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
- Разработка ведется на функциональном языке программирования OCaml, на котором реализован miniKanren

Постановка задачи

Целью работы является распараллеливание минимальной реализации miniKanren для OCaml под названием Unicanren[1]

Задачи:

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

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

Используемые инструменты, подходы

- Выбрана версия OCaml (5.0.0 beta2)[2] В этой версии добавили поддержку многопоточности в среде выполнения и с помощью этого параллелить программы на OCaml легче, чем это было в предыдущих версиях
- 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 (1/3)

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

Процесс работы (2/3)

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

```
let a = Task.async pool (fun _ -> wrap goal) in
let b = Task.async pool (fun _ -> wrap goal) in
let res = Task.run pool (fun () -> Task.await pool a,
                                   Task.await pool b) in
(match res with
 | Result.Ok a, Result.Ok b -> Stream.mplus a b
 | Ok _, Error _
 | Error _, Ok _
 | Error _, Error _ ->
   fail "%s␣%d" __FILE__ __LINE__)
|> Stream.take ~n:(-1)
```

Процесс работы (3/3)

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 -> return (Stream.mplus (Stream.return x))
                <*> merge_stream n
    | None -> return Stream.Nil
```

Тестирование

Перебор всех возможных вариантов составления конкатенирующих списков

Таблица: Замеры тестов для append^o на списках разной длины и на разном количестве потоков

Суммарная длина	Количество потоков	Время, сек.
7000	2	26.5 - 26.8
7000	1	27.3 - 27.5
8000	12	24.2 - 24.4
8000	1	40.1 - 40.6

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
- В Unicanren встроена параллелизация, работающая не только внешне

Всю проделанную работу можно посмотреть на github репозитории[4]

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