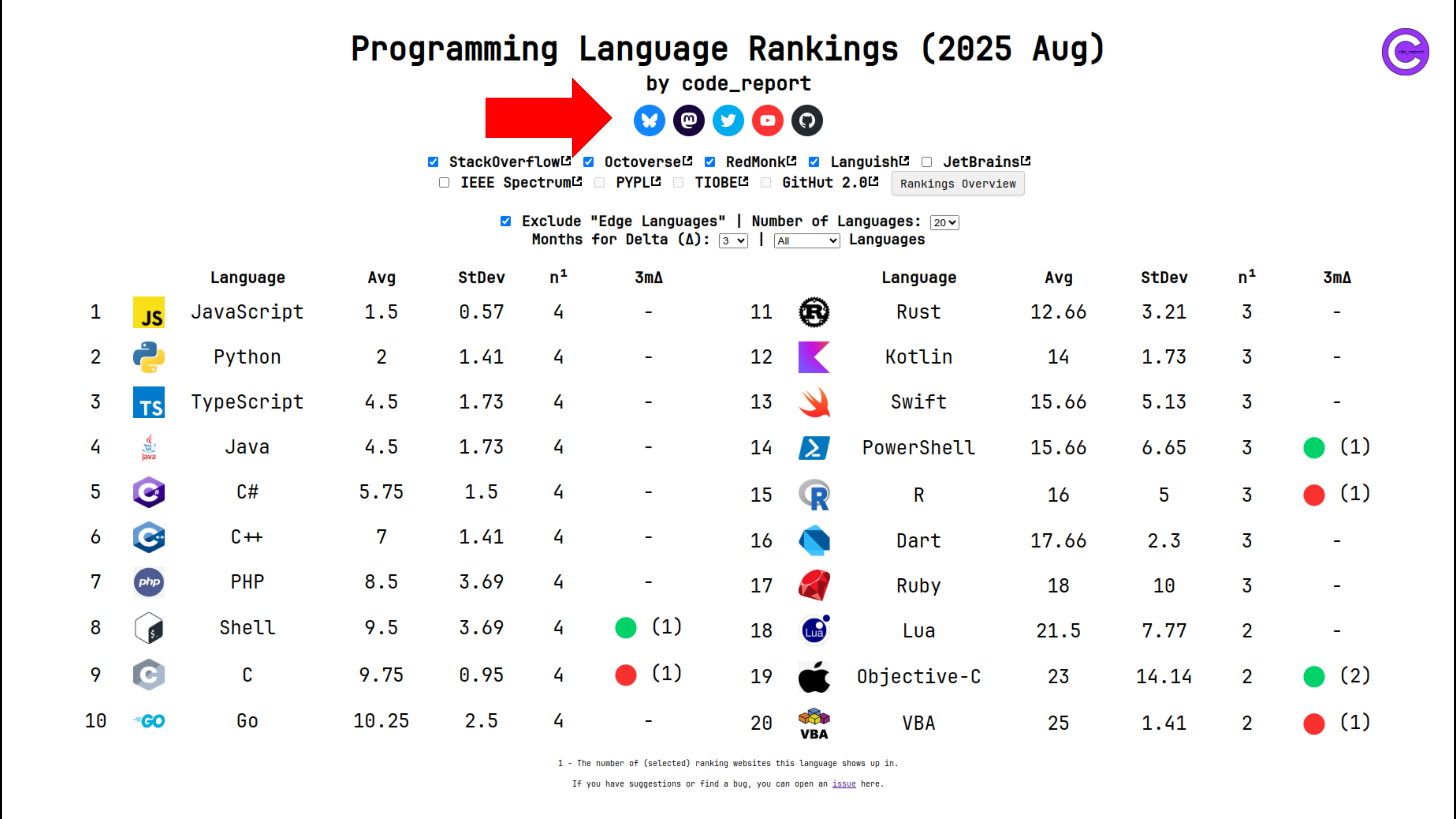
Postmodern for GPUs

Conor Hoekstra

code_report | codereport



Programming Language Rankings (2025 Aug)

by code_report

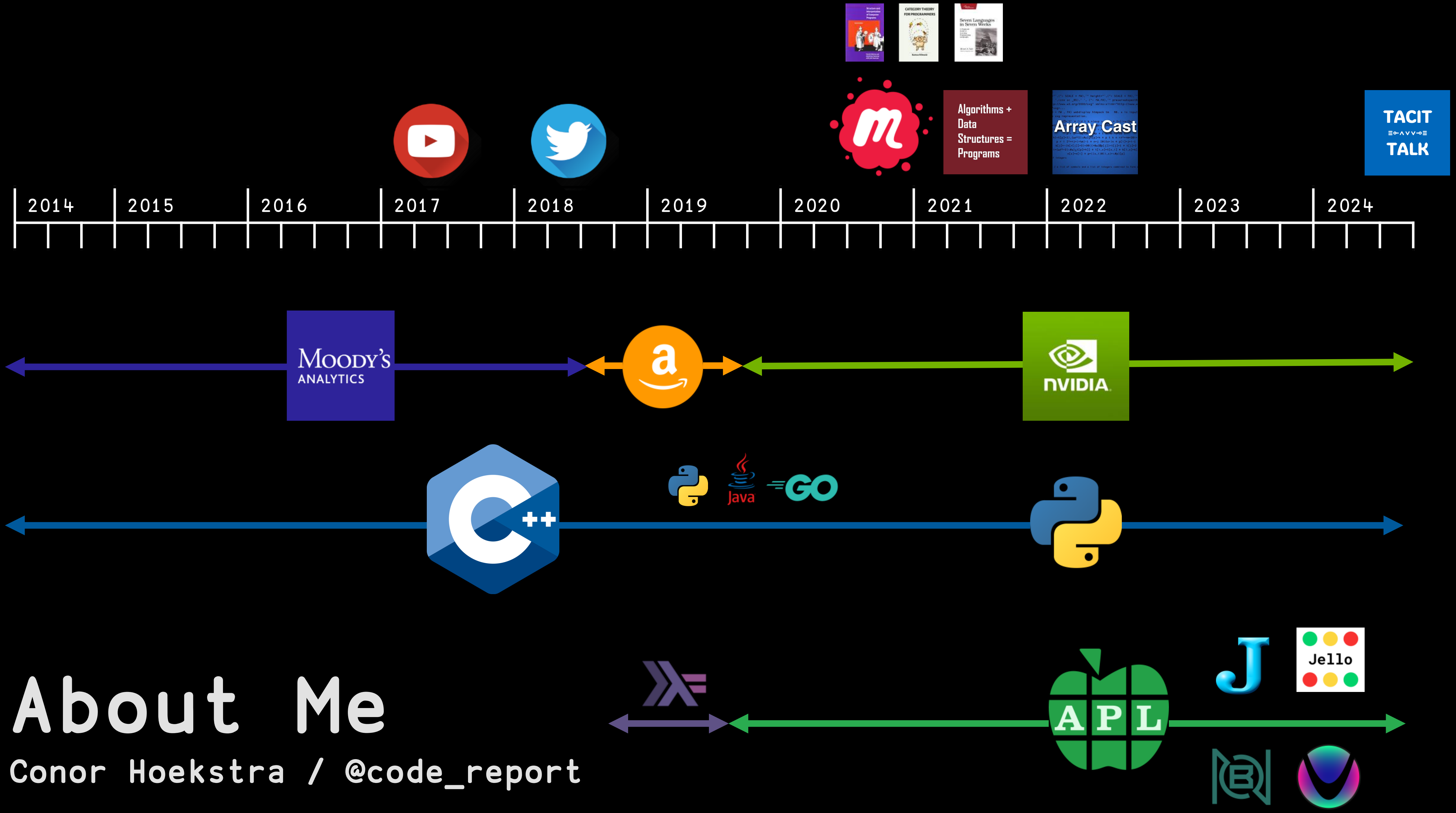
[✓] StackOverflow [✓] Octoverse [✓] RedMonk [✓] Languish [ ] JetBrains
[ ] IEEE Spectrum [ ] PYPL [ ] TIOBE [ ] GitHut 2.0 Rankings Overview

[✓] Exclude "Edge Languages" | Number of Languages: 20
Months for Delta (Δ): 3 | All Languages

	Language	Avg	StDev	n[1]	3mΔ
1	JavaScript	1.5	0.57	4	-
2	Python	2	1.41	4	-
3	TypeScript	4.5	1.73	4	-
4	Java	4.5	1.73	4	-
5	C#	5.75	1.5	4	-
6	C++	7	1.41	4	-
7	PHP	8.5	3.69	4	-
8	Shell	9.5	3.69	4	🟢 (1)
9	C	9.75	0.95	4	🔴 (1)
10	Go	10.25	2.5	4	-
11	Rust	12.66	3.21	3	-
12	Kotlin	14	1.73	3	-
13	Swift	15.66	5.13	3	-
14	PowerShell	15.66	6.65	3	🟢 (1)
15	R	16	5	3	🔴 (1)
16	Dart	17.66	2.3	3	-
17	Ruby	18	10	3	-
18	Lua	21.5	7.77	2	-
19	Objective-C	23	14.14	2	🟢 (2)
20	VBA	25	1.41	2	🔴 (1)

1 - The number of (selected) ranking websites this language shows up in.

If you have suggestions or find a bug, you can open an issue here.



About Me

Conor Hoekstra / @code_report

383 Videos

48 (32) Talks

Algorithms + Data Structures = Programs

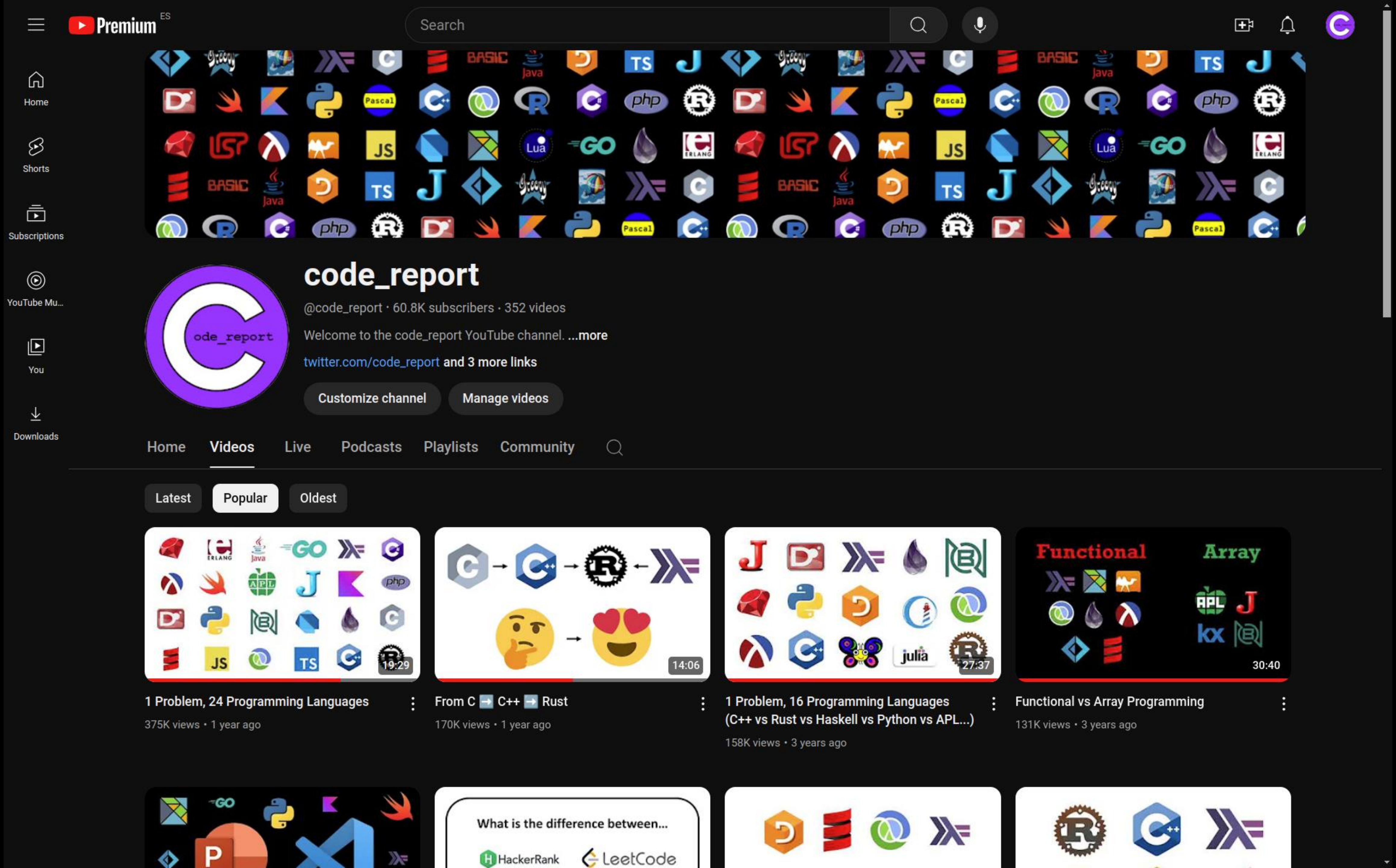
Array Cast

TACIT TALK

252 Episodes
@adspthepodcast

115 Episodes
@arraycast

27 Episodes
@codereport



Premium ES

Search

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
19:29
375K views • 1 year ago

From C ➡️ C++ ➡️ Rust
14:06
170K views • 1 year ago

1 Problem, 16 Programming Languages (C++ vs Rust vs Haskell vs Python vs APL...)
27:37
158K views • 3 years ago

Functional vs Array Programming
30:40
131K views • 3 years ago

What is the difference between...

HackerRank LeetCode

https://github.com/codereport/Content

C++ CORE COMPUTE LIBRARIES

The Standard Library for your entire system
https://github.com/NVIDIA/libcudacxx

The C++ parallel algorithms library
https://github.com/NVIDIA/thrust

Cooperative primitives for CUDA C++
https://github.com/NVIDIA/cub

CUDA CORE COMPUTE LIBRARIES

The Standard Library for your entire system
https://github.com/NVIDIA/tremailibcudacxx

The C++ parallel algorithms library
https://github.com/NVIDIA/thrust

Cooperative primitives for CUDA C++
https://NVIulA.cccl/tremain/cub

Overview

Problem 1

API Overview

Problem 2 →

Sum of Squares (SOS)

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

auto sos(int N) {
    return std::ranges::views::iota(0, N) |
           std::ranges::views::transform([](int x) { return x * x; }) |
           std::ranges::fold_left(0, std::plus{});
}

int main() { std::cout << sos(10) << std::endl; }
```

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

auto sos(int N) {
    return std::ranges::views::iota(0, N) |
           std::ranges::views::transform([](int x) { return x * x; }) |
           std::ranges::fold_left(0, std::plus{});
}

int main() { std::cout << sos(10) << std::endl; }
```

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

auto sos(int N) {
    return std::ranges::views::iota(0, N) |
           std::ranges::views::transform([](int x) { return x * x; }) |
           std::ranges::fold_left(0, std::plus{});
}

int main() { std::cout << sos(10) << std::endl; }
```

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

auto sos(int N) {
    return std::ranges::fold_left(
      std::ranges::views::iota(0, N) |
      std::ranges::views::transform([](int x) { return x * x; }),
    0,
    std::plus{});
}

int main() { std::cout << sos(10) << std::endl; }
```

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

auto sos(int N) {
    return std::ranges::fold_left(
      std::ranges::views::iota(0, N) |
      std::ranges::views::transform([](int x) { return x * x; }),
    0,
    std::plus{});
}

int main() { std::print("{}", sos(10)); }
```

Thrust

```cpp
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/host_vector.h>
#include <thrust/reduce.h>
#include <thrust/sequence.h>
#include <thrust/transform.h>
#include <iostream>

// Functor to square a number
struct square {
    __host__ __device__ int operator()(const int& x) const { return x * x; }
};

auto sos(int N) {
    // Create a device vector and fill it with sequence 0, 1, 2, ..., N-1
    thrust::device_vector<int> d_vec(N);
    thrust::sequence(d_vec.begin(), d_vec.end());

    // Square each element and sum the result
    return thrust::transform_reduce(
      d_vec.begin(), d_vec.end(), square(), 0, thrust::plus<int>());
}

int main() {
   auto result = sos(10);
   std::cout << result << std::endl;
   return 0;
}
```

```
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/host_vector.h>
#include <thrust/reduce.h>
#include <thrust/sequence.h>
#include <thrust/transform_reduce.h>
#include <iostream>

// Functor to square a number
struct square {
    __host__ __device__ int operator()(const int& x) const { return x * x; }
};

auto sos(int N) {
    // Create a device vector and fill it with sequence 0, 1, 2, ..., N-1
    thrust::device_vector<int> d_vec(N);
    thrust::sequence(d_vec.begin(), d_vec.end());

    // Square each element and sum the result
    return thrust::transform_reduce(
      d_vec.begin(), d_vec.end(), square(), 0, thrust::plus<int>());
}

int main() {
   auto result = sos(10);
   std::cout << result << std::endl;
   return 0;
}
```

```cpp
#include <thrust/device_vector.h>
#include <thrust/sequence.h>
#include <thrust/transform_reduce.h>
#include <iostream>

// Functor to square a number
struct square {
    __host__ __device__ int operator()(const int& x) const { return x * x; }
};

auto sos(int N) {
    // Create a device vector and fill it with sequence 0, 1, 2, ..., N-1
    thrust::device_vector<int> d_vec(N);
    thrust::sequence(d_vec.begin(), d_vec.end());

    // Square each element and sum the result
    return thrust::transform_reduce(
      d_vec.begin(), d_vec.end(), square(), 0, thrust::plus<int>());
}

int main() {
   auto result = sos(10);
   std::cout << result << std::endl;
   return 0;
}
```

```
#include <thrust/device_vector.h>
#include <thrust/sequence.h>
#include <thrust/transform_reduce.h>
#include <iostream>

struct square {
    __host__ __device__ int operator()(const int& x) const { return x * x; }
};

auto sos(int N) {
    thrust::device_vector<int> d_vec(N);
    thrust::sequence(d_vec.begin(), d_vec.end());
    return thrust::transform_reduce(
      d_vec.begin(), d_vec.end(), square(), 0, thrust::plus<int>());
}

int main() {
   auto result = sos(10);
   std::cout << result << std::endl;
   return 0;
}
```

```cpp
#include <thrust/device_vector.h>
#include <thrust/sequence.h>
#include <thrust/transform_reduce.h>
#include <iostream>

auto sos(int N) -> int {
    thrust::device_vector<int> d_vec(N);
    thrust::sequence(d_vec.begin(), d_vec.end());
    return thrust::transform_reduce(
      d_vec.begin(),
      d_vec.end(),
      [] __host__ __device__(int x) { return x * x; },
      0,
      thrust::plus<int>());
}

int main() {
   auto result = sos(10);
   std::cout << result << std::endl;
   return 0;
}
```

```cpp
#include <thrust/device_vector.h>
#include <thrust/sequence.h>
#include <thrust/transform_reduce.h>
#include <iostream>

auto sos(int N) -> int {
    auto iota = thrust::counting_iterator<int>(0);
    return thrust::transform_reduce(
      iota,
      iota + N,
      [] __host__ __device__(int x) { return x * x; },
      0,
      thrust::plus<int>());
}

int main() {
   auto result = sos(10);
   std::cout << result << std::endl;
   return 0;
}
```

```cpp
#include <thrust/iterator/counting_iterator.h>
#include <thrust/iterator/transform_iterator.h>
#include <thrust/reduce.h>
#include <iostream>

auto sos(int N) -> int {
    auto iota = thrust::counting_iterator<int>(0);
    auto map = thrust::make_transform_iterator(
      iota, [] __host__ __device__(int x) { return x * x; });
    return thrust::reduce(map, map + N, 0);
}

int main() {
    auto result = sos(10);
    std::cout << result << std::endl;
    return 0;
}
```

Parrot

Parrot

A high level, parallel, array-based library
with implicit fusion

```
auto sos(int N) {

}
```

```
auto sos(int N) {
    return parrot::range(N)
}
```

```
auto sos(int N) {
    return parrot::range(N)
      .sq()
}
```

```
auto sos(int N) {
    return parrot::range(N)
        .sq()
        .sum();
}
```

```cpp
#include "parrot.hpp"

auto sos(int N) {
    return parrot::range(N)
        .sq()
        .sum();
}

int main() { sos(10).print(); }
```

```
#include "parrot.hpp"

auto sos(int N) {
    return parrot::range(N).sq().sum();
}

int main() { sos(10).print(); }
```

```
nvcc
--extended-lambda
-std=c++20
-arch=native
-I ~/parrot-cpp
file_name.cu
```

Parrot API

Fused

1-index Maps (Unary)

⭐ map
abs
dble
enumerate
even
half
log
exp
neg
odd
rand
sign
sq
sqrt

1-index Maps (Binary)

⭐ map2
add (+)
div (/)
gt (>)
gte (>=)
idiv
lt (<)
lte (<=)
max
min
minus (-)
times (*)
eq (==)
pairs

2-index Maps

⭐ map_adj
deltas
differ

Joins

append
prepend

Products

cross
outer

Reshapes

take
drop
transpose
reshape
cycle
repeat

Copying

replicate

Permutations

rev
gather

Conditionally Fused

Compactions

⭐ keep
filter
where
uniq
distinct

Materializing

Reductions

⭐ reduce
all
any
maxr
max_by_key
minr
minmax
prod
sum

Scans

⭐ scan
alls
anys
maxs
mins
prods
sums

Permutations

sort
sort_by
sort_by_key

Compactions

rle

Copying

replicate

Split-Reductions

chunk_by_reduce

Comparisons

match

Properties

size
rank
shape

Accessors

value
front
back
to_host

Array Creation

array
range
scalar
matrix

I/O

print

Function Objects

Accessors

fst
snd

Binary Operations

eq
gt
gte
lt
lte
max
min
mul
add

Problem 2

(taken from PaddlePaddle)

```cpp
// https://github.com/PaddlePaddle/Paddle/blob/80f1123eb0c...

template <typename T>
static void Get________(
  const phi::GPUContext& dev_ctx,
  const DenseTensor* input_tensor,
  const int64_t num_cols,
  const int64_t num_rows,
  T* out_tensor,
  int64_t* indices_tensor)
{
  DenseTensor input_tmp;
  input_tmp.Resize(common::make_ddim({num_rows, num_cols}));
  T* input_tmp_data = dev_ctx.Alloc<T>(&input_tmp);
  phi::Copy(dev_ctx, *input_tensor, dev_ctx.GetPlace(), false, &input_tmp);

  thrust::device_ptr<T> out_tensor_ptr(out_tensor);
  thrust::device_ptr<int64_t> indices_tensor_ptr(indices_tensor);

  for (int64_t i = 0; i < num_rows; ++i) {
    T* begin = input_tmp_data + num_cols * i;
    T* end = input_tmp_data + num_cols * (i + 1);
    thrust::device_vector<int64_t> indices_data(num_cols);
    thrust::sequence(thrust::device,
      indices_data.begin(),
      indices_data.begin() + num_cols);
    thrust::sort_by_key(thrust::device, begin, end, indices_data.begin());
    int unique = 1 + thrust::inner_product(thrust::device,
      begin,
      end - 1,
      begin + 1,
      0,
      thrust::plus<int>(),
      thrust::not_equal_to<T>());
    thrust::device_vector<T> keys_data(unique);
    thrust::device_vector<int64_t> cnts_data(unique);
    thrust::reduce_by_key(thrust::device,
      begin,
      end,
      thrust::constant_iterator<int>(1),
      keys_data.begin(),
      cnts_data.begin());
    auto it = thrust::max_element(
      thrust::device, cnts_data.begin(), cnts_data.begin() + unique);
    T ____ = keys_data[it - cnts_data.begin()];
    int64_t counts = cnts_data[it - cnts_data.begin()];
    auto pos = thrust::find(thrust::device, begin, end, mode);
    int64_t index = indices_data[pos - begin + counts - 1];
    out_tensor_ptr[i] = static_cast<T>(mode);
    indices_tensor_ptr[i] = static_cast<int64_t>(index);
  }
}
```

```cpp
// https://github.com/PaddlePaddle/Paddle/blob/80f1123eb0c...

template <typename T>
static void Get________(/* ... */) {
  // initialization

  for (int64_t i = 0; i < num_rows; ++i) {
    T* begin = input_tmp_data + num_cols * i;
    T* end = input_tmp_data + num_cols * (i + 1);
    thrust::device_vector<int64_t> indices_data(num_cols);
    thrust::sequence(thrust::device,
      indices_data.begin(),
      indices_data.begin() + num_cols);
    thrust::sort_by_key(thrust::device, begin, end, indices_data.begin());
    int unique = 1 + thrust::inner_product(thrust::device,
      begin,
      end - 1,
      begin + 1,
      0,
      thrust::plus<int>(),
      thrust::not_equal_to<T>());
    thrust::device_vector<T> keys_data(unique);
    thrust::device_vector<int64_t> cnts_data(unique);
    thrust::reduce_by_key(thrust::device,
      begin,
      end,
      thrust::constant_iterator<int>(1),
      keys_data.begin(),
      cnts_data.begin());
    auto it = thrust::max_element(
      thrust::device, cnts_data.begin(), cnts_data.begin() + unique);
    T ____ = keys_data[it - cnts_data.begin()];
    int64_t counts = cnts_data[it - cnts_data.begin()];
    auto pos = thrust::find(thrust::device, begin, end, mode);
    int64_t index = indices_data[pos - begin + counts - 1];
    out_tensor_ptr[i] = static_cast<T>(mode);
    indices_tensor_ptr[i] = static_cast<int64_t>(index);
  }
}
```

```cpp
int unique = 1 + thrust::inner_product(
  begin,
  end - 1,
  begin + 1,
  0,
  thrust::plus<int>(),
  thrust::not_equal_to<T>());
```

```cpp
int unique = 1 + thrust::inner_product(
  begin,
  end - 1,
  begin + 1,
  0,
  thrust::plus<int>(),
  thrust::not_equal_to<T>());
```

```
int unique = thrust::unique_count(begin, end);
```

[[digression]]

```
template <typename I>
auto unique_count(I first, I last) {
    return 1 + thrust::inner_product(first,
                                     last - 1,
                                     first + 1,
                                     0,
                                     thrust::plus{},
                                     thrust::not_equal_to{});
}
```

```cpp
template <typename I>
auto unique_count(I first, I last) {
    auto zip = thrust::make_zip_iterator(first, first + 1);
    auto map = thrust::make_transform_iterator(
      zip, thrust::make_zip_function(thrust::not_equal_to{}));
    return 1 + thrust::reduce(map,
                              map + (last - first - 1),
                              0,
                              thrust::plus{});
}
```

```
template <class Derived,
          class ForwardIt,
          class BinaryPred>
typename thrust::iterator_traits<ForwardIt>::difference_type
_CCCL_HOST_DEVICE
unique_count(execution_policy<Derived> &policy,
             ForwardIt first,
             ForwardIt last,
             BinaryPred binary_pred)
{
    if (first == last) {
        return 0;
    }
    auto size = thrust::distance(first, last);
    auto it   = thrust::make_zip_iterator(thrust::make_tuple(first, thrust::next(first)));
    return 1 + thrust::count_if(policy, it, thrust::next(it, size - 1),
                                  zip_adj_not_predicate<BinaryPred>{binary_pred});
}
```

```cpp
template <typename I>
auto unique_count_zip_count_if(I first, I last) {
    auto zip = thrust::make_zip_iterator(first, first + 1);
    auto neq = thrust::make_zip_function(thrust::not_equal_to{});
    return 1 + thrust::count_if(zip, zip + (last - first - 1), neq);
}
```

```
auto unique_count(auto data) {
    return data.map_adj(parrot::neq{}).sum() + 1;
}
```

```
auto unique_count(auto data) {
    return data.differ().sum() + 1;
}
```

	Thrust	
thrust::transform_reduce	tranform_iterator + reduce	⭐ map + reduce
thrust::unique_count	zip_iterator + count_if	⭐ map_adj + sum
thrust::tabulate	counting_iterator + transform	range + ⭐ map

[[end of digression]]

```
// https://github.com/PaddlePaddle/Paddle/blob/80f1123eb0c...

template <typename T>
static void Get________(/* ... */) {
  // initialization

  for (int64_t i = 0; i < num_rows; ++i) {
    T* begin = input_tmp_data + num_cols * i;
    T* end = input_tmp_data + num_cols * (i + 1);
    thrust::device_vector<int64_t> indices_data(num_cols);
    thrust::sequence(thrust::device,
      indices_data.begin(),
      indices_data.begin() + num_cols);
    thrust::sort_by_key(thrust::device, begin, end, indices_data.begin());
    int unique = thrust::unique_count(thrust::device, begin, end);
    thrust::device_vector<T> keys_data(unique);
    thrust::device_vector<int64_t> cnts_data(unique);
    thrust::reduce_by_key(thrust::device,
      begin,
      end,
      thrust::constant_iterator<int>(1),
      keys_data.begin(),
      cnts_data.begin());
    auto it = thrust::max_element(
      thrust::device, cnts_data.begin(), cnts_data.begin() + unique);
    T ____ = keys_data[it - cnts_data.begin()];
    int64_t counts = cnts_data[it - cnts_data.begin()];
    auto pos = thrust::find(thrust::device, begin, end, mode);
    int64_t index = indices_data[pos - begin + counts - 1];
    out_tensor_ptr[i] = static_cast<T>(mode);
    indices_tensor_ptr[i] = static_cast<int64_t>(index);
  }
}
```

```cpp
// https://github.com/PaddlePaddle/Paddle/blob/80f1123eb0c...

template <typename T>
static void Get________(/* ... */) {
  // initialization

  for (int64_t i = 0; i < num_rows; ++i) {
    T* begin          = input_tmp_data + num_cols * i;
    T* end            = input_tmp_data + num_cols * (i + 1);
    auto indices_data = thrust::device_vector<int64_t>(num_cols);

    thrust::sequence(indices_data.begin(), indices_data.begin() + num_cols);
    thrust::sort_by_key(begin, end, indices_data.begin());

    int unique     = thrust::unique_count(thrust::device, begin, end);
    auto keys_data = thrust::device_vector<T>(unique);
    auto cnts_data = thrust::device_vector<int64_t>(unique);

    thrust::reduce_by_key(
      begin,
      end,
      thrust::constant_iterator<int>(1),
      keys_data.begin(),
      cnts_data.begin());

    auto it = thrust::max_element(cnts_data.begin(), cnts_data.begin() + unique);

    T ____                 = keys_data[it - cnts_data.begin()];
    int64_t counts         = cnts_data[it - cnts_data.begin()];
    auto pos               = thrust::find(begin, end, mode);
    int64_t index          = indices_data[pos - begin + counts - 1];
    out_tensor_ptr[i]      = static_cast<T>(mode);
    indices_tensor_ptr[i] = static_cast<int64_t>(index);
  }
}
```

```cpp
template <typename Array>
auto GetModeBySort_Parrot(
        const Array& data, int num_rows, int num_cols) {
    using T = typename Array::value_type;
    std::vector<thrust::pair<T, int>> results;

    for (int r = 0; r < num_rows; ++r) {
        auto mode = parrot::stats::mode(data.row(r)).value();
        auto index = data.row(r).last_index_of(mode);
        results.push_back(thrust::make_pair(mode, index));
    }

    return parrot::array(results);
}
```

```cpp
template <typename Array>
auto GetModeBySort_Parrot(
        const Array& data, int num_rows, int num_cols) {
    using T = typename Array::value_type;
    std::vector<thrust::pair<T, int>> results;

    for (int r = 0; r < num_rows; ++r) {
        auto mode = data.row(r)
                        .sort()
                        .rle()
                        .max_by_key(parrot::snd())
                        .value();
        auto index = data.row(r).last_index_of(mode);
        results.push_back(thrust::make_pair(mode, index));
    }

    return parrot::array(results);
}
```

Parrot

A high level, parallel, array-based library
with implicit fusion

Thank You

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

Conor Hoekstra

code_report

codereport

Questions?

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

choekstra@nvidia.com

Conor Hoekstra

code_report

codereport