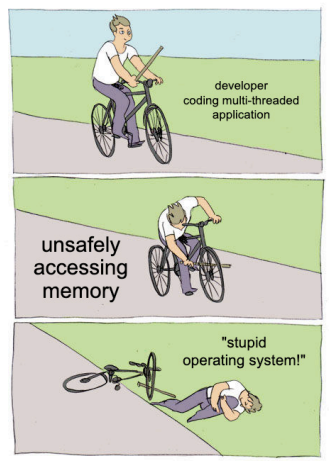
Thrust

Karolis Ryselis

Kauno Technologijos Universitetas

ktu
1922



Paskaitos turinys

Funkcinis programavimas

- Funkcinis programavimas — programavimo ideologija, kai viskas yra funkcija ir funkcijų pašaliniai poveikiai yra aiškiai atskirti nuo skaičiavimų logikos.
- Programa įsivaizduojama kaip funkcija, turinti parametrus ir pagal juos apskaičiuojanti reikšmę.
- Funkcinis programavimas yra **deklaratyvus** — programuotojas aprašo, **ką** nori padaryti, bet ne **kaip** padaryti.
- Vienas iš skiriamųjų funkcinio programavimo bruožų — visos duomenų struktūros yra nekeičiamos (*immutable*).

Pagrindinės gryno funkcinio programavimo kalbos

- Haskell
- Idris
- Elm

Klasikinės programavimo kalbos masiškai prideda funkcinio programavimo palaikymą.

Pagrindinės negryno funkcinio programavimo kalbos

- Lisp
- Clojure
- OCaml
- F#
- Scala
- Elixir

Darbas su sąrašais funkciniame programavime

- Su sąrašais galima atlikti tris pagrindines operacijas:
  - atvaizdis (*map*) — iš sąrašo gauti naują sąrašą, kur kiekvienam pradinio sąrašo elementui pritaikoma funkcija ir gaunamas naujo sąrašo elementas.
  - filtravimas (*filter*) — iš sąrašo gauti jo poaibį su elementais, tenkinančiais tam tikrą kriterijų.
  - sumažinimas (*reduce*) — iš sąrašo gauti skaliarą pritaikant tam tikrą funkciją.

map galima realizacija

```
function map(vector, func) {
    const result = [];
    for (let i = 0; i < vector.length; i++) {
        const item = func(vector[i]);
        result.push(item);
    }
    return result;
}
const testVector = [...Array(10).keys()];
console.log(testVector);

console.log("***map***");
console.log(testVector.map(x => x * x));
console.log(map(testVector, x => x * x));
```

map galima realizacija

Programos rezultatai:

```
[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
***map****
[ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 ]
[ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 ]
```

filter galima realizacija

```
function filter(vector, predicate) {
    const result = [];
    for (let i = 0; i < vector.length; i++) {
        if (predicate(vector[i])) {
            result.push(vector[i]);
        }
    }
    return result;
}

console.log("***filter****");
console.log(testVector.filter(x => x % 2 === 0));
console.log(filter(testVector, x => x % 2 === 0));
```

filter galima realizacija

Programos rezultatai:

```
***filter****
[ 0, 2, 4, 6, 8 ]
[ 0, 2, 4, 6, 8 ]
```

reduce galima realizacija

```
function reduce(vector, func) {
    let result = 0;
    for (let i = 0; i < vector.length; i++) {
        result = func(result, vector[i]);
    }
    return result;
}
console.log("***reduce****");
console.log(testVector.reduce((acc, x) => acc + x));
console.log(reduce(testVector, (acc, x) => acc + x));
```

reduce galima realizacija

Programos rezultatai:

```
***reduce****
45
45
```

C++ darbas su vektoriais

```
auto data_vector = get_vector();
vector<int> filtered_vector(data_vector.size());
// filter only even numbers
copy_if(data_vector.begin(), data_vector.end(),
    filtered_vector.begin(),
    [](auto item){ return item % 2 == 0;});
vector<int> squared_vector(filtered_vector.size());
// map to squares
transform(filtered_vector.begin(), filtered_vector.end(),
    squared_vector.begin(),
    [](auto item) { return item * item;});
// reduce to sum
auto sum = accumulate(squared_vector.begin(), squared_vector.end(),
    0, [](int accumulator, int operand){
      return accumulator + operand;
    });
cout << sum << endl;
```

Haskell darbas su sąrašais

```haskell
getVector :: [Integer]
getVector = take 20 [1,2..]

getSum :: Integer
getSum =
  vectorSum where
    vector = getVector
    -- filter only even numbers
    filteredVector = filter even vector
    -- map to squares
    squaredVector = map (\x -> x * x) filteredVector
    -- reduce to product
    vectorSum = foldl (+) 0 squaredVector
```

Python darbas su generatoriais

```
def get_vector():
    return range(1, 21)


vector = get_vector()
# filter only even numbers
even = (number for number in vector if number % 2 == 0)
# map to squares
squares = (number * number for number in even)
# reduce to sum
result = functools.reduce(lambda acc, x: acc + x, squares, 0)
print(result)
```

Java 8 darbas su srautais

```
private static IntStream getVector() {
    return IntStream.range(1, 21);
}

public static void main(String[] args) {
    IntStream vector = getVector();
    int sum = vector
            .filter(x -> x % 2 == 0) // filter only even
            .map(x -> x * x) // map to squares
            .reduce(0, (acc, x) -> acc + x); // reduce to sum
    System.out.println(sum);
}
```

JavaScript darbas su masyvais

```
function getVector() {
    return [...Array(20).keys()].map(i => i + 1);
}

const vector = getVector();
const sum = vector
    .filter(i => i % 2 === 0) // filter only even
    .map(i => i * i) // map to squares
    .reduce((acc, i) => acc + i, 0); // reduce to sum
console.log(sum)
```

Ruby darbas su masyvais

```ruby
def get_array
  1..21
end

array = get_array
vector_sum = array
             .select(&:even?) # filter only even numbers
             .collect { |x| x * x } # get squares
             .reduce(0) { |acc, num| acc + num } # reduce to sum
puts vector_sum
```

OCaml darbas su sąrašais

```
open Format

let numbers = List.init 20 succ;;
let even = List.filter (fun x -> x mod 2 == 0) numbers;;
let squares = List.map (fun x -> x * x) even;;
let total = List.fold_left ( + ) 0 squares;;
printf "%d\n" total;;
```

C# darbas su enumeratoriais

```
private static IEnumerable<int> GetVector() => Enumerable.Range(0, 20);

var vector = GetVector();
var sum = vector
    .Where(x => x % 2 == 0)
    .Select(x => x * x)
    .Aggregate((acc, x) => acc + x);
Console.WriteLine(sum);
```

D darbas su masyvais

```
int[] getVector()
{
    return iota(0, 20).array;
}

auto vector = getVector();
auto sum = vector
    .filter!(x => x % 2 == 0)
    .map!(x => x * x)
    .reduce!((acc, x) => acc + x);
writeln(sum);
```

`zip` operacija

- Kartais reikia iteruoti per du ar daugiau masyvus ar sąrašus kartu.
- Pvz., vykdant vektorių sudėtį, kai vektoriai yra vienodo ilgio sąrašai, reikia sudėti sąrašo elementus su vienodais indeksais.
- Tokiu atveju praverčia `zip` operacija, kuri leidžia iteruoti per du sąrašus iškart.

Python zip

```
import random


LIST_SIZE = 50


def get_random_list(list_size):
    return [random.randint(0, 100) for _ in range(list_size)]


if __name__ == '__main__':
    vector1 = get_random_list(LIST_SIZE)
    vector2 = get_random_list(LIST_SIZE)
    sum_vector = [x + y for x, y in zip(vector1, vector2)]
    print(sum_vector)
```

Haskell `zip`

```haskell
randomList :: Int -> IO([Int])
randomList 0 = return []
randomList n = do
  r  <- randomRIO (1,50)
  rs <- randomList (n-1)
  return (r:rs)

app :: IO ()
app = do
  vector1 <- randomList 50
  vector2 <- randomList 50
  print $ zipWith (+) vector1 vector2
```

Funktoriai

- Funktorius kategorijų teorijoje yra atvaizdis tarp kategorijų, t.y., taisyklės, kaip iš vienos kategorijos objekto gauti kitos kategorijos objektą.
- Programavime funktorius yra tipas, kuriuo galima atvaizduoti (*map*'inti).
- Klasikinėse programavimo kalbose visos ne `void` tipo funkcijos yra funktoriai.
- Skirtingose kalbose funktorių aibė gali būti neapribota funkcijomis.
- Funktoriai yra parametrai, kuriuos perduodame į `map`, `reduce`, `filter` ir kitas funkcijas — jie nusako, kas turi būti atliekama su elementais sąraše.

C++ funktoriai

- C++ funktoriaus vaidmenį dažniausiai atlieka funkcijos.
- Bendruoju atveju C++ funktorius yra klasė, kuri elgiasi kaip funkcija.
- Funktoriaus iškvietimo sintaksė yra tokia pat, kaip funkcijos.
- Kad klasė būtų funktorius, ji turi užkloti () operatorių.
- Struktūros taip pat gali turėti metodus, dėl to taip pat gali būti funktoriais.

Thrust

- Thrust — greitų lygiagrečiųjų algoritmų ir duomenų struktūrų biblioteka.
- Bibliotekos teikiama sąsaja panaši į C++ STL.

Thrust vektoriai

- Thrust palaiko vektorius CPU ir GPU atmintyje.
- CPU vektorių atitinka klasė `host_vector`.
- GPU vektorių atitinka klasė `device_vector`.
- Priskiriant `device_vector` objektą `host_vector` objektui arba atvirkščiai, kopijos konstruktorius pasirūpina duomenų perkėlimu tarp CPU ir GPU.

Thrust vektoriai

- Thrust vektoriams galioja tos pačios taisyklės, kaip dirbant su CUDA:
  - Su `host_vector` dirbti gali tik CPU funkcijos, su `device_vector` — tik GPU funkcijos.
  - Reikia duomenis sukelti į GPU, su jais vykdyti `__device__` arba `__global__` funkcijas ir rezultatus nusikopijuoti atgal.

Thrust funkcinio programavimo palaikymas

`copy_if(InputIterator first, InputIterator last, OutputIterator result, Predicate pred)`

Kopijuoja visus elementus nuo `first` iki `last`, kurie tenkina sąlygą `pred`, į rezultatų rinkinį `result` (*filter* operacija).

`transform(InputIterator first, InputIterator last, OutputIterator result, UnaryFunction op)`

Kiekvienam elementui nuo `first` iki `last` pritaiko funktorių `op` ir įrašo į rezultatų rinkinį `result` (*map* operacija).

Thrust funkcinio programavimo palaikymas

`reduce(InputIterator first, InputIterator last, T init, BinaryFunction binary_op)`

Kiekvienam elementui nuo `first` iki `last` vykdo operaciją `binary_op` kartu su iki tol sukaupta reikšme, pradedant nuo `init`, ir sukauptą reikšmę grąžina.

Visos funkcijos yra taikomos tiek `host_vector`, tiek `device_vector`, bet `device_vector` kviečiamos funkcijos bus vykdomos **GPU**, todėl turėtų būti *kernel* funkcijos.

Thrust funkcinio programavimo palaikymas

- Thrust turi rinkinį funktorių, kuriuos galima naudoti Thrust algoritmuose:
  - `plus`
  - `multiplies`
  - `greater`
  - …
- Visi Thrust funktoriai yra struktūros, todėl norint sukurti savo funktorių reikia kurti struktūras-funktorius.

Thrust funkcinis programavimas

```
struct is_even {
    __device__ bool operator ()(int item) {
        return item % 2 == 0;
    }
};

struct square {
    __device__ int operator ()(int item) {
        return item * item;
    }
};

struct sum_func {
    __device__ int operator ()(int accumulator, int item) {
        return accumulator + item;
    }
};
```

Thrust funkcinis programavimas

```
int main() {
    auto host_data_vector = get_vector();
    device_vector<int> data_vector = host_data_vector;
    device_vector<int> filtered_vector(data_vector.size());
    copy_if(data_vector.begin(), data_vector.end(),
        filtered_vector.begin(), is_even());
    device_vector<int> squared_vector(filtered_vector.size());
    transform(filtered_vector.begin(), filtered_vector.end(),
        squared_vector.begin(), square());
    int sum = reduce(squared_vector.begin(), squared_vector.end(),
        0, sum_func());
    cout << sum << endl;
    return 0;
}
```

Vektorių sudėtis Thrust

```
struct add_tuple_values {
    __device__ int operator ()(thrust::tuple<int, int> tuple) {
        return tuple.get<0>() + tuple.get<1>();
    }
};
```

Vektorių sudėtis Thrust

```
auto v1 = get_random_host_vector(VECTOR_SIZE);
auto v2 = get_random_host_vector(VECTOR_SIZE);
device_vector<int> dv1 = v1;
device_vector<int> dv2 = v2;
device_vector<int> dv_res(VECTOR_SIZE);
auto begin = make_zip_iterator(
    thrust::make_tuple(dv1.begin(), dv2.begin())
);
auto end = make_zip_iterator(
    thrust::make_tuple(dv1.end(), dv2.end())
);
thrust::transform(begin, end, dv_res.begin(),
    add_tuple_values());
host_vector<int> res = dv_res;
```

Vektorių sudėtis Thrust

Programos rezultatai:

```
host vector v1 contains:
87 52 73 13 77 88 82 95 8 97
host vector v2 contains:
3 30 87 11 43 40 45 41 78 2
host vector res contains:
90 82 160 24 120 128 127 136 86 99
```