Przetwarzanie danych masowych

Wykład 12 – Języki do przetwarzania danych masowych

dr inż. Tomasz Kajdanowicz, Roman Bartusiak, Piotr Bielak, Krzysztof Rajda

10 stycznia 2022 r.

Politechnika Wrocławska

Spis treści

- Python
- Rust
- Erlang
- Scala
- C++
- Go
- MPI

Materiały dodatkowe

- Frank Mueller, *How to Parallelize Your Code: Taking Stencils from OpenMP to MPI, CUDA and TensorFlow*
- David W. Walker, *Parallel Programming with OpenMP, MPI, and CUDA*
- Alfio Lazzaro, *Code Performance Optimizations*

Politechnika Wrocławska

Politechnika Wrocławska

Spis treści

- Python
- Rust
- Erlang
- Scala
- C++
- Go
- MPI

Wprowadzenie

Python

- język interpretowany
- silnie*, dynamicznie typowany
- wiele dostępnych bibliotek i paczek
- najpopularniejszy język w DS / ML
- przyspieszanie obliczeń dzięki FFI do języków C, C++

Silnie*, dynamicznie typowany

Python

Dynamicznie: Nie zmienne, ale obiekty są typowane!

```
a = 1  # type(a) => <class 'int'>
a = 'Hello'  # type(a) => <class 'str>
```

Silnie*: typy nie zmieniają się niespodziewanie

```
a = 1
b = '23'
print(a + b)
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# TypeError: unsupported operand type(s) for +: 'int
    ' and 'str'
```

Silnie*, dynamicznie typowany

Python

… ale z drugiej strony to działa (list zachowuje się jak bool; *implicit casting*)

```
a = [1, 2]
if a:
    print('Not empty!')
```

Bazowe typy danych (1)

Python

```python
# Tekst: str
a = 'Hello'

# Numeryczne: int, float, complex
a = 42
b = 777.0
c = 34.5 + 23.8j

# Sekwencje: list, tuple, range
a = [1, 2, 'Hello', True]
b = (1, 2, 'Hello', True)
c = range(1, 10, 2)
```

Bazowe typy danych (2)

Python

```python
# Mapowanie: dict
a = {'x': 2, 5: 'hello', True: 42}
# Klucze musza byc niemutowalne!

# Zbiory: set, frozenset
a = set([1, 1, 1, 4])  # a = {1, 4}

# Wartosci logiczne: bool
a = True
b = False

# Typy binarne: bytes, bytearray, memoryview
a = b'Hello'
```

Politechnika Wrocławska

Politechnika Wrocławska

Bardziej zaawansowane typy danych (1)

Python

```python
# Namedtuples
from collections import namedtuple

Person = namedtuple('Person', ['name', 'trademark'])
testo = Person(
    name='Lukasz Stanislawowski',
    trademark='Rolex'
)
testo.trademark = 'poramancza'
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# AttributeError: can't set attribute
```

Bardziej zaawansowane typy danych (2)

Python

```
# Wlasne klasy
# not: Person(object) - Python 2 syntax
class Person:
    def __init__(self, name, trademark):
        self._name = name
        self._trademark = trademark

testo = Person(
    name='Lukasz Stanislawowski',
    trademark='Rolex'
)
testo._trademark = 'pomarancza'
# Perfectly fine for interpreter, but avoid that,
    please...
```

Bardziej zaawansowane typy danych (3)

Python

```python
# Enums
from enum import Enum

class Trademarks(Enum):
    POMARANCZA = 0
    ROLEX = 1

print(Trademark.POMARANCZA)
# Trademark.POMARANCZA

print(repr(Trademark.POMARANCZA))
# <Trademark.POMARANCZA: 0>
```

Python manifest

Python

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Nowe funkcjonalności w Pythonie

- jednym z głównych aspektów pisania kodu jest jego czytelność
- warto sprawdzać zmiany wprowadzane w najnowszych wersjach języka
- często nowe funkcjonalności pozwalają pisać czytelniejsze i łatwiejszy w utrzymywaniu kod
- obecnie: Python 3.10
- zobaczmy kilka z tych funkcjonalności ...

Type hints (1)

Python

Nie to samo co statyczne typowanie! Tutaj: podpowiedzi dla linterów

```python
a: int = 7
a: int = 'Hi'  # Works, but good IDE will complain

a: str = 'Hello'
a: bool = True
a: dict = {'x': 1, 'y': 0}
a: MyClass = MyClass(x=0, y=42)

```

Politechnika Wrocławska

Type hints (2)

Python

Type hints dla mapowań, zbiorów, list itd. można zapisywać na dwa sposoby: (A) "Dict[]", "Set[]", "List[]", albo (B) "dict[]", "set[]", "list[]" (od Pythona 3.9).

```
from typing import Dict, List, Set

a: List[int] = [1, 2, 3]   # lub `a: list[int]`
a: Set[str] = {'Hello', 'Hi'}  # lub `a: set[str]`
a: Dict[str, int] = {'x': 0, 'y': 1}  # lub `a: dict[str, int]`

```

Type hints (3)

Python

Type hints można stosować także dla metod i funkcji:

```
from typing import List

def contains(x: List[int], val: int) -> bool:
    return val in x

class Person:
    def __init__(self, name: str, trademark:
    Trademark) -> None:
        self._name = name
        self._trademark = trademark

```

Type hints (4)

Python

Warto wspomnieć: Data classes

```
from dataclass import dataclass, field
from typing import List

@dataclass
class Person:
    name: str
    age: int
    trademark: Trademark
    videos: List[str] = field(
        init=False,
        repr=False,
        default_factory=list
    )  # NOT: videos: List[str] = []
```

Type hints (5)

Python

Warto wspomnieć:

```python
from typing import Optional, Sequence, Tuple, Union

# Albo str albo int
# (od Pythona 3.10: `x: str | int`)
def foo(x: Union[str, int]) -> None: ...

# 3-krotka of str, str and int
def foo(x: Tuple[str, str, int]) -> None: ...

# Jakikolwiek iterowalny typ int'ow
def foo(x: Sequence[int]) -> None: ...

# Wartosc opcjonalna (!= wartosc domyslna)
# Tutaj: str albo None
def foo(x: Optional[str] = None) -> None: ...

# vs wartosc domyslna
def foo(x: str = 'Hi') -> None: ...
```

Politechnika Wrocławska

Type hints (6)

Python

Warto wspomnieć:

```
from typing import Callable, List, TypeVar

T = TypeVar('T')

def my_map(
    vals: List[T],
    fn: Callable[[T,], T]
) -> List[T]:
    return [fn(x) for x in vals]

def double(x: int) -> int:
    return x * 2

def custom_len(x: str) -> int:
    return len(x)


my_map(vals=[1, 2], fn=double)  # OK
my_map(vals=[1, 'Hi'], fn=double)  # WRONG, why?
my_map(vals=['A', 'B'], fn=double)  # WRONG, why?
my_map(vals=['A', 'B'], fn=custom_len)  # OK
my_map(vals=[1, 2], fn=custom_len)  # WRONG, why?
```

Politechnika Wrocławska

Other features

Python

- f-strings,
- breakpoint(),
- positional only arguments,
- literal types,
- typed dicts,
- final objects,
- **structural pattern matching**
- ...
- `realpython.com/python310-new-features/`

Politechnika Wrocławska

Politechnika Wrocławska

Jak zrównoleglać obliczenia?

Python

- natywnie:
  - wątki,
  - procesy,
- zew. biblioteki:
  - celery,
  - pyfunctional,
  - ray,
  - dask,
  - ...

Natywne zrównoleglanie (1)

Python

Processes:

```python
# ...
import multiprocessing as mp

def make_work(x: int, y: List[int]) -> int: ...

def worker_fn(args: tuple) -> int:
    return make_work(*args)

def run():
    args: List[Tuple[int, List[int]]] = [
        (1, [2, 3, 4]),
        ...
    ]

    with mp.Pool(processes=mp.cpu_count()) as pool:
        results = pool.map(worker_fn, args)

```

Natywne zrównoleglanie (2)

Python

Threads:

```
# ...
import multiprocessing as mp
from multiprocessing.pool import ThreadPool

# same code as previously

def run():
    # same code as previously

    with Threadpool(processes=mp.cpu_count()) as
    pool:
        results = pool.map(worker_fn, args)

```

Zrównoleglanie zew. bibliotekami (1)

Python

Celery:

- rozproszona kolejka zadań,
- używa kolejki RabbitMQ,
- znana wszystkim studentom – używane na laboratoriach

```
from celery import Celery

app = Celery('myapp', broker='amqp://')

@app.task
def add(x, y):
    return x + y


if __name__ == '__main__':
    app.start()
```

Politechnika Wrocławska

Zrównoleglanie zew. bibliotekami (2)

Python

PyFunctional:

- funkcyjne API do streamów (collections),
- tryb sekwencyjny i zrównoleglony,
- zrównoleglenie tylko na jednej maszynie,
- bezpośredni odczyt/zapis z/do baz danych, plików CSV, ...
- leniwa ewaluacja,
- działanie zrównoleglone: `pseq`
- `github.com/EntilZha/PyFunctional`

```
from functional import seq

(seq(1, 2, 3, 4)
    .map(lambda x: x * 2)
    .filter(lambda x: x > 4)
    .reduce(lambda x, y: x + y))
```

Zrównoleglanie zew. bibliotekami (3)

Python

Ray:

- framework do przetwarzania rozproszonego,
- używa Redis,
- wdrożenie na AWS, GCE, K8s

```python
import ray
ray.init()

@ray.remote
def f(x):
    return x * x

futures = [f.remote(i) for i in range(4)]
print(ray.get(futures))
```

Zrównoleglanie zew. bibliotekami (4)

Python

Dask:

- framework do przetwarzania rozproszonego,
- zintegrowany z Numpy, Pandas, Scikit-learn, XGBoost,
- wdrożenie na K8s, Hadoop/YARN, SSH,

```python
# Arrays implement the Numpy API
import dask.array as da
x = da.random.random(size=(...), chunks=(...))
x + x.T - x.mean(axis=0)

# Dataframes implement the Pandas API
import dask.dataframe as dd
df = dd.read_csv('s3://file.csv')
df.groupby(df.account_id).balance.sum()

# Dask-ML implements the Scikit-Learn API
from dask_ml.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train, test)
```

Spis treści

Python

Rust

Erlang

Scala

C++

Go

MPI

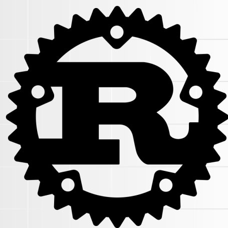
Politechnika Wrocławska

Politechnika Wrocławska

Wprowadzenie (1)

Rust

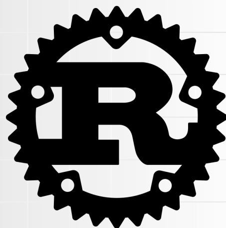
- silnie rozwijany język
- utworzony przez Mozilla
- używany przez Microsoft (2019)
- język kompilowany
- szybki (LLVM backend – tak jak C++)
- bardzo dobry toolkit (cargo)
- kompilator **gwarantuje bezpieczeństwo pamięciowe** (*memory safety*)
- borrow checker (brak GC)



Wprowadzenie (2)

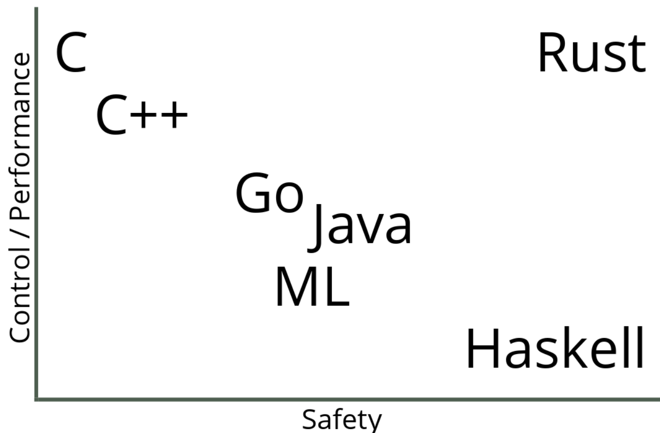
Rust

- **statyczne typowanie**,
- systems programming language,
- dostęp do niskopoziomowych zasobów systemu operacyjnego – tak jak w C (pipes, sockets, message queues),
- proste FFI,
- nadaje się do systemów wbudowanych,
- WASM – kompilacja z JavaScript,
- kompilacja do natywnych GPU kernels*,
- **wysoki próg wejścia** (stroma krzywa uczenia),



Porównanie Rusta do innych języków

Rust



Politechnika Wrocławska

Cargo (1)

Rust

- zarządzanie zależnościami,
- code linter (`cargo clippy`),
- dokumentacja (`cargo doc`),
- formatowanie kodu (`cargo fmt`),
- uruchamianie testów (`cargo test`),
- aktualizacja kompilatora i toolkita (`rustup`)

Cargo (2)

Rust

```
[package]
name = "docker-cmd"
version = "0.2.0"
authors = ["pbielak"]
edition = "2018"

[dependencies]
console = "0.7.5"
dialoguer = "0.3.0"
derive_more = "0.13.0"
nix = "0.13.0"
structopt = "0.2"
tabwriter = "1.1.0"
```

Politechnika Wrocławska

Borrow checker

Rust

- no dangling pointers,
- no double free (security risk!),
- no data races (easy concurrency),
- jeśli się skompiluje to kod jest poprawny (w 99% wszystkich przypadków),

Podstawy języka (1)

Rust

```rust
// Main function definition
fn main() {
    println!("Hello, world!");
}

// Variables
let a = true;
let b: bool = true;

a = false; // Error!

// Mutable variables
let mut a = true;
a = false; // OK!

// Variable bindings
let (x, y) = (1, 2);
```

Podstawy języka (2)

Rust

```rust
// Function definition
// No return statement needed!
// (If it is the last one*)
fn double(x: i32) -> i32 {
    2 * x
}

// Functions can be assigned to variables
let f = double;
println!("2 * {} = {}", 1, f(1));

// If statements assignment
let is_below_eighteen = if age < 18 { true } else {
    false };
```

Podstawy języka (3)

Rust

```rust
// Pattern matching
let f = File::open("hello.txt");

let f = match f {
    Ok(file) => file,
    Err(error) => panic!("Oh noes: {:?}", error),
};
```

Podstawy języka (4)

Rust

```rust
// No classes - structs + traits
#[derive(Debug)]
struct Person<'a> {
    name: &'a str,
    age: u8,
}

let p = Person {
    name: "Lukasz Stanislawowski",
    age: 32
};
```

Podstawy języka (5)

Rust

```rust
// No classes - structs + traits
#[derive(Debug)]
struct Person<'a> {
    name: &'a str,
    age: u8,
}

impl<'a> Person<'a> {
    fn new(name: &'a str, age: u8) -> Person<'a> {
        Person { name, age }
    }

    fn is_online(&self) -> bool {
        false
    }
}

let p = Person::new("Lukasz Stanislawowski", 32);
```

Podstawy języka (6)

Rust

```rust
// ... same code ...

trait OnlineChecker {
    fn is_online(&self) -> bool;
}

impl<'a> Person<'a> {
    fn new(name: &'a str, age: u8) -> Person<'a> {
        Person { name, age }
    }
}

impl<'a> OnlineChecker for Person<'a> {
    fn is_online(&self) -> bool { false }
}

impl OnlineChecker for i32 {
    fn is_online(&self) -> bool { true }
}
```


Politechnika Wrocławska

Zasady borrow checker'a

Rust

"First, any borrow must last for a scope no greater than that of the owner. Second, you may have one or the other of these two kinds of borrows, but not both at the same time:

- one or more references (&T) to a resource,
- exactly one mutable reference (&mut T)."

Źródło:
doc.rust-lang.org/1.8.0/book/references-and-borrowing.html

Borrow checker – przykład (1.1)

Rust

```rust
let mut v = vec![1, 2, 3];

for i in &v {
    println!("{}", i);
    v.push(34);
}
```

Borrow checker – przykład (1.2)

Rust

```
1  // error: cannot borrow `v` as mutable because it is
      also borrowed as immutable
2  //     v.push(34);
3  //     ^
4  // note: previous borrow of `v` occurs here; the
      immutable borrow prevents
5  // subsequent moves or mutable borrows of `v` until
      the borrow ends
6  // for i in &v {
7  //           ^
8  // note: previous borrow ends here
9  // for i in &v {
10 //     println!("{}", i);
11 //     v.push(34);
12 // }
13 // ^
```

Borrow checker – przykład (2.1)

Rust

```
let mut x = 5;
let y = &mut x;

*y += 1;

println!("{}", x);
```

Borrow checker – przykład (2.2)

Rust

```
// error: cannot borrow `x` as immutable because it
    is also borrowed as mutable
//     println!("{}", x);
//                    ^


let mut x = 5;

let y = &mut x;    // -+ &mut borrow of x starts
    here
                   //  |
*y += 1;           //  |
                   //  |
println!("{}", x); // -+ - try to borrow x here
                   // -+ &mut borrow of x ends here
```

Politechnika Wrocławska

Jak zrównoleglać?

Rust

- głównie: wątki + kanały (*channels*),
- zew. biblioteki: actix, rayon, tokio
- async*,

Wątki (1)

Rust

```
use std::thread;

fn main() {
    thread::spawn(|| {
        println!("Hello from a thread!");
    });
}
```

Wątki (2)

Rust

```rust
use std::thread;

fn main() {
    let x = 1;
    thread::spawn(move || {
        println!("x is {}", x);
    });
}
```

Wątki (3.1)

Rust

```
// NOT WORKING!
use std::thread;
use std::time::Duration;

fn main() {
    let mut data = vec![1, 2, 3];

    for i in 0..3 {
        thread::spawn(move || {
            data[0] += i;
        });
    }

    thread::sleep(Duration::from_millis(50));
}
```

Wątki (3.2)

Rust

```rust
// OK!
use std::sync::{Arc, Mutex};
use std::thread;
use std::time::Duration;

fn main() {
    let data = Arc::new(Mutex::new(vec![1, 2, 3]));

    for i in 0..3 {
        let data = data.clone();
        thread::spawn(move || {
            let mut data = data.lock().unwrap();
            data[0] += i;
        });
    }

    thread::sleep(Duration::from_millis(50));
}
```

Wątki (4)

Rust

```rust
use std::thread;
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();

    for i in 0..10 {
        let tx = tx.clone();

        thread::spawn(move || {
            let answer = i * i;

            tx.send(answer).unwrap();
        });
    }

    for _ in 0..10 {
        println!("{}", rx.recv().unwrap());
    }
}
```

Politechnika Wrocławska

Spis treści

Python

Rust

Erlang

Scala

C++

Go

MPI

Politechnika Wrocławska

Wprowadzenie

Erlang

- współbieżny,
- funkcyjny język programowania,
- garbage collection,
- **programowanie aktorowe**,
- popularne oprogramowanie: RabbitMQ, WhatsApp

Politechnika Wrocławska

Erlang – środowisko uruchomieniowe (*runtime*)

- rozproszony,
- odporny (*fault tolerant*),
- soft real-time
- HA, non-stop applications
- Hot swapping (podmiana kodu bez zatrzymywania systemu)

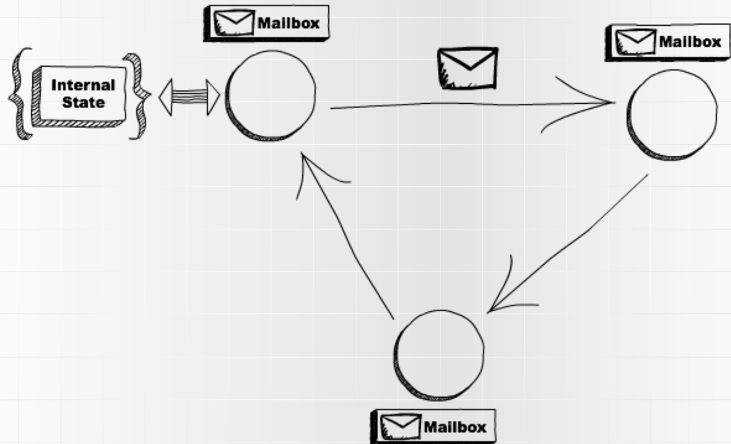
Programowanie aktorowe (1)

Erlang

- programowanie aktorowe = prawdziwa obiektowość (*OOP*),
- aktor = podstawowa jednostka obliczeniowa,
- aktor otrzymuje wiadomości i dokonuje pewnych obliczeń na ich podstawie,
- aktorzy są w pełni odseparowani od siebie (brak pamięci współdzielonej, itp.),

Programowanie aktorowe (2)

Erlang



Politechnika Wrocławska

Programowanie aktorowe (3)

Erlang

W momencie otrzymania wiadomości, aktor może wykonać z jedną z następujących 3 akcji:

- utworzyć więcej aktorów,
- wysłać wiadomości do innych aktorów,
- ustalić co zrobić z następną wiadomością.

Politechnika Wrocławska

Programowanie aktorowe (4)

Erlang

- programista nie powinien się przejmować odpornością na awarie,
- nadzorca (*supervisor*), które restartuje aktorów, jeśli nastąpi awaria,
- rozproszenie jest proste (ograniczona do serializacji wiadomości)

Spis treści

- Python
- Rust
- Erlang
- Scala
- C++
- Go
- MPI

Politechnika Wrocławska

Politechnika Wrocławska

Wprowadzenie (1)

Scala

- JVM
- język kompilowany
- statycznie typowany
- można używać wraz z całym ekosystemem JVM
- łączy wiele paradygmatów

Politechnika Wrocławska

Wprowadzenie (2)

Scala

- domyślna niemutowalność
- val vs var
- immutable case classes
- leniwość
- higher order functions

Val vs var

Scala

```scala
var a = 0
a = a + 1
val b = 0
b = b + 1 // NOPE!
```

Case class (1)

Scala

```scala
case class Person(name: String)
val person = Person("Lukasz Stanislawowski")
person.name="Testo" //NOPE!
```


Politechnika Wrocławska

Case class (2)

Scala

```scala
case class Person(name: String)
var person = Person("Lukasz Stanislawowski")
person.name="Testo" //NOPE!
```

Case class (3)

Scala

```
case class Person(name: String)
var person = Person("Lukasz Stanislawowski")
var newPerson = person.copy(name = "Testo") //WEEEE!
```

Definicje funkcji

Scala

```
1 def someDefinition(a:String): String = a
2
3 val someLambda = (a: String) => a
4
5 def higherOrderFunction(f: String=>String): String=>
    String = f
```

String substitution

Scala

```
val a = 10
val b = 11

val c: String = s"$a + $b = ${a+b}"
```

Politechnika Wrocławska

Typy generyczne

Scala

```scala
case class Container[T](data: T)
Container("string")
Container(1)
Container(Container("wow"))
//Container[Container[String]]
```

Wartości implicit (1)

Scala

```scala
case class Context(data: String)
val context = Context("data")

def functionThatRequiresContext(data: String, ctx: Context): String = data

functionThatRequiresContext("notNice", context)
```


Politechnika Wrocławska

Wartości implicit (2)

Scala

```scala
case class Context(data: String)
implicit val context = Context("data")

def functionThatRequiresContext(data: String)(
    implicit ctx: Context): String = data

functionThatRequiresContext("nice!!!")
```

Type class (1)

Scala

```scala
val s: String = "fun"

s.makeFun //NOPE!
```

Type class (2)

Scala

```
implicit class Funner(s: String){
    def makeFun: String = "funHasBeenMade"
}
val s: String = "fun"

s.makeFun //WOOOOOW!
```

Type class (3)

Scala

```
object Show {
    trait Show[A] {
      def show(a: A): String
    }

    def show[A](a: A)(implicit sh: Show[A]) = sh.
    show(a)

    implicit val intCanShow: Show[Int] =
        new Show[Int] {
          def show(int: Int): String = s"int $int"
        }
}
```

Type class (4)

Scala

```scala
import Show._
print(show(1))
```

Type class (5)

Scala

```scala
import Show._
print(show(1)(intCanShow))
```

Type class (6)

Scala

```
import Show._
case class Person(name: String)

implicit val personCanBeShown: Show[Person] =
    new Show[Int] {
        def show(p: Person): String = p.name
    }

print(show(Person("Lukasz S.")))
```


Politechnika Wrocławska

Politechnika Wrocławska

Parallel collections (1)

Scala

- odzielna biblioteka od wersji 2.13
- lepiej używać klasy Vector
- nie zawsze warto zrównoleglać!

Parallel collections (2)

Scala

```
1 import scala.collection.parallel.
      CollectionConverters._
2 val lastNames = List("Smith","Jones","Frankenstein",
      "Bach","Jackson","Rodin").par
3 lastNames.map(_.toUpperCase)
```


Politechnika Wrocławska

Parallel collections (3)

Scala

```scala
import scala.collection.parallel.
    CollectionConverters._
val parArray = (1 to 10000).toArray.par
parArray.fold(0)(_ + _)
```

Parallel collections (4)

Scala

```scala
import scala.collection.parallel.
    CollectionConverters._
val lastNames = List("Smith","Jones","Frankenstein",
    "Bach","Jackson","Rodin").par
lastNames.filter(_.head >= 'J')
```

Wielowątkowość (1)

Scala

- Runnable
- Futures
- Promises
- ExecutionContext
- zmiana kontekstu zajmuje czas!

Wielowątkowość (2)

Scala

- Fork join pool – asynchroniczność
- Thread pool – operacje blokujące

Wielowątkowość (3)

Scala

```scala
object RunnableUsingGlobalExecutionContext extends App {
  val ctx = scala.concurrent.ExecutionContext.global
  ctx.execute(new Runnable {
    def run() = print("Hey, I am on a separate thread!.")
  })
  Thread.sleep(1000)
}
```

Wielowątkowość (4)

Scala

```
import scala.concurrent.{ Future, ExecutionContext}
object FutureeUsingGlobalExecutionContext extends
    App {
  implicit val ctx = ExecutionContext.global
  Future {
    print("Hey, I am on a separate thread!.")
  }
  Thread.sleep(1000)
}
```

Politechnika Wrocławska

Strumienie

Scala

- failure
- map
- flatMap
- sequential

Akka

Scala

- Akka
- Play
- Akka-Stream
- Akka-Typed

Politechnika Wrocławska

Spis treści

- Python
- Rust
- Erlang
- Scala
- C++
- Go
- MPI

Wprowadzenie

C++

- wysoka wydajność
- optymalizacje
- nisko-poziomowy
- pełna kontrola

Optymalizacje (1)

C++

```
Optimization Options
    -faggressive-loop-optimizations -falign-functions[=n[:m:[n2[:m2]]]] -falign-jumps[=n[:m:[n2[:m2]]]] -falign-labels[=n[:m:[n2[:m2]]]] -falign-loops[=n[:m:[n2[:m2]]]] -fassociative-math
    -fauto-profile  -fauto-profile[=path] -fauto-inc-dec  -fbranch-probabilities -fbranch-target-load-optimize  -fbranch-target-load-optimize2 -fbtr-bb-exclusive  -fcaller-saves
    -fcombine-stack-adjustments  -fconserve-stack -fcompare-elim  -fcprop-registers  -fcrossjumping -fcse-follow-jumps  -fcse-skip-blocks  -fcx-fortran-rules -fcx-limited-range -fdata-sections  -fdce
    -fdelayed-branch -fdelete-null-pointer-checks  -fdevirtualize  -fdevirtualize-speculatively -fdevirtualize-at-ltrans  -fdse -fearly-inlining  -fipa-sra  -fexpensive-optimizations
    -ffat-lto-objects -ffast-math  -ffinite-math-only  -ffloat-store  -fexcess-precision=style -fforward-propagate  -ffp-contract=style  -ffunction-sections -fgcse  -fgcse-after-reload  -fgcse-las
    -fgcse-lm  -fgraphite-identity -fgcse-sm  -fhoist-adjacent-loads  -fif-conversion -fif-conversion2  -findirect-inlining -finline-functions  -finline-functions-called-once  -finline-limit=n
    -finline-small-functions  -fipa-cp  -fipa-cp-clone -fipa-bit-cp  -fipa-vrp  -fipa-pta  -fipa-profile  -fipa-pure-const -fipa-reference  -fipa-reference-addressable -fipa-stack-alignment
    -fipa-icf  -fira-algorithm=algorithm -flive-patching=level -fira-region=region  -fira-hoist-pressure -fira-loop-pressure  -fno-ira-share-save-slots -fno-ira-share-spill-slots
    -fisolate-erroneous-paths-dereference  -fisolate-erroneous-paths-attribute -fivopts  -fkeep-inline-functions  -fkeep-static-functions -fkeep-static-consts  -flimit-function-alignment
    -flive-range-shrinkage -floop-block  -floop-interchange  -floop-strip-mine -floop-unroll-and-jam  -floop-nest-optimize -floop-parallelize-all  -flra-remat  -flto  -flto-compression-level
    -flto-partition=alg  -fmerge-all-constants -fmerge-constants  -fmodulo-sched  -fmodulo-sched-allow-regmoves -fmove-loop-invariants  -fno-branch-count-reg -fno-defer-pop
    -fno-fp-int-builtin-inexact  -fno-function-cse -fno-guess-branch-probability  -fno-inline  -fno-math-errno  -fno-peephole -fno-peephole2  -fno-printf-return-value  -fno-sched-interblock
    -fno-sched-spec  -fno-signed-zeros -fno-toplevel-reorder  -fno-trapping-math  -fno-zero-initialized-in-bss -fomit-frame-pointer  -foptimize-sibling-calls -fpartial-inlining  -fpeel-loops
    -fpredictive-commoning -fprefetch-loop-arrays -fprofile-correction -fprofile-use  -fprofile-use=path  -fprofile-values -fprofile-reorder-functions -freciprocal-math  -free  -frename-registers
    -freorder-blocks -freorder-blocks-algorithm=algorithm -freorder-blocks-and-partition  -freorder-functions -frerun-cse-after-loop  -freschedule-modulo-scheduled-loops -frounding-math
    -fsave-optimization-record -fsched2-use-superblocks  -fsched-pressure -fsched-spec-load  -fsched-spec-load-dangerous -fsched-stalled-insns-dep[=n]  -fsched-stalled-insns[=n]
    -fsched-group-heuristic  -fsched-critical-path-heuristic -fsched-spec-insn-heuristic  -fsched-rank-heuristic -fsched-last-insn-heuristic  -fsched-dep-count-heuristic -fschedule-fusion
    -fschedule-insns  -fschedule-insns2  -fsection-anchors -fselective-scheduling  -fselective-scheduling2 -fsel-sched-pipelining  -fsel-sched-pipelining-outer-loops -fsemantic-interposition
    -fshrink-wrap  -fshrink-wrap-separate -fsignaling-nans -fsingle-precision-constant  -fsplit-ivs-in-unroller  -fsplit-loops -fsplit-paths -fsplit-wide-types  -fssa-backprop  -fssa-phiopt
    -fstdarg-opt  -fstore-merging  -fstrict-aliasing -fthread-jumps  -ftracer  -ftree-bit-ccp -ftree-builtin-call-dce  -ftree-ccp  -ftree-ch -ftree-coalesce-vars  -ftree-copy-prop  -ftree-dce
    -ftree-dominator-opts -ftree-dse  -ftree-forwprop  -ftree-fre  -fcode-hoisting -ftree-loop-if-convert  -ftree-loop-im -ftree-phiprop  -ftree-loop-distribution  -ftree-loop-distribute-patterns
    -ftree-loop-ivcanon  -ftree-loop-linear  -ftree-loop-optimize -ftree-loop-vectorize -ftree-parallelize-loops=n  -ftree-pre  -ftree-partial-pre  -ftree-pta -ftree-reassoc  -ftree-scev-cprop
    -ftree-sink  -ftree-slsr  -ftree-sra -ftree-switch-conversion  -ftree-tail-merge -ftree-ter  -ftree-vectorize  -ftree-vrp  -funconstrained-commons -funit-at-a-time  -funroll-all-loops
    -funroll-loops -funsafe-math-optimizations  -funswitch-loops -fipa-ra  -fvariable-expansion-in-unroller  -fvect-cost-model  -fvpt -fweb  -fwhole-program  -fwpa  -fuse-linker-plugin --param
    name=value -O  -O0  -O1  -O2  -O3  -Os  -Ofast  -Og
```

Optymalizacje (2)

C++

Ekstrakcja wspólnych fragmentów kodu:

```
a = b * c + d;
e = b * c + 3;
```

```
tmp = b * c
a = tmp + d;
e = tmp + 3;
```


Politechnika Wrocławska

Optymalizacje (3)

C++

Usuwanie nieużywanego kodu:

```
1 int main(){
2     int v[2];
3     for (int i = 0; i<2; i++)
4         v[i] = i*i
5     return 0;
6 }
```

Politechnika Wrocławska

Optymalizacje (4)

C++

Automatyczna wektoryzacja

- instrukcje SIMD używane do przyspieszania obliczeń w pętli
- zależna od dostępnych instrukcji SIMD
- 2x dla operacji Single Precision
- może wprowadzać różne tryby zaokrąglania liczb

Optymalizacje (5)

C++

Przeniesienie kodu niezmienniczego względem pętli:

```
for (int i=0; i<n; i++) {
    x = y * z;
    a[i] = 2 * i + x * x;
}
```

```
x = y * z;
tmp = x * x;
for (int i=0; i<n; i++) {
    a[i] = 2 * i + tmp;
}
```

Optymalizacje (6)

C++

Upraszczanie przez dodawanie nowych zmiennych:

```
x = y * z;
tmp = x * x;
for (int i=0; i<n; i++) {
    a[i] = 2 * i + tmp;
}
```

```
x = y * z;
tmp = x * x;
for (int i=0; i<n; i++, tmp+=2) {
    a[i] = tmp;
}
```

Optymalizacje (7.1)

C++

Rozwijanie pętli:

```
for (int i=0; i<n; i++) {
    a[i] += 2.2 * b[i];
}
```

```
for (int i=0; i<n; i+=4) {
    a[i] += 2.2 * b[i];
    a[i+1] += 2.2 * b[i+1];
    a[i+2] += 2.2 * b[i+2];
    a[i+3] += 2.2 * b[i+3];
}
```

Politechnika Wrocławska

Optymalizacje (7.2)

C++

Rozwijanie pętli:

- lepszy pipelining w CPU
- lepsza wektoryzacja
- zwiększenie rozmiaru pliku wyjściowego

Politechnika Wrocławska

Optymalizacje (8)

C++

Function inlining:

- wyniki wywołań funkcji są odkładane na stosie
- wywołania funkcji zatrzymują dalsze optymalizacje (np. wektoryzację)
- zwiększenie rozmiaru pliku wyjściowego

Ręczne optymalizacje (1)

Zamiana pętli:

Źle:

```cpp
for (int j=0; i<columns; j++) {
    for (int i=0; i<rows; i++) {
        mymatrix[i][j] += increment;
    }
}
```

Poprawnie:

```cpp
for (int i=0; i<rows; i++) {
    for (int j=0; i<columns; j++) {
        mymatrix[i][j] += increment;
    }
}
```

Ręczne optymalizacje (2)

C++

Łączenie pętli:

```
for (int j=0; i<columns; j++) {
    for (int i=0; i<rows; i++) {
        mymatrix[i][j] = othermatrix[i][j]*2;
    }
}
for (int j=0; i<columns; j++) {
    for (int i=0; i<rows; i++) {
        mymatrix[i][j] += 1;
    }
}
```

```
for (int j=0; i<columns; j++) {
    for (int i=0; i<rows; i++) {
        mymatrix[i][j] = othermatrix[i][j]*2;
        mymatrix[i][j] += 1;
    }
}
```

OpenMP (1)

C++

- wielowątkowość
- explicit
- API albo `pragma`
- zrównoleglanie ze względu na zadanie albo dane
- SIMD

Politechnika Wrocławska

OpenMP (2)

C++

```
#pragma omp parallel
#pragma omp for
for (int i=0; i<10; i++) {
    // do something with i
}
```

OpenMP (3)

C++

```
1 #pragma omp parallel num_threads(3)
2 #pragma omp for
3 for (int i=0; i<10; i++) {
4     // do something with i
5 }
```

OpenMP (4)

C++

```
#pragma omp parallel
int id = omp_get_thread_num();
int total = omp_get_num_threads();
#pragma omp for
for (int i=0; i<10; i++) {
    // do something with i
}
```

Politechnika Wrocławska

Spis treści

- Python
- Rust
- Erlang
- Scala
- C++
- Go
- MPI

Wprowadzenie

Go

- język kompilowany
- statycznie typowany
- utworzony przez Google
- wysoka wydajność
- prosty rozwój aplikacji
- prosta składnia

Politechnika Wrocławska

Protoactor

Go

- bardzo szybki
- protobuffers
- wirtualni aktorzy
- do 10x przyspieszenia względem Erlang’a
- do 100x przyspieszenia względem Akka.NET
- Kotlin, C#, Go

Politechnika Wrocławska

Spis treści

- Python
- Rust
- Erlang
- Scala
- C++
- Go
- MPI

Wprowadzenie

MPI

- szeroko używany standard przekazywania wiadomości między węzłami z rozproszoną pamięcią
- nadawca i odbiorca muszą zdefiniować typ danych
- komunikacja point-to-point
- komunikacja kolektywna (*collective*)
- definiuje podstawowe typy danych

Politechnika Wrocławska

Komunikacja

MPI

- zazwyczaj komunikacja all-to-all nie jest używana
- MPI określa topologię aplikacji
- wsparcie dla Cartesian topology
- przesuwanie danych wzdłuż wybranego wymiaru
- kolektywna komunikacja względem wybranego wymiaru

Przykład (1)

MPI

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[]){
    MPI_Init(&argc, &argv);
    printf("Hello, world!\n");
    MPI_Finalize();
    return 0;
}
```

Politechnika Wrocławska

Identyfikacja procesów

MPI

- ile jest łącznie procesów?
- którym procesem jestem?

Politechnika Wrocławska

Przykład (2)

MPI

```c
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[] ){
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf( "I am %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Typy komunikacji

MPI

- broadcast
- multicast
- all-to-all
- barrier
- scatter
- gather
- all gather
- reduce

Politechnika Wrocławska

Politechnika
Wrocławska

Przetwarzanie danych masowych

Wykład 12 – Języki do przetwarzania danych masowych

dr inż. Tomasz Kajdanowicz, Roman Bartusiak, Piotr Bielak,
Krzysztof Rajda

10 stycznia 2022 r.