Distributed Ranges for C++ / multi-XPU

What is this

... and why you may need it?

May 2024

Łukasz Ślusarczyk

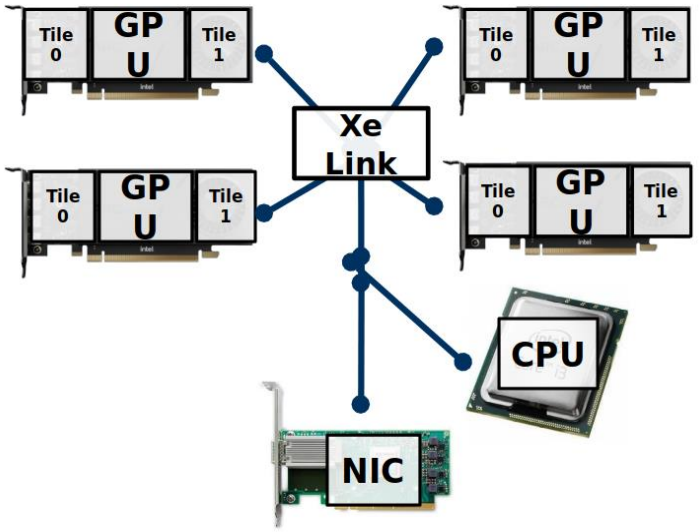
The problem – writing parallel programs is hard

Multiple memory domains

1. Nodes
2. CPUs inside nodes
3. GPUs
4. Tiles inside GPUs

Supernodes

- single-addressable space
- having NUMA



Software is needed to

reduce complexity

and

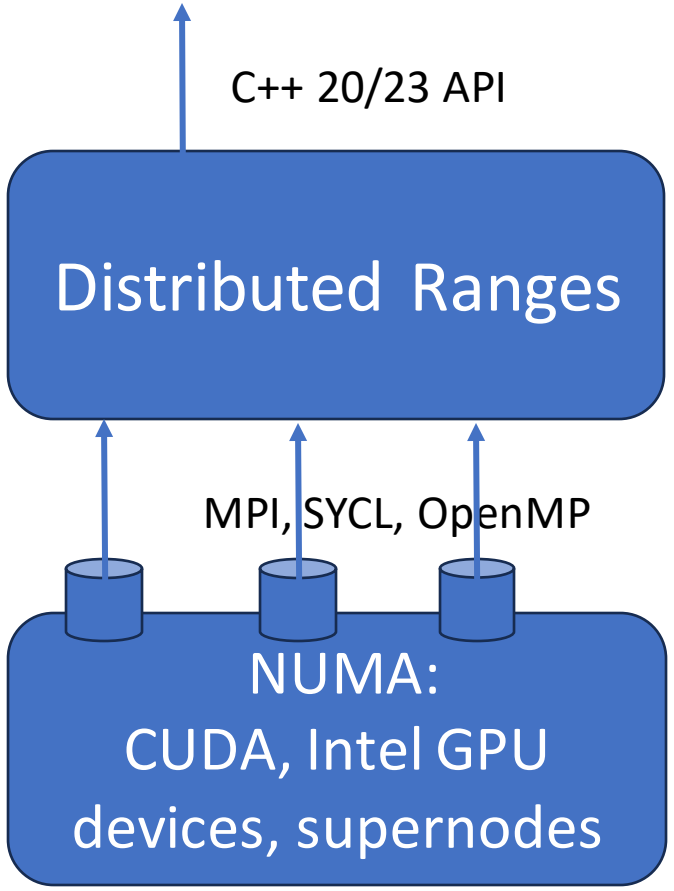
increase productivity

The solution – Distributed Ranges

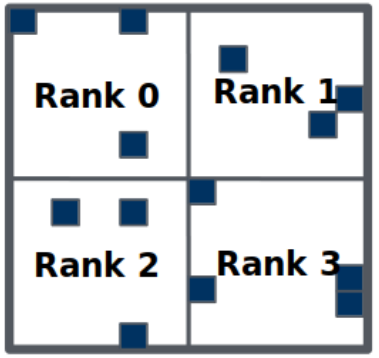
One may go **DIY**, to the classical, explicit programming, do MPI

… but with the library this is **automated**:

- **see** all the system **as a whole**
- write single-addressable program
- have **automatic resources management**, that exploits NUMA



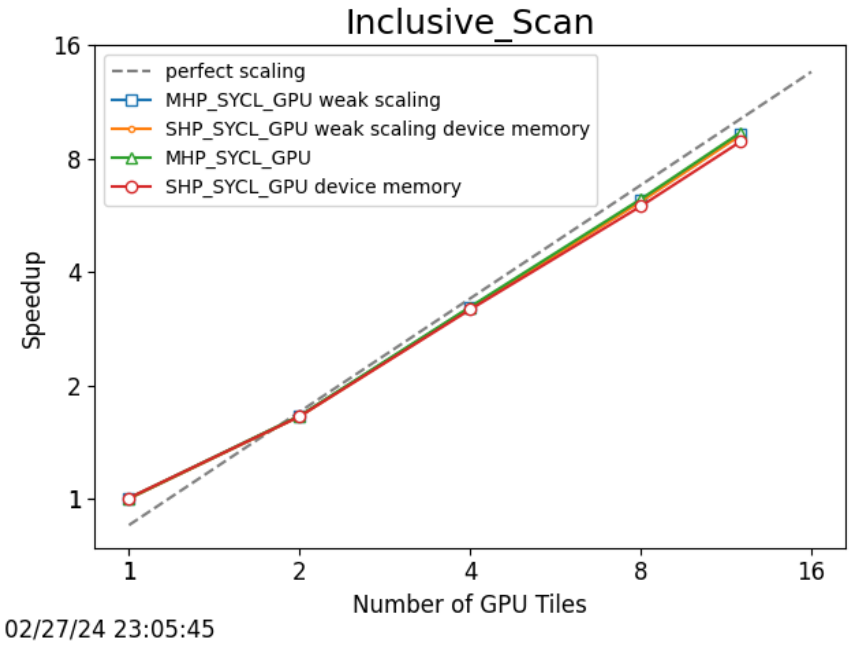
Productivity
Performance
Portability



Distributed Ranges provides C++ API:

- with **NUMA-aware allocators**
- **Distributed Data Structures (vector, matrix)**
- … and a set of useful **algorithms** on them
- (reduce, scan, sort, copy, transform…)
- … and ease communication (**halo exchange**)
- … and a way to write custom operators (**for_each**, **foreach_stencil**…)

Productivity
Performance
Portability



Achieve **high-performance**
for **multi-node**, **multi-GPU, supernode NUMA executions**

- good (near linear) weak/strong scaling
- Small overhead comparing to perfect solution (5-10%)

Productivity
Performance
Portability

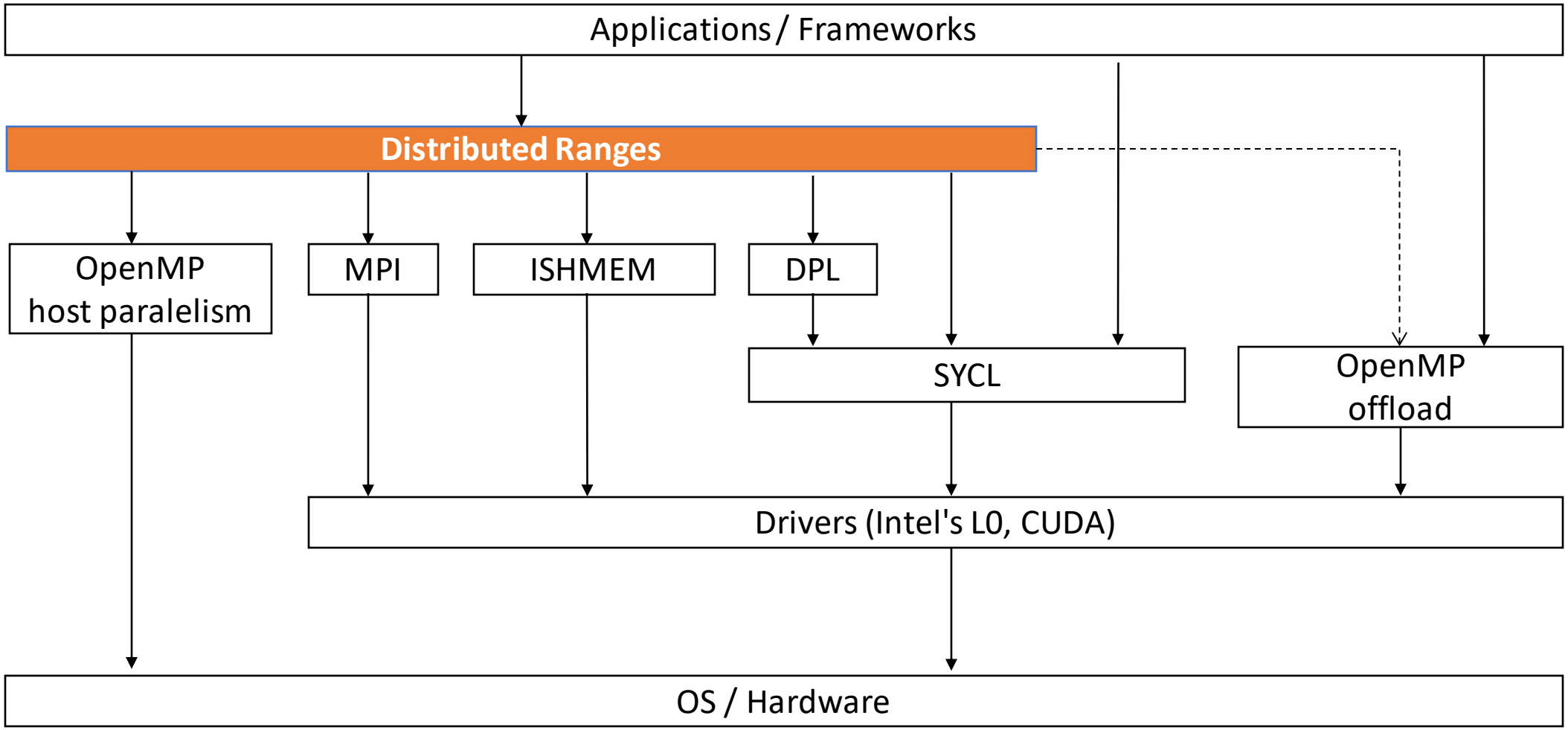
+ BONUS

```
float dot_product(vector<float>& x,
                  vector<float>& y) {

  auto z =  views::zip(x, y)
          | views::transform([](auto element)
{
             auto [a, b] = element;
             return a * b;
           });

  return reduce(par_unseq, z, 0, std::plus());
}
```

Offer **high-level, standard C++** way of writing distributed code.

- not yet-another-cpp-library
- use **C++20/23** concepts, ranges

Where Do Distributed Ranges Live?



What Distributed Ranges Can do

Create **Distributed Data Structures**

- NUMA aware allocators
- Vector
- Multi-Dimensional Array
- Halo support

Allow **low-lewel access**

- local memory (my segments)
- smart iterator (local & remote)
- fast iterator (local)
- zip
- sliding
- Communicator / rank / MPI window

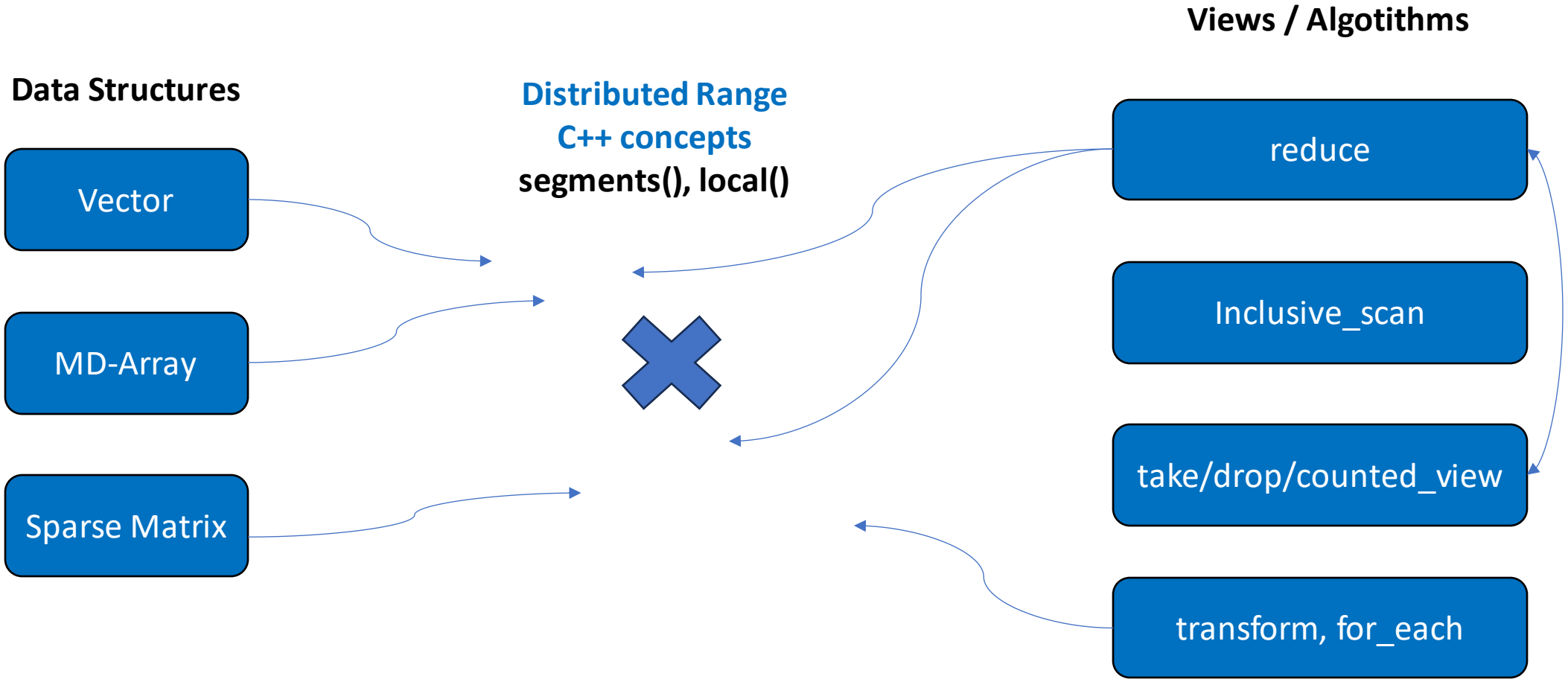
Execute **Algorithms**

- copy, fill, iota
- exclusive/inclusive scan, reduce
- transform, transpose
- sort
- for_each (for custom user code)

Create **views**

- enumerate, iota
- counted, drop, take
- transform
- zip
- sliding

Internals


Views / Algotithms
Data Structures
Distributed Range
C++ concepts
segments(), local()
Vector
MD-Array
Sparse Matrix
reduce
Inclusive_scan
take/drop/counted_view
transform, for_each

Where and how it runs?

Single Process Model

- multiple CPUs/GPUs handled inside **one process**
- scheduling **async SYCL** operations, waiting for them
- more like ehnanced oneDPL library
- can be used for single node only
- ideal for supernode

Multi Process Model

- multiple CPUs/GPUs handled **inside multiple processes**
- one process per one device
- scheduling **async SYCL** operations AND using **MPI** communication
- **SPMD** programming model
- can be used for **single and multiple nodes**

How to use it? ”Decoder” Example

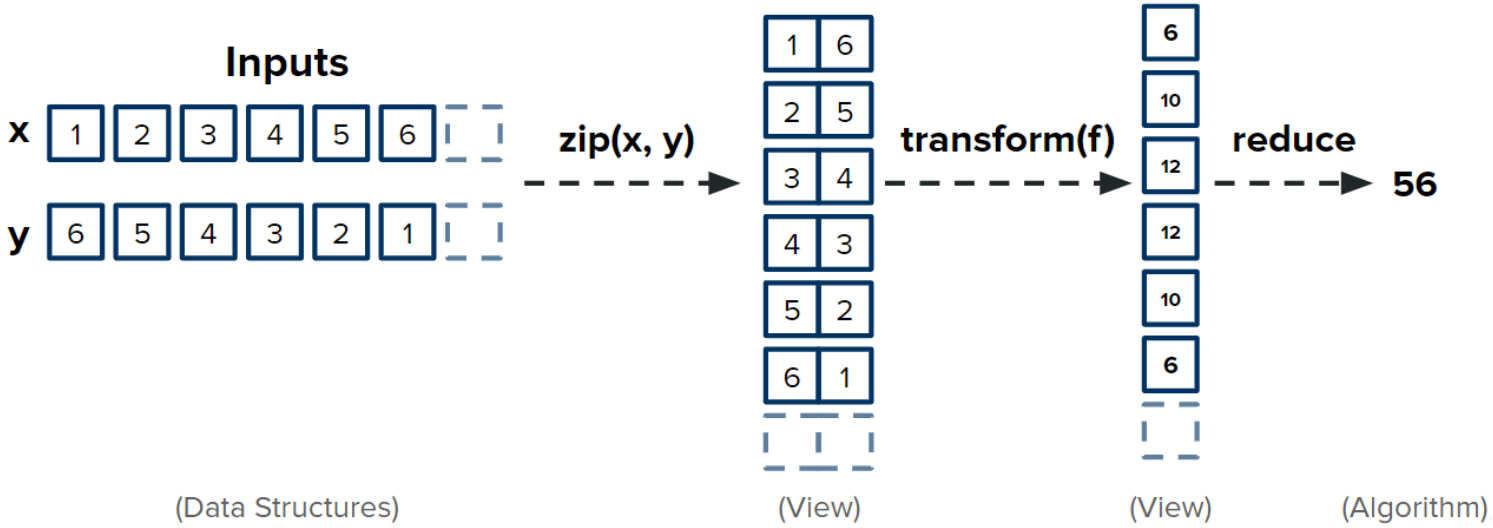
- Create distributed structure
- Split and copy parts of an encoded string on process number 0 to all devices
- **Perform decoding in parallel on each device**
- Copy back all decoded parts to a host memory on process number 0

```
1  #include <dr/mhp.hpp>
2  #include <fmt/core.h>
3
4  namespace mhp = dr::mhp;
5
6  int main(int argc, char **argv) {
7
8    mhp::init(sycl::default_selector_v);
9
10   mhp::distributed_vector<char> dv(81);
11   std::string decoded_string(80, 0);
12
13   mhp::copy(
14       0,
15       std::string("Mjqqt%|twqi&%Ymnx%nx%ywfsxrnxnts%kwtr%ymj%tsj%fsi%tsq~%"
16                   "Inxywngzyji%Wfsljx%wjfqr&"),
17       dv.begin());
18
19   mhp::for_each(dv, [](char &val) { val -= 5; });
20   mhp::copy(0, dv, decoded_string.begin());
21
22   if (mhp::rank() == 0)
23     fmt::print("{}\n", decoded_string);
24
25   mhp::finalize();
26
27   return 0;
28 }
```

Multi-process model (SPMD / MPI+SYCL)

”Dot Product” Example

```
int main(int argc, char **argv) {
  auto devices = dr::shp::get_numa_devices(sycl::default_selector_v);
  dr::shp::init(devices);

  std::size_t n = 100;

  dr::shp::distributed_vector<int> x(n);
  dr::shp::distributed_vector<int> y(n);

  std::iota(x.begin(), x.end(), 0);
  std::iota(y.begin(), y.end(), 0);

  auto v = dot_product_distributed(x, y);
```



Single-process model, SYCL only

```
8     template <dr::distributed_range X, dr::distributed_range Y>
9     auto dot_product_distributed(X &&x, Y &&y) {
10      auto z = dr::shp::views::zip(x, y) | dr::views::transform([](auto &&elem) {
11                 auto &&[a, b] = elem;
12                 return a * b;
13               });
14
15      return dr::shp::reduce(dr::shp::par_unseq, z, 0, std::plus());
16    }
```

”1D Cellular Automaton” Example

- Define **halo regions** in distributed structure
- **Halo exchange** operation
- **transform** in parallel by custom function

```
17    constexpr std::size_t asize = 60;
18    constexpr std::size_t steps = 60;
19
20    constexpr uint8_t ca_rule = 28;
21
22    auto newvalue = [](auto &&p) {
23      auto v = &p;
24      uint8_t pattern = 4 * v[-1] + 2 * v[0] + v[1];
25      return (ca_rule >> pattern) % 2;
26    };
```

```
28  int main(int argc, char **argv) {
29
30    mhp::init(sycl::default_selector_v);
31
32    auto dist = dr::mhp::distribution().halo(1);
33    mhp::distributed_vector<uint8_t> a1(asize + 2, 0, dist),
34        a2(asize + 2, 0, dist);
35
36    auto in = rng::subrange(a1.begin() + 1, a1.end() - 1);
37    auto out = rng::subrange(a2.begin() + 1, a2.end() - 1);
38
39    /* initial value of the automaton - customize it if you want to */
40    in[0] = 1;
41
42    if (mhp::rank() == 0)
43      fmt::print("{}\n", in);
44
45    for (std::size_t s = 0; s < steps; s++) {
46      dr::mhp::halo(in).exchange();
47
48      mhp::transform(in, out.begin(), newvalue);
49
50      std::swap(in, out);
51
52      /* fmt::print() is rather slow here, as it gets element by element from
53       * remote nodes. Use with care. */
54      if (mhp::rank() == 0)
55        fmt::print("{}\n", in);
56    }
57
58    mhp::finalize();
59
60    return 0;
61  }
```

Multi-process model (SPMD / MPI+SYCL)

How to get to internals? ”Segments” Example

- Create distributed structure
- Fill in parallel all parts
- Root rank **prints metadata of distribution – how many segments were created**
- Every rank prints its **local segment** size and data

```
5   #include <dr/mhp.hpp>
6   #include <fmt/core.h>
7
8   namespace mhp = dr::mhp;
9
10  int main(int argc, char **argv) {
11
12    mhp::init(sycl::default_selector_v);
13
14    fmt::print(
15        "Hello, World! Distributed ranges process is running on rank {} / {} on "
16        "host {}\n",
17        mhp::rank(), mhp::nprocs(), mhp::hostname());
18
19    std::size_t n = 100;
20
21    mhp::distributed_vector<int> v(n);
22    mhp::iota(v, 1);
23
24    if (mhp::rank() == 0) {
25      auto &&segments = v.segments();
26      fmt::print("Created distributed vector of size {} with {} segments.\n",
27                 v.size(), segments.size());
28    }
29
30    fmt::print("Rank {} owns segment of size {} and content {}\n", mhp::rank(),
31               mhp::local_segment(v).size(), mhp::local_segment(v));
32
33    mhp::finalize();
34
35    return 0;
36  }
```

Multi-process model (SPMD / MPI+SYCL)

More Multi-process / SPMD / MPI examples

Example 4: **2D Structures, adding matrices**
https://github.com/oneapi-src/distributed-ranges-tutorial/blob/main/src/example4.cpp

Example 5: **2D Stencil with halo exchange**
https://github.com/oneapi-src/distributed-ranges-tutorial/blob/main/src/example5.cpp

Example 6: **2D Pattern search**
https://github.com/oneapi-src/distributed-ranges-tutorial/blob/main/src/example6.cpp

Code (ready project with CMake build definition to clone, modify and run, easy starting point for your project)
https://github.com/oneapi-src/distributed-ranges-tutorial

Getting Started Guide
https://www.intel.com/content/www/us/en/developer/articles/guide/get-started-with-distributed-ranges.html
... or search „Distributed Ranges" in the web

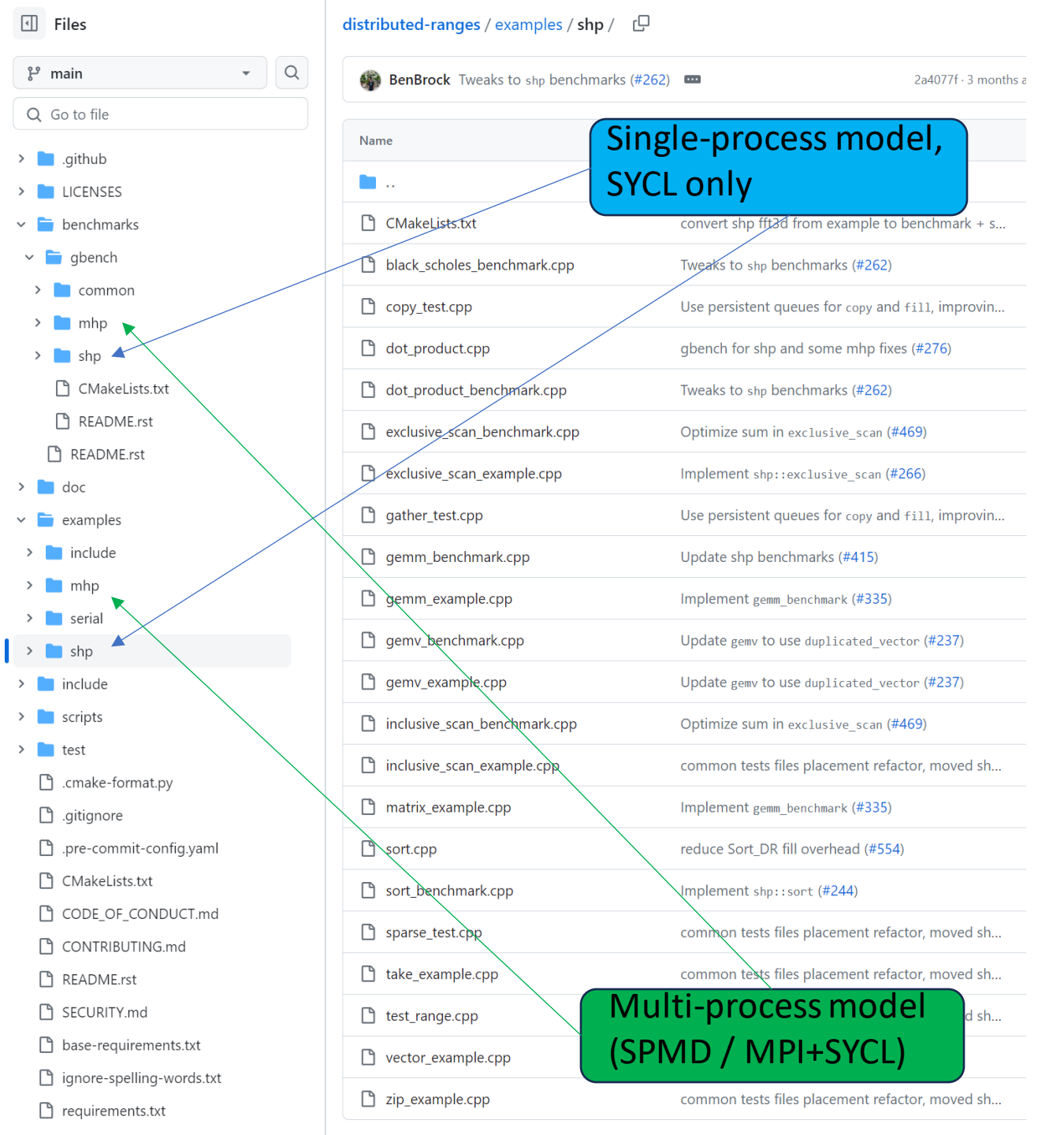
… and a lot more examples

Examples
https://github.com/oneapi-src/distributed-ranges/tree/main/examples/shp

https://github.com/oneapi-src/distributed-ranges/tree/main/examples/mhp

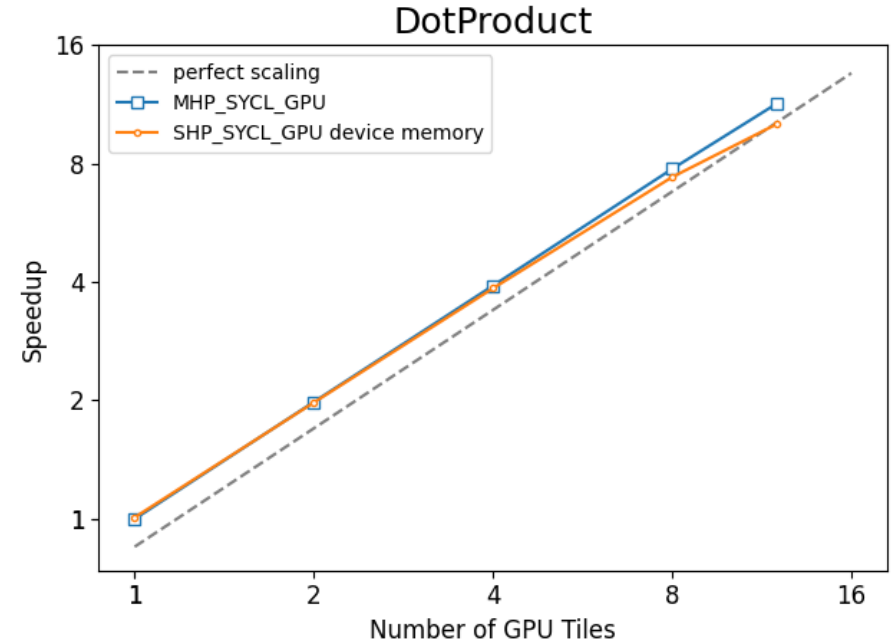
Benchmarks
https://github.com/oneapi-src/distributed-ranges/tree/main/benchmarks/gbench/shp

https://github.com/oneapi-src/distributed-ranges/tree/main/benchmarks/gbench/mhp

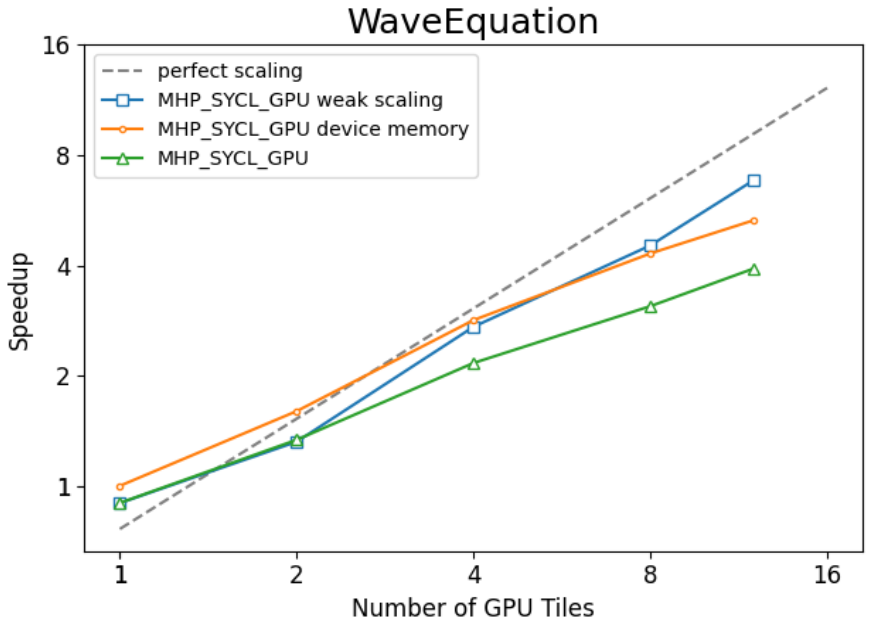


How does it scale?

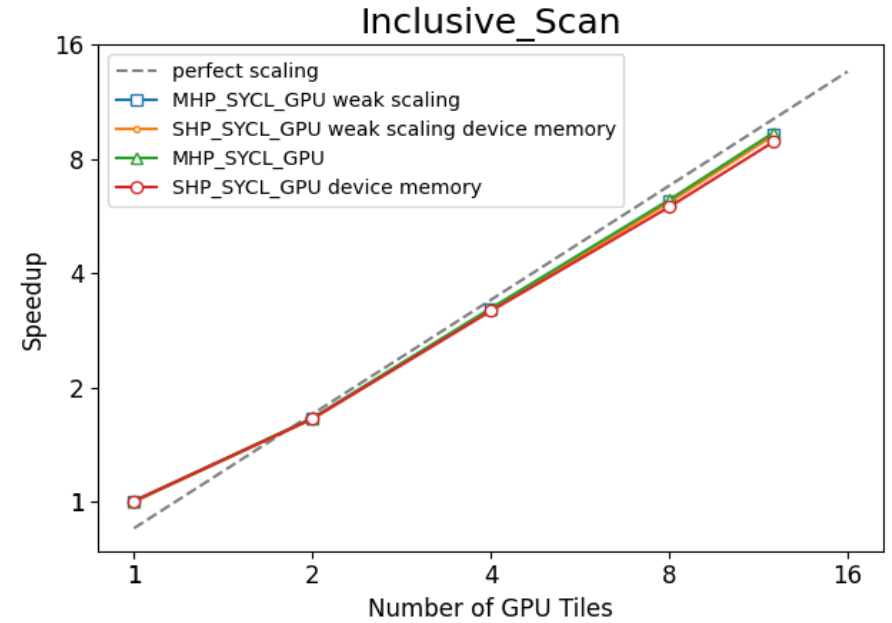
- Match roofline performance
- Weak scaling works well
- Depends on algorithm and its communication needs
- Some algorithms need more tuning (sort)



02/27/24 23:05:44



02/27/24 23:05:46

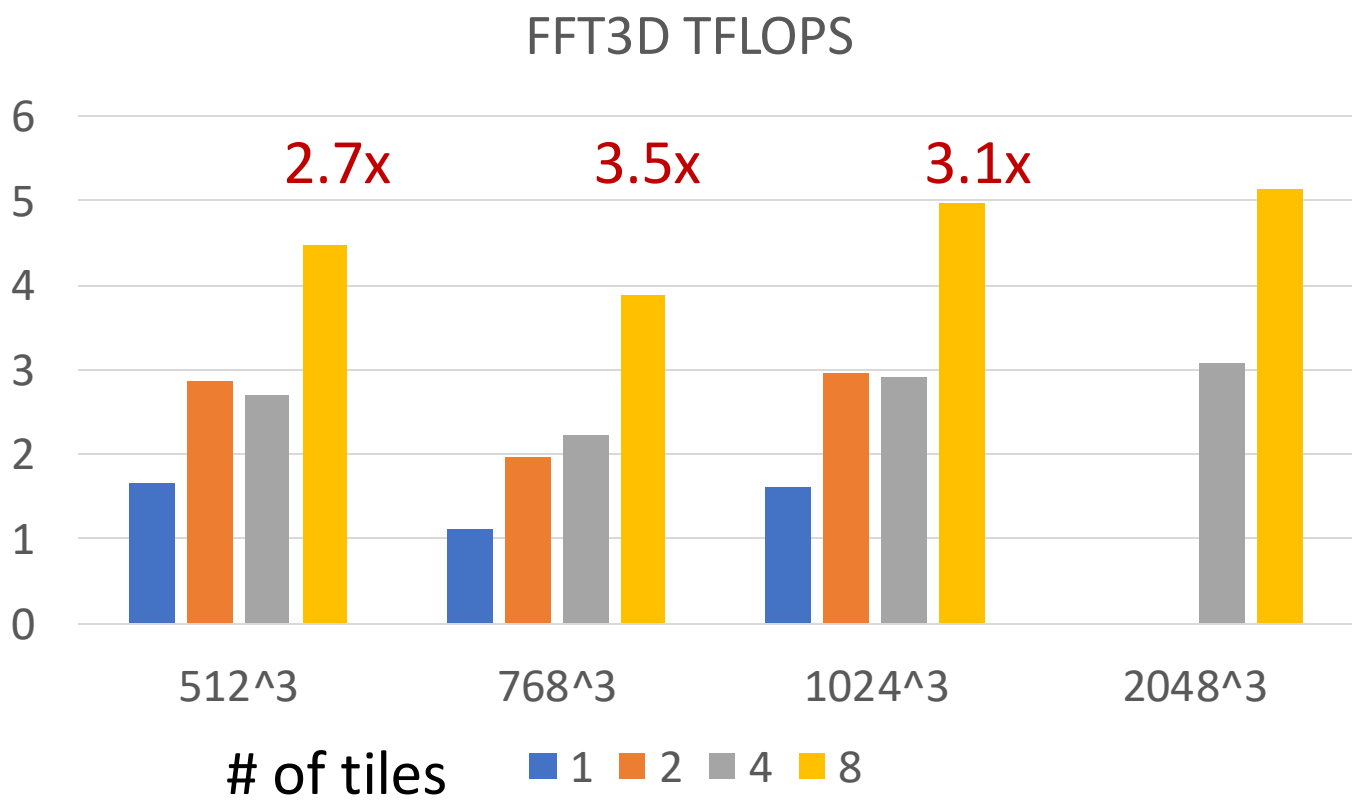


02/27/24 23:05:45

FFT3D/shp scaling on a 4-PVC node

- Slab distribution
- One queue per tile
- Transpose using load/store: same kernel on 1 and 8 tiles

- Performance impacted by Kernel launch overhead and serialization (SYCL global lock)
- Improvement
  - Host parallelization with OpenMP thread or std::thread
  - Smarter MKL kernel selection



Speedup of 8 tiles/ 1tile

Our offer to you

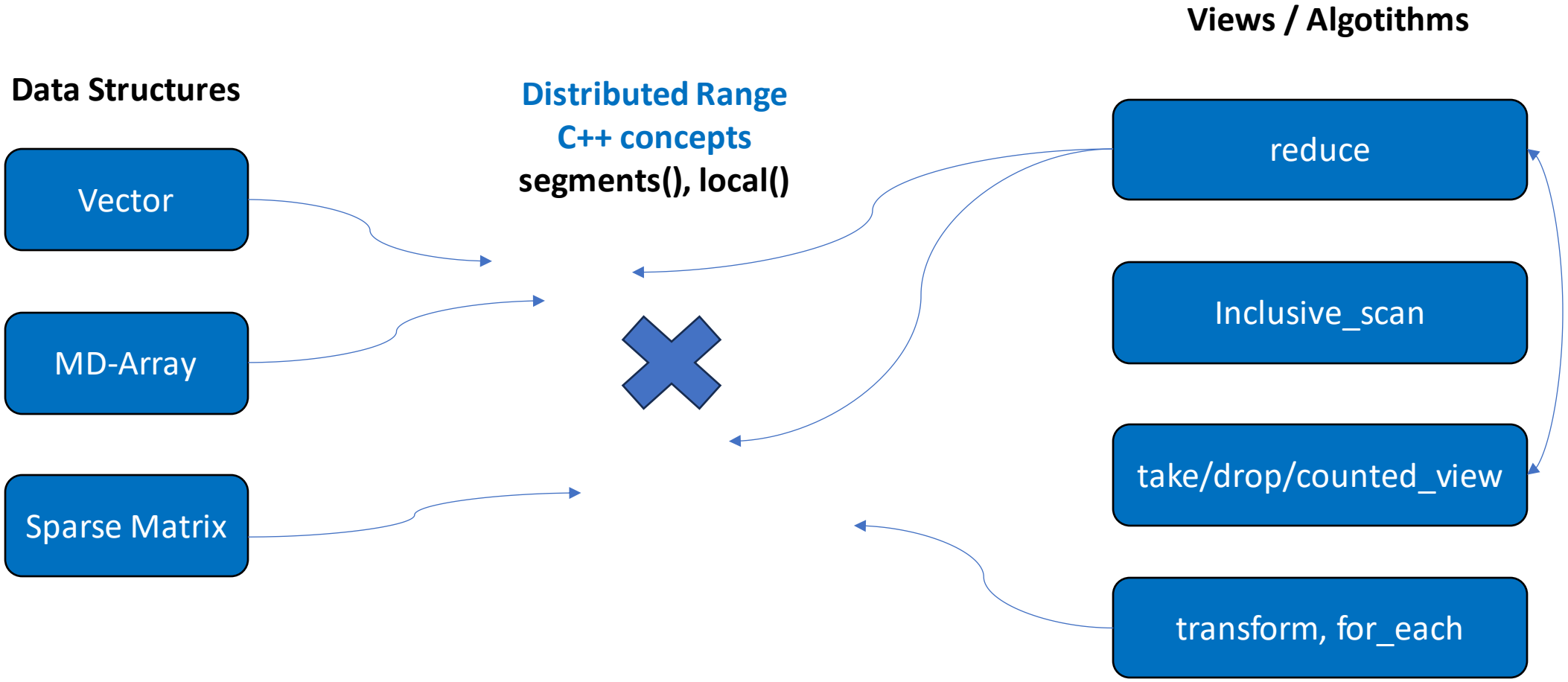
We want to collaborate on your project if you can use Distributed Ranges somehow.
We want to discover new ways of thinking, find new issues, do proof check of using advanced C++ for productivity.

1. Evaluate, use it, tell us how to expand it, make it more useful.
2. Use as **a reference implementation of using SYCL on multi-device** supernode.
3. Try it, and tell us **what** building block (algorithm, view, functionality) **you need** strenghtened in the library for your application...
4. ... if it only fits in the scope of the libary **we will deliver** and **tune** the performance for you.

https://github.com/oneapi-src/distributed-ranges
https://github.com/oneapi-src/distributed-ranges/issues

dds@intel.com

Internals


Views / Algotithms
Data Structures
Distributed Range
C++ concepts
segments(), local()
Vector
MD-Array
Sparse Matrix
reduce
Inclusive_scan
take/drop/counted_view
transform, for_each

Internals: write code in clean, novel, ranges-based way in C++20 standard

Ranges Library

C++ 20 added the **ranges library**

A **range** is a **collection of values**

Range concepts provide a **standard way** to iterate over values


```
// Iteration
for (auto&& value : range) {
  printf("%d\n", value);
}

// Algorithms
auto r = std::ranges::reduce(range);
auto r = std::ranges::partial_sum(range);

// Views
auto add_two = [](auto v) { return v + 2; };
auto view =
     std::ranges::transform_view(range, add_two);
```

Internals: write code in clean, novel, ranges-based way in C++20 standard

Distributed Range

A distributed range:

1) Is a **range** (satisfies `forward_range`)
2) Has **segments** (implements `segments` CPO)

```
template <typename R>
concept distributed_range = forward_range<R>
                         && requires(R &r) {
                              segments(r);
                            };
```

Segments returns a **range** of **remote ranges**.

This exposes **distribution** and **locality** of the distributed range.

Global View

Segments View

Internals: Complete Guide

See CppCon presentation to learn all details about concepts, remote pointers, iterator types, views, algorithms and data structures:
https://youtu.be/X_dlJcV21YI?si=3FR8ANSZasf5S4S3
https://github.com/CppCon/CppCon2023/blob/main/Presentations/Distributed_Ranges_CppCon23.pdf

Thank you!

Your **feedback** on how to **expand the DR library** for **your needs** and **use cases** is welcome.

https://github.com/oneapi-src/distributed-ranges

Contact us by form:
https://github.com/oneapi-src/distributed-ranges/issues

"Distributed Ranges: A Model for Distributed Data Structures, Algorithms, and Views"
Publication Accepted to ACM ICS'24