazy values need work
Questions?

github.com/stedolan/ocaml
stephen.dolan@cl.cam.ac.uk