Functional GPU Programming

Conor Hoekstra

 code_report | codereport 

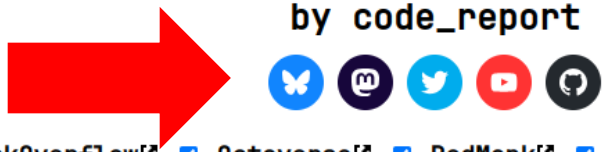
Array GPU Programming

Conor Hoekstra

 code_report | codereport 

Programming Language Rankings (2025 Aug)

by code_report



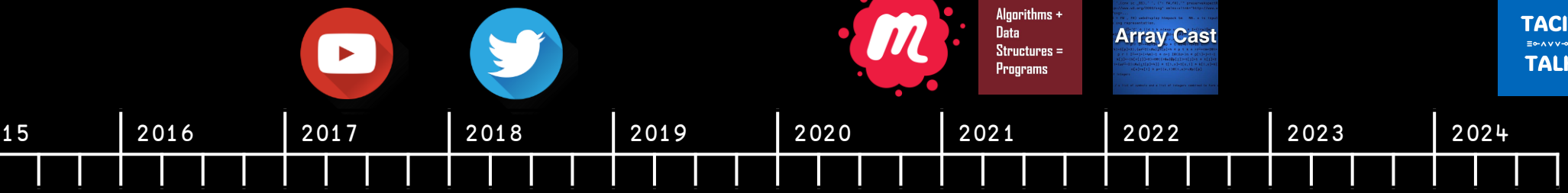
☑ StackOverflow ☑ Octoverse ☑ RedMonk ☑ Languish ☐ JetBrains
☐ IEEE Spectrum ☐ PYPL ☐ TIOBE ☐ GitHut 2.0 Rankings Overview

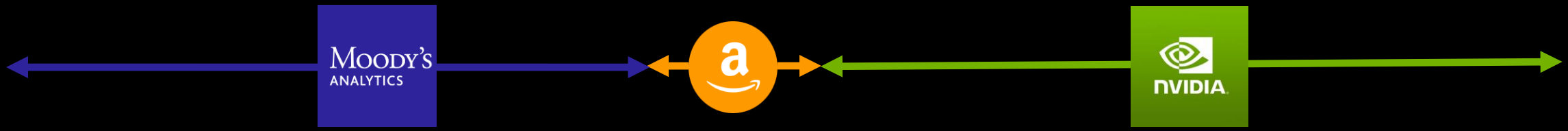
☑ Exclude "Edge Languages" | Number of Languages: 20
Months for Delta (Δ): 3 | All Languages

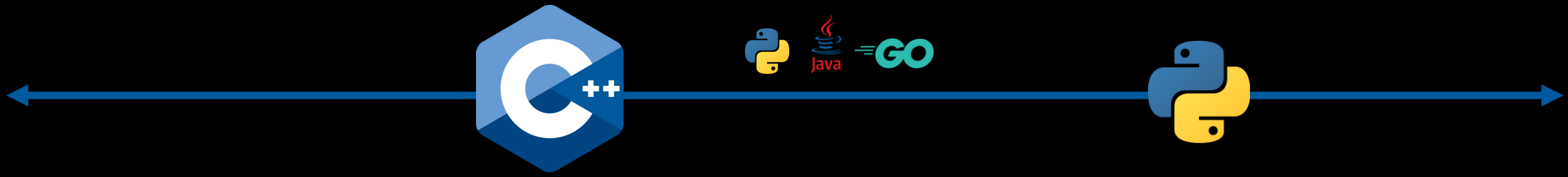
	Language	Avg	StDev	n[1]	3mΔ		Language	Avg	StDev	n[1]	3mΔ
1	JavaScript	1.5	0.57	4	-	11	Rust	12.66	3.21	3	-
2	Python	2	1.41	4	-	12	Kotlin	14	1.73	3	-
3	TypeScript	4.5	1.73	4	-	13	Swift	15.66	5.13	3	-
4	Java	4.5	1.73	4	-	14	PowerShell	15.66	6.65	3	🟢 (1)
5	C#	5.75	1.5	4	-	15	R	16	5	3	🔴 (1)
6	C++	7	1.41	4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





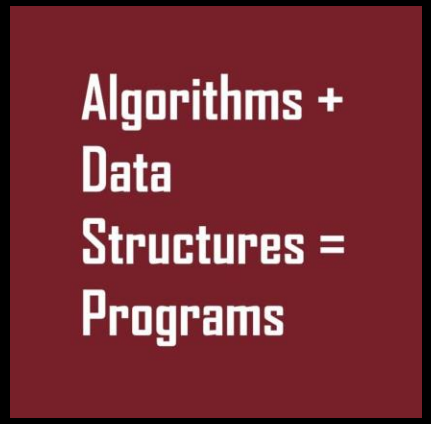


About Me

Conor Hoekstra / @code_report

383 Videos

48 (32) Talks



254 Episodes
@adspthepodcast

115 Episodes
@arraycast

27 Episodes
@codereport

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

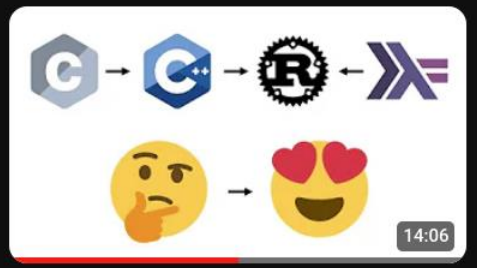
Customize channel | Manage videos

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

Parrot

Parrot

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

Parrot

https://github.com/nvlabs/parrot

Parrot

https://github.com/nvlabs/parrot

Parrot

https://github.com/nvlabs/parrot

Apache 2.0 License

Overview

Problem 1

API Overview

Problem 2

Problem 3

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;
}
```


```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_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;
}
```


```cpp
#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;
}
```

```
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();
}
```

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

```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(); }
```

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

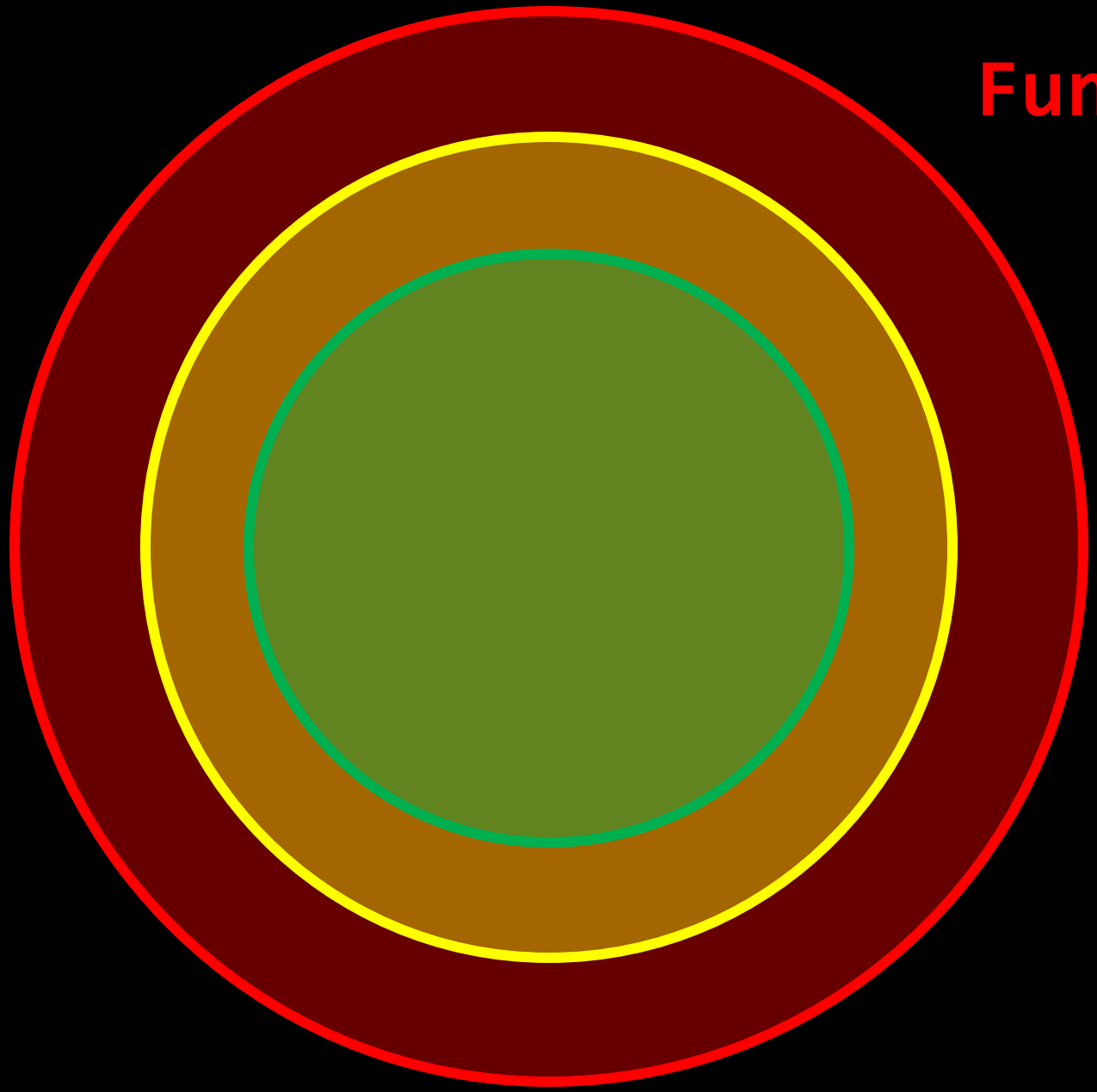
- eq
- gt
- gte
- lt
- lte
- max
- min
- mul
- add

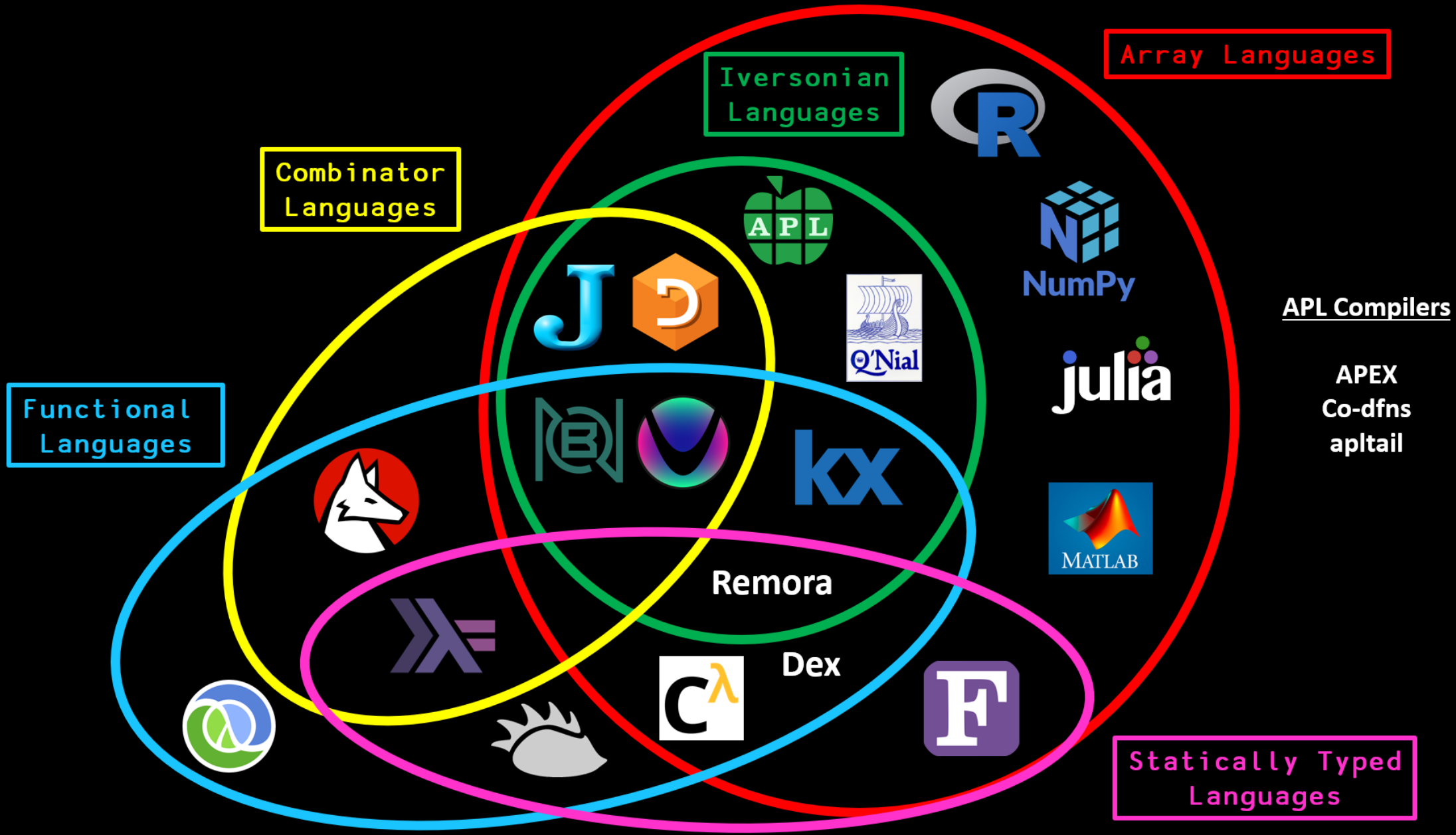
Functional vs Array Programming

Array = Functional
+ Rank Polymorphism

Array* = Functional
+ Rank Polymorphism
+ Symbols

* Iversonian (APL, BQN, Uiua)


Functional
Array
Iversonian
Array


Array Languages
Iversonian Languages
Combinator Languages
Functional Languages
Statically Typed Languages
APL
NumPy
Q'Nial
julia
kx
MATLAB
Remora
Dex
APL Compilers
APEX
Co-dfns
apltail

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

Fused
1-index Maps (Unary)

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

Fused
1-index Maps (Binary)

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

```cpp
auto add_one(auto& matrix) {
    for (auto& row : matrix) {
        for (auto& e : row) {
            e += 1;
        }
    }
    return matrix;
}
```

```cpp
auto add_one(const auto& matrix) {
    return matrix | std::ranges::views::transform([](const auto& row) {
                return row | std::ranges::views::transform(
                                 [](auto e) { return e + 1; });
            });
}
```

```
auto add_one(auto matrix) {
    return matrix + 1;
}
```

```
auto add_one(auto matrix) {
    return matrix.add(1);
}
```

```
auto add_one(auto matrix) {
    return matrix + 1;
}
```

Fused
1-index Maps (Unary)

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

Fused
1-index Maps (Binary)

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

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)


```
// 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>());
```


```
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]]


```cpp
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{});
}
```


```cpp
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::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]]


```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 = 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);
}
```

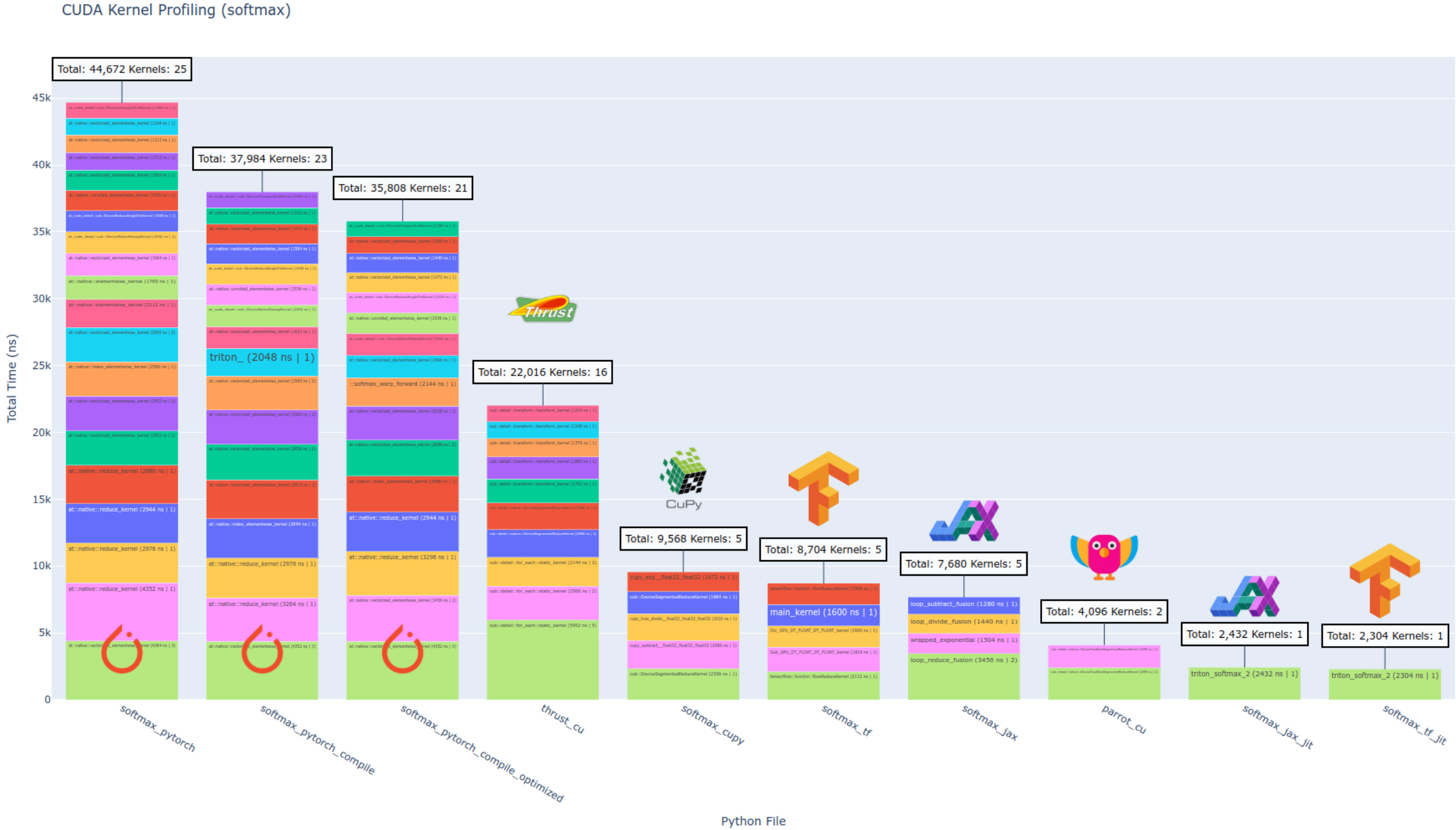
Softmax

```
int main() {
    auto matrix = // ... (matrix initialization)

    auto cols = matrix.shape()[1];
    auto z    = matrix - matrix.maxr<2>().replicate(cols);
    auto num  = z.exp();
    auto den  = num.sum<2>();
    (num / den.replicate(cols)).print();

    return 0;
}
```

kp


CUDA Kernel Profiling (softmax)
Total: 44,672 Kernels: 25
Total: 37,984 Kernels: 23
Total: 35,808 Kernels: 21
Total: 22,016 Kernels: 16
Total: 9,568 Kernels: 5
Total: 8,704 Kernels: 5
Total: 7,680 Kernels: 5
Total: 4,096 Kernels: 2
Total: 2,432 Kernels: 1
Total: 2,304 Kernels: 1
triton_ (2048 ns | 1)
::softmax_warp_forward (2144 ns | 1)
main_kernel (1600 ns | 1)
triton_softmax_2 (2432 ns | 1)
triton_softmax_2 (2304 ns | 1)
Thrust
CuPy
45k
40k
35k
30k
25k
20k
15k
10k
5k
0
Total Time (ns)
softmax_pytorch
softmax_pytorch_compile
softmax_pytorch_compile_optimized
thrust_cu
softmax_cupy
softmax_tf
softmax_jax
parrot_cu
softmax_jax_jit
softmax_tf_jit
Python File

Parrot

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

Parrot

https://github.com/nvlabs/parrot

Apache 2.0 License

Thank You

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

Conor Hoekstra

code_report

codereport

Free Instructor-Led CUDA C++ Training

As a C++ Under the Sea attendee, you have exclusive, free access to a full-day, virtual XLab. Join NVIDIA experts on October 22 as they explain how to build a CUDA application in C++, from start to finish.

Spaces are limited so register today to secure your place.

Register Now

NVIDIA

Questions?

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

choekstra@nvidia.com

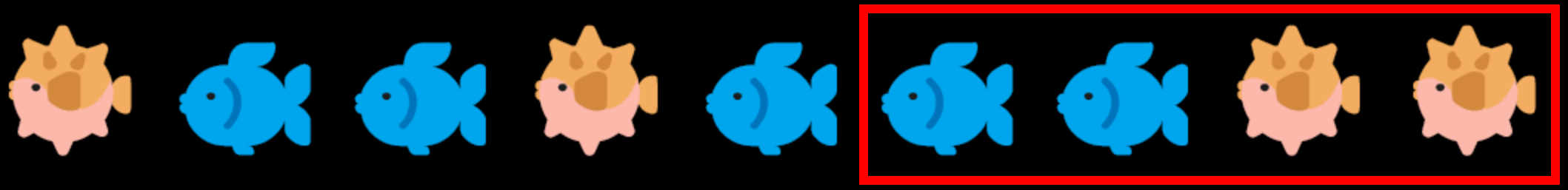
Conor Hoekstra

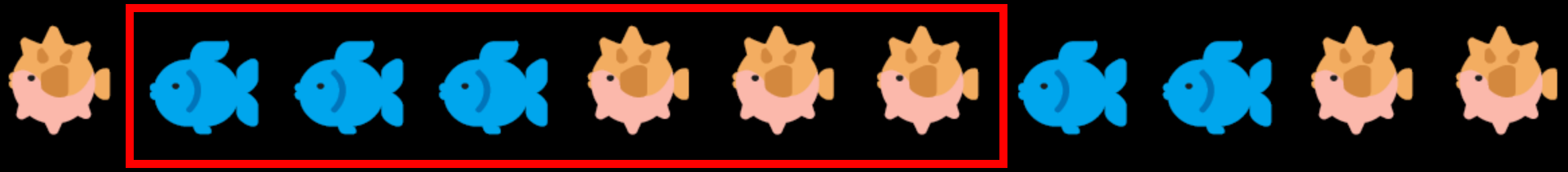
code_report

codereport

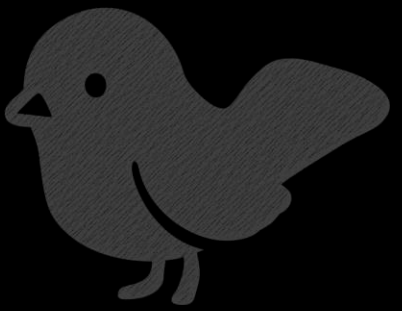
Sushi for Two

https://codeforces.com/contest/1138/problem/A





```cpp
template <int N>
constexpr auto sushi_for_two(std::array<int, N> sushi) {
    int current_sushi       = 0;
    int sushi_in_a_row      = 0;
    int prev_sushi_in_a_row = 0;
    int max_of_mins         = 0;
    for (auto const s : sushi) {
        if (current_sushi != s) {
            current_sushi = s;
            if (prev_sushi_in_a_row == 0) {
                prev_sushi_in_a_row = sushi_in_a_row;
                sushi_in_a_row      = 1;
            } else {
                auto const min      = std::min(sushi_in_a_row, prev_sushi_in_a_row);
                max_of_mins         = std::max(max_of_mins, min);
                prev_sushi_in_a_row = sushi_in_a_row;
                sushi_in_a_row      = 1;
            }
        } else {
            sushi_in_a_row += 1;
        }
    }
    auto const min = std::min(sushi_in_a_row, prev_sushi_in_a_row);
    max_of_mins    = std::max(max_of_mins, min);
    return max_of_mins * 2;
}
```



```cpp
using namespace ranges::views;
using namespace combinators;

auto sushi_for_two(std::vector<int> sushi) {
  auto indices = concat(                                    //
      concat(single(0),                                     //
             zip(zip_with(_neq_, sushi, sushi | drop(1)),   //
                 iota(1))                                   //
                 | filter(_fst)                             //
                 | values),                                 //
      single(sushi.size()));                                //
  auto deltas = zip_with(_sub_, indices | drop(1), indices);
  return 2 * ranges::max(zip_with(_min_, deltas, deltas | drop(1)));
}
```

```
auto sushi_for_two(auto sushi) {
    return sushi.differ()
        .where()
        .prepend(0)
        .append(sushi.size())
        .deltas()
        .map_adj(parrot::min{})
        .dble()
        .maxr();
}
```