Санкт-Петербургский государственный университет
Кафедра системного программирования

Параллельная реализация miniKanren

Андрей Антонович Диденко, Б-07.21 ММ

Научный руководитель: Д.С.Косарев, ассистент кафедры системного программирования

Санкт-Петербург
2023

Введение

- Работа направлена на повышение эффективности вычислений на языке miniKanren путем декомпозиции задачи на потоках
- Данное решение предназначено для разработчиков, использующих miniKanren
- У работы нет аналогов
- За основу взят язык программирования OCaml, а также встаиваемый в него язык miniKanren

Постановка задачи

Целью работы является распараллеливание минимальной реализа- ции miniKanren для OCaml

Задачи:

- Выбрать версию OCaml, на которой будет реализована параллельность
- Внешне распараллелить две независимые задачи на miniKanren
- Встроить в Unicanren параллелизацию, которая работает не толь- ко внешне[1]

[1] `https://github.com/Kakadu/unicanren` (Дата доступа: 03.01.2023)

Используемые инструменты, подходы

- Выбрана самая свежая версия OCaml (5.0.0 beta2)[2] В этой версии добавили multicore в runtime и с помощью этого легко параллелить программы на OCaml
- Для распараллеливания используется библиотека DomainsLib[3], в которой присутствуют 2 модуля: Task – для вызова многопоточности и Chan – для передачи информации между доменами
- Вся проделанная работа выполнена на минимальной реализации miniKanren – Unicanren, которая состоит из 4 базовых конструкций: Fresh, Unify, Conde и Conj

[2] https://ocaml.org/news/ocaml-5.0.beta2(Дата доступа: 08.12.22)

[3] https://github.com/ocaml-multicore/parallel-programming-in-multicore-ocaml(Дата доступа: 08.12.22)

Процесс работы(1/4)

- Параллельный запуск append^o
- Параллельный запуск revers^o (при котором возникла проблема нехватки памяти)
- Параллельный запуск некоторых функций, которые возвращают несколько ответов
- Последней задачей оказалось слияние потоков
- Объеденить все задачи и положить в интерпретатор

Процесс работы (2/4)

Листинг 1: **Параллелизация двух revers^o на списках длины 700**

```
let goal = Call ("reverso", [ len; Var "xs" ]) in
let state = State.(empty |> "xs" ⟶ Var 10
|> add_rel "appendo" [ "xs"; "ys"; "xys" ] appendo_body
|> add_rel "reverso" [ "xy"; "yx" ] reverso_body)in
let wrap g = let s = StateMonad.run (eval g) state in s in
let pool = Task.setup_pool ~num_domains:12 () in
let a = Task.async pool (fun _ -> wrap goal) in
let b = Task.async pool (fun _ -> wrap goal) in
(match Task.run pool (fun () -> Task.await pool a,
Task.await pool b) with
 | Result.Ok a, Result.Ok b -> Stream.mplus a b
 | Ok _, Error _ | Error _, Ok _ | Error _, Error _ ->
   failwithf "%s %d" __FILE__ __LINE__)
|> Stream.take ~n:(-1)
```

Процесс работы (3/4)

Листинг 2: **Параллельная реализация Cond^e**

```
let c = Chan.make_unbounded () in
  let rec force_stream x = match x with
    | Stream.Cons (x, y) -> Chan.send c x;
      force_stream (Lazy.force y)
    | Stream.Nil -> ()
    | _ -> assert false in
  let* st = read in
  let pool = Task.setup_pool ~num_domains:12 () in
  let rec merge_stream n =
    match Chan.recv_poll c with
    | Some x -> let* () = put st in
    return (Stream.mplus (Stream.return x)) <*> merge_stream n
    | None -> return Stream.Nil
```

Процесс работы (4/4)

Листинг 3: **Параллельная реализация Cond^e**

```
    in
    let make_task acc =
    Task.async pool (fun _ ->
    force_stream (StateMonad.run (eval acc) st
    |> Result.get_ok))
    in
    let make_task_list lst =
    Stdlib.List.map make_task lst
    in
    Task.run pool (fun () ->
    Stdlib.List.iter (fun x -> Task.await pool x)
    (make_task_list lst));
    merge_stream c
```

Тестирование

Таблица: Замеры тестов для двух append^o

Длинна списка	Количество потоков	время, сек.
7000	2	26.7
7000	1	27.5
8000	12	24.2
8000	1	40.4

Вывод

После нескольких экспериментов сделан вывод, что запускать при малом количестве потоков, а также на малых обьемах данных не так выгодно, но при росте количества потоков пользы заметно больше.

Дальнейшие планы

- Протестировать на некотором количестве данных и убедиться, что работа действительно повышает эффективнось выполнения задач
- Доработать неисправности и сделать рефакторинг существуюшей реализации

Результаты

- Выбрана 5.0.0 - Second Beta версия OCaml для реализации
- Внешне распараллелены две независимые задачи на miniKanren
  - append^o
  - revers^o
- Встроить в Unicanren параллелизацию, которая работает не толь- ко внешне

Всю проделанную работу можно посмотреть на github репозитории[4]

[4] `https://github.com/K0lba/unicanren` (Дата доступа: 03.01.2023)