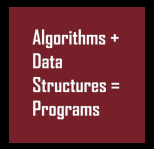
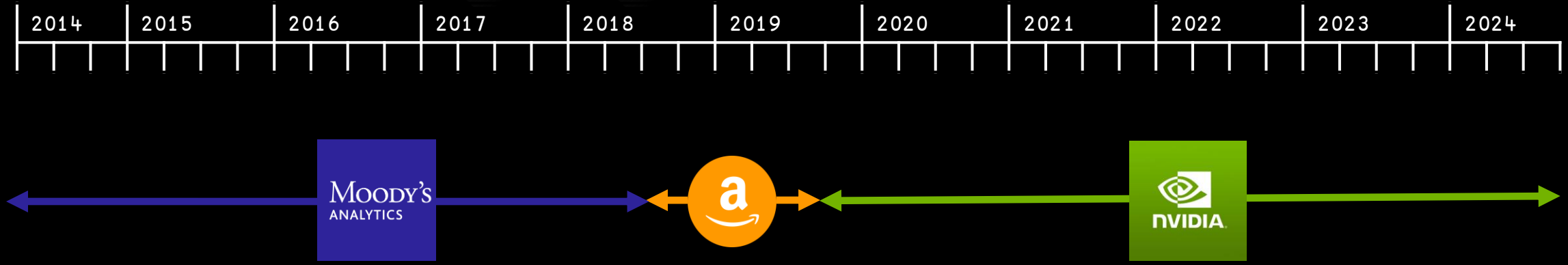
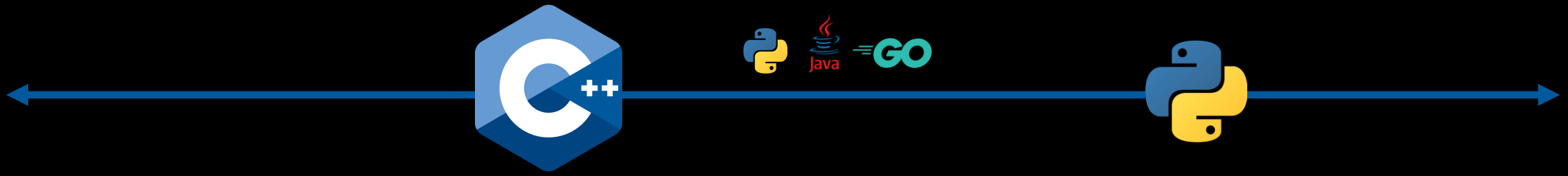
Arrays, Fusion & CPUs vs GPUs

Conor Hoekstra

code_report | codereport



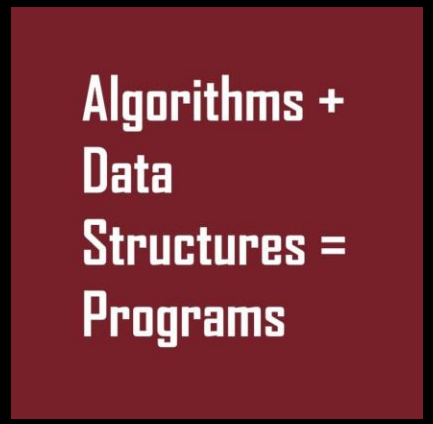




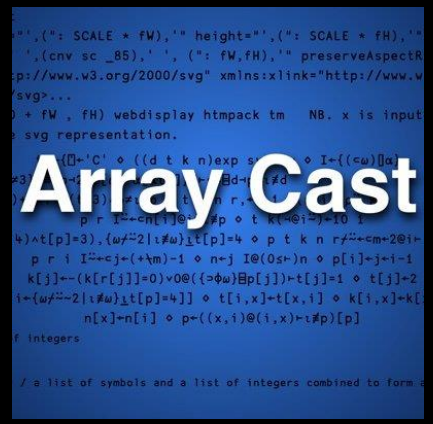
About Me

Conor Hoekstra / @code_report

351 Videos 40 (28) Talks



203 Episodes
@adspthepodcast



89 Episodes
@arraycast

5 Episodes
@codereport

20 Episodes
@conorhoekstra

Home
Shorts
Subscriptions
YouTube Mu...
You
Downloads

code_report

@code_report · 60.8K subscribers · 352 videos

Welcome to the code_report YouTube channel. ...more

twitter.com/code_report and 3 more links

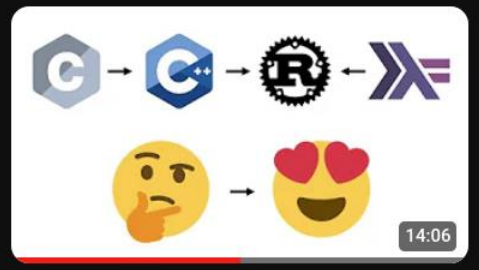
Customize channel | Manage videos

Home | Videos | Live | Podcasts | Playlists | Community

Latest | Popular | Oldest

1 Problem, 24 Programming Languages

375K views • 1 year ago



From C ➡ C++ ➡ Rust

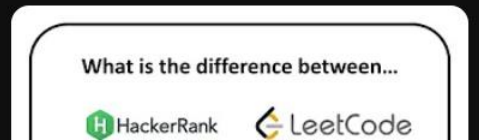
170K views • 1 year ago

1 Problem, 16 Programming Languages (C++ vs Rust vs Haskell vs Python vs APL...)

158K views • 3 years ago

Functional vs Array Programming

131K views • 3 years ago



https://github.com/codereport/Content

Problem

RMMMI

RMMMI

Reduce Map Map Map Iota

Live Code in BQN

+´1+2×10|↕100

```cpp
#include <print>
#include <ranges>

auto rmmmi(int n) -> int {
    auto res = 0;
    for (int i = 1; i < n; i += 1)
        res += (i % 10) * 2 + 1;
    return res;
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <print>
#include <ranges>

auto rmmmi(int n) -> int {
    auto res = 0;
    for (auto const i : std::views::iota(1, n))
        res += (i % 10) * 2 + 1;
    return res;
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <algorithm>
#include <numeric>
#include <print>
#include <vector>

auto rmmmi(int n) -> int {
    auto vec = std::vector<int>(n - 1);
    std::iota(vec.begin(), vec.end(), 1);
    std::transform(vec.begin(), vec.end(), vec.begin(), [](auto e) { return (e % 10) * 2 + 1; });
    return std::accumulate(vec.begin(), vec.end(), 0, std::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <algorithm>
#include <numeric>
#include <print>
#include <vector>

auto rmmmi(int n) -> int {
    auto vec = std::vector<int>(n - 1);
    std::iota(vec.begin(), vec.end(), 1);
    return std::transform_reduce( //
      vec.begin(),
      vec.end(),
      0,
      std::plus{},
      [](auto e) { return (e % 10) * 2 + 1; });
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <algorithm>
#include <numeric>
#include <print>
#include <vector>

auto rmmmi(int n) -> int {
    auto i = std::views::iota(1);
    return std::transform_reduce(
      i.begin(),
      i.begin() + n,
      0,
      std::plus{},
      [](auto e) { return (e % 10) * 2 + 1; });
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <algorithm>
#include <numeric>
#include <print>
#include <vector>

auto rmmmi(int n) -> int {
    auto vec = std::vector<int>(n - 1);
    std::iota(vec.begin(), vec.end(), 1);
    std::transform(vec.begin(), vec.end(), vec.begin(), [](auto e) { return (e % 10) * 2 + 1; });
    return std::accumulate(vec.begin(), vec.end(), 0, std::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <algorithm>
#include <numeric>
#include <print>
#include <vector>

auto rmmmi(int n) -> int {
    auto vec = std::vector<int>(n - 1);
    std::iota(vec.begin(), vec.end(), 1);
    std::transform(vec.begin(), vec.end(), vec.begin(), [](auto e) { return e % 10; });
    std::transform(vec.begin(), vec.end(), vec.begin(), [](auto e) { return e * 2; });
    std::transform(vec.begin(), vec.end(), vec.begin(), [](auto e) { return e + 1; });
    return std::accumulate(vec.begin(), vec.end(), 0, std::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <algorithm>
#include <print>
#include <ranges>

auto rmmmi(int n) -> int {
    return std::ranges::fold_left(
      std::views::iota(1, n)
      | std::views::transform([](auto e) { return e % 10; })
      | std::views::transform([](auto e) { return e * 2;  })
      | std::views::transform([](auto e) { return e + 1;  }),
      0,
      std::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <algorithm>
#include <print>
#include <ranges>

auto rmmmi(int n) -> int {
    return std::ranges::fold_left(
      std::views::iota(1, n)
      | std::views::transform([](auto e) { return (e % 10) * 2 + 1; }),
      0,
      std::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```cpp
#include <algorithm>
#include <print>
#include <ranges>

auto rmmmi(int n) -> int {
    return std::ranges::fold_left(
      std::views::iota(1, n)
      | std::views::transform([](auto e) { return e % 10; })
      | std::views::transform([](auto e) { return e * 2;  })
      | std::views::transform([](auto e) { return e + 1;  }),
      0,
      std::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

[[digression]]

https://godbolt.org/z/v48dTjfEG

[[end of digression]]

Comparison

Comparison

Comparison

Versions

C++	gcc	14.1
C++	nvcc	12.5
C++	nvc++	24.7
Rust		1.81
Swift		5.10
Python		3.12

```cpp
#include <algorithm>
#include <print>
#include <ranges>

auto rmmmi(int n) -> int {
    return std::ranges::fold_left(
      std::views::iota(1, n)
      | std::views::transform([](auto e) { return e % 10; })
      | std::views::transform([](auto e) { return e * 2;  })
      | std::views::transform([](auto e) { return e + 1;  }),
      0,
      std::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

+ Range-v3

```cpp
#include <algorithm>
#include <print>

#include <range/v3/all.hpp>

auto rmmmi(int n) -> int {
    return ranges::fold_left(
      ranges::views::iota(1, n)
      | ranges::views::transform([](auto e) { return e % 10; })
      | ranges::views::transform([](auto e) { return e * 2;  })
      | ranges::views::transform([](auto e) { return e + 1;  }),
      0,
      std::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

+ Flux

```
#include <algorithm>
#include <print>

#include <flux.hpp>

auto rmmmi(int n) -> int {
    return flux::iota(1, n)
      .map([](auto e) { return e % 10; })
      .map([](auto e) { return e * 2; })
      .map([](auto e) { return e + 1; })
      .sum();
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

+ Flux

```cpp
#include <algorithm>
#include <print>

#include <combinators.hpp>
#include <flux.hpp>

using namespace combinators;

auto rmmmi(int n) -> int {
    return flux::iota(1, n)
      .map(mod_(10))
      .map(_mul(2))
      .map(_plus(1))
      .sum();
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

+ Flux

```cpp
#include <algorithm>
#include <print>

#include <flux.hpp>

auto rmmmi(int n) -> int {
    return flux::iota(1, n)
      .map([](auto e) { return e % 10; })
      .map([](auto e) { return e * 2; })
      .map([](auto e) { return e + 1; })
      .sum();
}

auto main() -> int {
    auto const n = 100001;
    std::print("{}", rmmmi(n));
    return 0;
}
```

```rust
fn rmmmi(n: i32) -> i32 {
    (1..n)
        .map(|x| x % 10)
        .map(|x| x * 2)
        .map(|x| x + 1)
        .sum();
}

pub fn main() {
    let n = 100001;
    print!("{}", rmmmi(n));
}
```

[[digression]]

```rust
fn rmmmi(n: i32) -> i32 {
    (1..n)
        .map(|x| x % 10)
        .map(|x| x * 2)
        .map(|x| x + 1)
        .sum();
}

pub fn main() {
    let n = 100001;
    print!("{}", rmmmi(n));
}
```

```rust
fn rmmmi(n: i32) -> i32 {
    return (1..n)
        .into_iter()
        .map(|x| x % 10)
        .map(|x| x * 2)
        .map(|x| x + 1)
        .sum();
}

pub fn main() {
    let n = 100001;
    print!("{}", rmmmi(n));
}
```

cargo clippy

```
warning: unneeded `return` statement
 --> __fused_rust_iter.rs:2:5
  |
2 | /     return (1..n)
3 | |         .into_iter()
4 | |         .map(|x| x % 10)
5 | |         .map(|x| x * 2)
6 | |         .map(|x| x + 1)
7 | |         .sum();
  | |_______________^
  |
```

```
warning: useless conversion to the same type: `std::ops::Range<i32>`
 --> __fused_rust_iter.rs:2:12
  |
2 |       return (1..n)
  |  ____________^
3 | |         .into_iter()
  | |_____________________^ help: consider removing `.into_iter()`: `(1..n)`
  |
```

```
cargo clippy --fix
```

[[end of digression]]

```rust
fn rmmmi(n: i32) -> i32 {
    (1..n)
        .map(|x| x % 10)
        .map(|x| x * 2)
        .map(|x| x + 1)
        .sum();
}

pub fn main() {
    let n = 100001;
    print!("{}", rmmmi(n));
}
```

```swift
func rmmmi(_ n: Int) -> Int {
    return (1...n).lazy
        .map { $0 % 10 }
        .map { $0 * 2 }
        .map { $0 + 1 }
        .reduce(0, +)
}

let n = 100000
print(rmmmi(n))
```

```python
def rmmmi(n: int) -> int:
    return sum((i % 10) * 2 + 1 for i in range(1, n))


n = 100001
print(rmmmi(n))
```

```python
import numpy as np

def rmmmi(n: int) -> int:
    return np.sum((np.arange(1, n) % 10) * 2 + 1)


n = 100001
print(rmmmi(n))
```

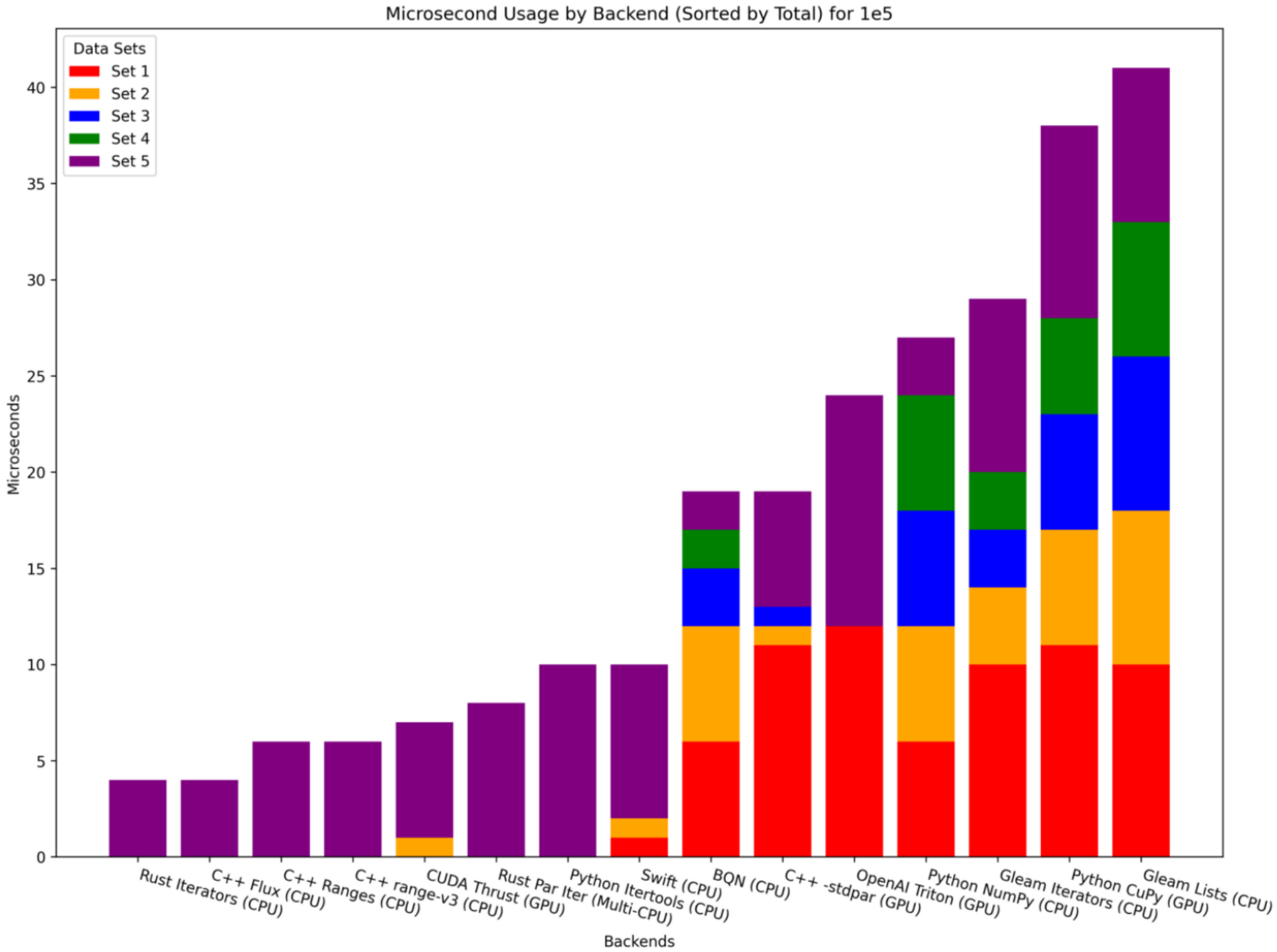
```python
import cupy as np

def rmmmi(n: int) -> int:
    return cp.sum((cp.arange(1, n) % 10) * 2 + 1)


n = 100001
print(rmmmi(n))
```

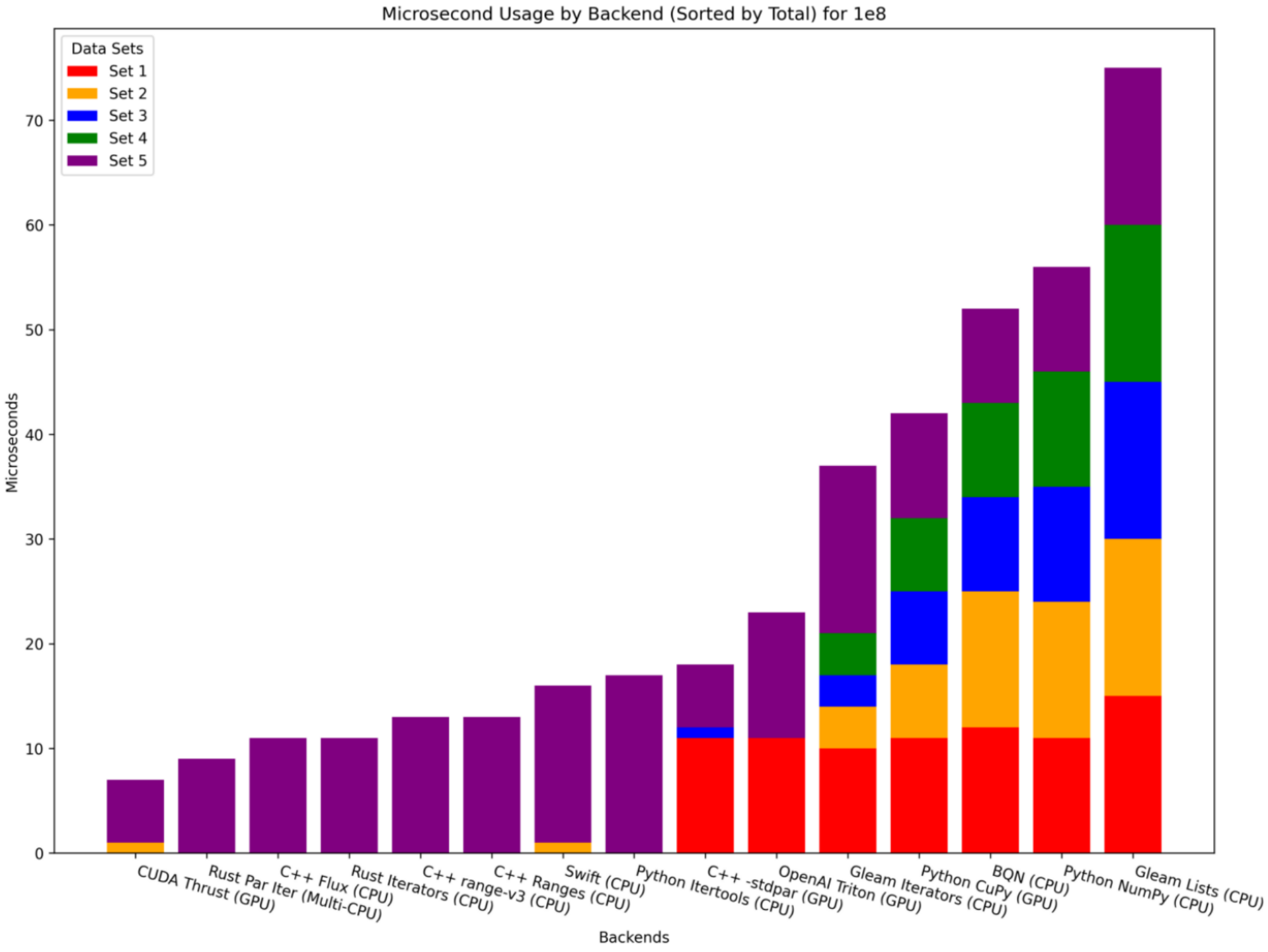
```cpp
#include <iostream>

#include <thrust/functional.h>
#include <thrust/iterator/counting_iterator.h>
#include <thrust/iterator/transform_iterator.h>
#include <thrust/reduce.h>

auto rmmmi(int n) -> int {
    auto a = thrust::make_counting_iterator(1);
    auto b = thrust::make_transform_iterator(a, [] __host__ __device__(int x) { return x % 10; });
    auto c = thrust::make_transform_iterator(b, [] __host__ __device__(int x) { return x * 2; });
    auto d = thrust::make_transform_iterator(c, [] __host__ __device__(int x) { return x + 1; });
    return thrust::reduce(d, d + n, 0, thrust::plus{});
}

auto main() -> int {
    auto const n = 100001;
    std::cout << rmmmi(n);
    return 0;
}
```

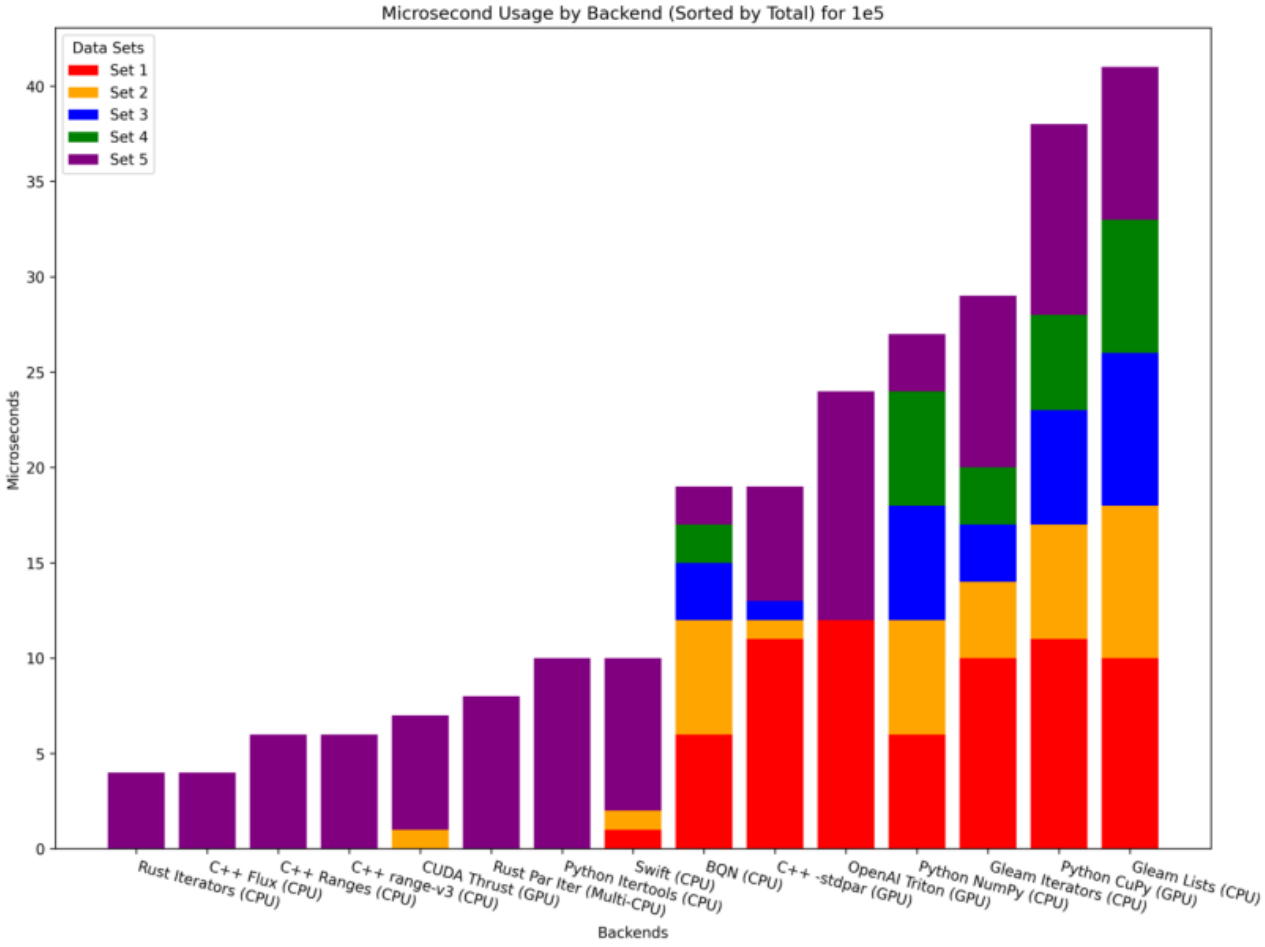
+ -stdpar

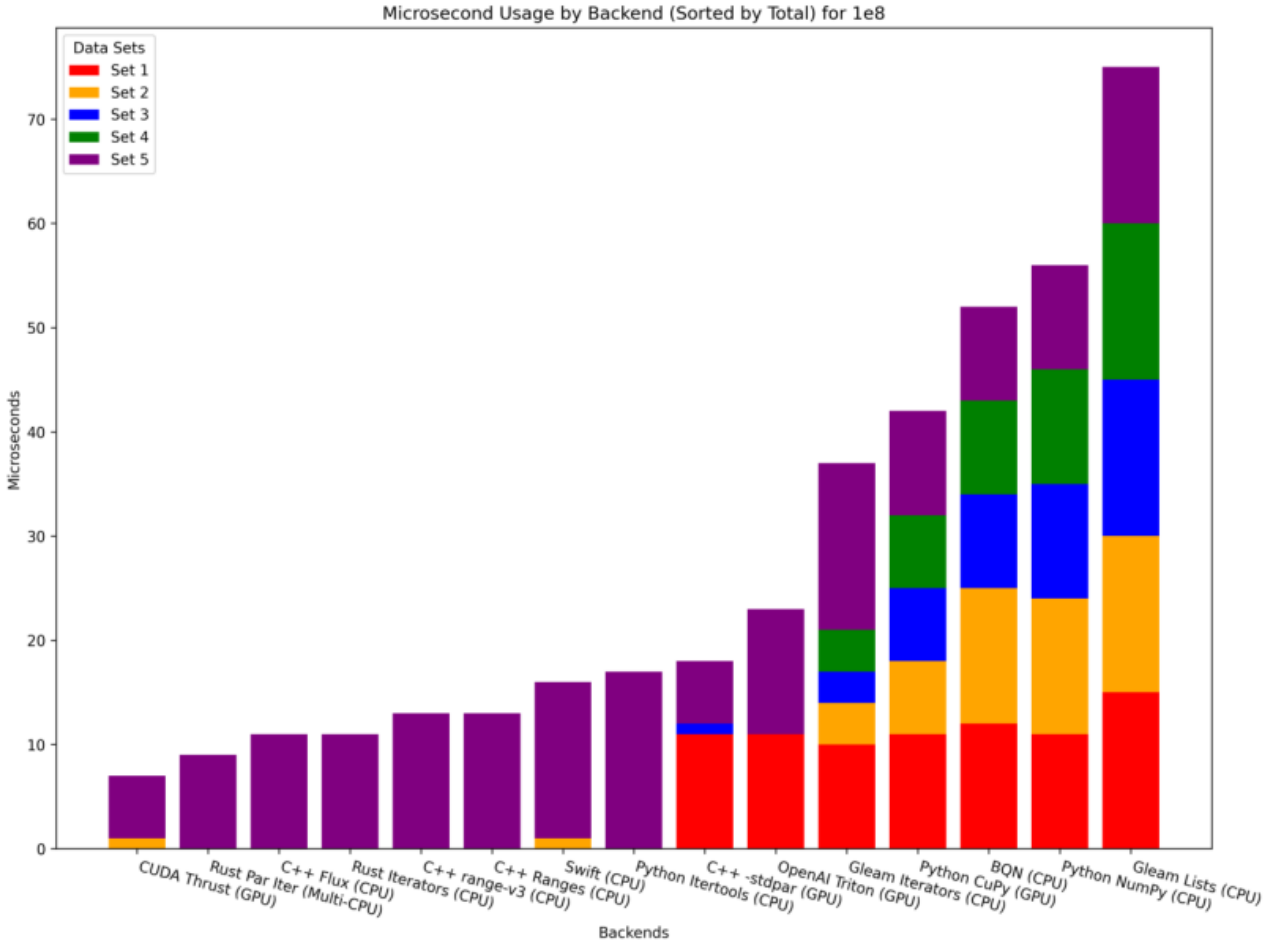
```cpp
#include <execution>
#include <iostream>
#include <numeric>
#include <ranges>

auto rmmmi(int n) -> int {
    auto a = std::views::iota(1, n + 1)
           | std::views::transform([](auto x) { return x % 10; })
           | std::views::transform([](auto x) { return x * 2;  })
           | std::views::transform([](auto x) { return x + 1;  });
    return std::reduce(std::execution::par_unseq,
                       a.begin(),
                       a.begin() + n,
                       0, std::plus{});
}

auto main() -> int {
    auto const n = 100000;
    std::cout << rmmmi(n);
    return 0;
}
```

Profiling


Microsecond Usage by Backend (Sorted by Total) for 1e5
Data Sets
Set 1
Set 2
Set 3
Set 4
Set 5
Microseconds
0
5
10
15
20
25
30
35
40
Rust Iterators (CPU)
C++ Flux (CPU)
C++ Ranges (CPU)
C++ range-v3 (CPU)
CUDA Thrust (GPU)
Rust Par Iter (Multi-CPU)
Python Itertools (CPU)
Swift (CPU)
BQN (CPU)
C++ -stdpar (GPU)
OpenAI Triton (GPU)
Python NumPy (CPU)
Gleam Iterators (CPU)
Python CuPy (GPU)
Gleam Lists (CPU)
Backends


Microsecond Usage by Backend (Sorted by Total) for 1e8
Data Sets
Set 1
Set 2
Set 3
Set 4
Set 5
Microseconds
0
10
20
30
40
50
60
70
CUDA Thrust (GPU)
Rust Par Iter (Multi-CPU)
C++ Flux (CPU)
Rust Iterators (CPU)
C++ range-v3 (CPU)
C++ Ranges (CPU)
Swift (CPU)
Python Itertools (CPU)
C++ -stdpar (GPU)
OpenAI Triton (GPU)
Gleam Iterators (CPU)
Python CuPy (GPU)
BQN (CPU)
Python NumPy (CPU)
Gleam Lists (CPU)
Backends


Microsecond Usage by Backend (Sorted by Total) for 1e5
Data Sets
Set 1
Set 2
Set 3
Set 4
Set 5
Microseconds
0
5
10
15
20
25
30
35
40
Rust Iterators (CPU)
C++ Flux (CPU)
C++ Ranges (CPU)
C++ range-v3 (CPU)
CUDA Thrust (GPU)
Rust Par Iter (Multi-CPU)
Python Itertools (CPU)
Swift (CPU)
BQN (CPU)
C++ -stdpar (GPU)
OpenAI Triton (GPU)
Python NumPy (CPU)
Gleam Iterators (CPU)
Python CuPy (GPU)
Gleam Lists (CPU)
Backends


Microsecond Usage by Backend (Sorted by Total) for 1e8
Data Sets
Set 1
Set 2
Set 3
Set 4
Set 5
Microseconds
0
10
20
30
40
50
60
70
CUDA Thrust (GPU)
Rust Par Iter (Multi-CPU)
C++ Flux (CPU)
Rust Iterators (CPU)
C++ range-v3 (CPU)
C++ Ranges (CPU)
Swift (CPU)
Python Itertools (CPU)
C++ -stdpar (GPU)
OpenAI Triton (GPU)
Gleam Iterators (CPU)
Python CuPy (GPU)
BQN (CPU)
Python NumPy (CPU)
Gleam Lists (CPU)
Backends

Ranking of Backend by Test Size

	1e3	1e4	1e5	1e6	1e7	1e8
0	C++ Flux (CPU)	C++ Flux (CPU)	C++ Flux (CPU)	CUDA Thrust (GPU)	CUDA Thrust (GPU)	CUDA Thrust (GPU)
1	Rust Iterators (CPU)	Rust Iterators (CPU)	C++ Ranges (CPU)	C++ Flux (CPU)	Rust Par Iter (Multi-CPU)	Rust Par Iter (Multi-CPU)
2	C++ Ranges (CPU)	C++ Ranges (CPU)	C++ range-v3 (CPU)	C++ range-v3 (CPU)	C++ Flux (CPU)	Python CuPy (GPU)
3	C++ range-v3 (CPU)	C++ range-v3 (CPU)	Rust Iterators (CPU)	C++ Ranges (CPU)	C++ range-v3 (CPU)	C++ Flux (CPU)
4	BQN (CPU)	BQN (CPU)	CUDA Thrust (GPU)	Rust Iterators (CPU)	C++ Ranges (CPU)	C++ range-v3 (CPU)
5	Swift (CPU)	Swift (CPU)	BQN (CPU)	BQN (CPU)	Rust Iterators (CPU)	Rust Iterators (CPU)
6	Python Itertools (CPU)	CUDA Thrust (GPU)	Python NumPy (CPU)	Rust Par Iter (Multi-CPU)	BQN (CPU)	C++ Ranges (CPU)
7	Python NumPy (CPU)	Python NumPy (CPU)	Swift (CPU)	Python NumPy (CPU)	Python NumPy (CPU)	Circle --stdpar (GPU)
8	CUDA Thrust (GPU)	Python Itertools (CPU)	Python Itertools (CPU)	Swift (CPU)	Python CuPy (GPU)	BQN (CPU)
9	Gleam Lists (CPU)	Gleam Lists (CPU)	Rust Par Iter (Multi-CPU)	C++ -stdpar (GPU)	C++ -stdpar (GPU)	C++ -stdpar (GPU)
10	Rust Par Iter (Multi-CPU)	Rust Par Iter (Multi-CPU)	Gleam Lists (CPU)	Python CuPy (GPU)	Swift (CPU)	OpenAI Triton (GPU)
11	Gleam Iterators (CPU)	Gleam Iterators (CPU)	Gleam Iterators (CPU)	Python Itertools (CPU)	Circle --stdpar (GPU)	Python NumPy (CPU)
12	C++ -stdpar (GPU)	C++ -stdpar (GPU)	C++ -stdpar (GPU)	Gleam Lists (CPU)	Python Itertools (CPU)	Swift (CPU)
13	Python CuPy (GPU)	Python CuPy (GPU)	Python CuPy (GPU)	Gleam Iterators (CPU)	Gleam Lists (CPU)	Python Itertools (CPU)
14	Circle --stdpar (GPU)	Circle --stdpar (GPU)	Circle --stdpar (GPU)	Circle --stdpar (GPU)	Gleam Iterators (CPU)	Gleam Lists (CPU)
15	OpenAI Triton (GPU)	OpenAI Triton (GPU)	OpenAI Triton (GPU)	OpenAI Triton (GPU)	OpenAI Triton (GPU)	Gleam Iterators (CPU)

1e3

1e4

1e5

1e6

1e7

1e8


Thrust
Thrust
Thrust
Thrust
Thrust

Conclusion

Conclusion

Conclusion

- C++:
  - Prefer "fusion" libraries
  - Flux is 🔥🔥🔥
- Other:
  - Rust tooling is ❤️
  - Swift perf is 😐
  - Numpy > Python
  - For GPU, use Thrust or

Thank You

https://github.com/codereport/Content/Talks

Conor Hoekstra

code_report

codereport

Questions?

https://github.com/codereport/Content/Talks

Conor Hoekstra

code_report

codereport