MULTI-DIMENSIONAL C++

Bryce Adelstein Lelbach

Principal Architect

Standard C++ Library Evolution Chair, US Programming Languages Chair

@blelbach

Today, C++ has no reasonable standard abstraction for multi-dimensional data.

Today, C++ has no reasonable standard abstraction for multi-dimensional data.

C++ needs abstractions that allow us to write generic multi-dimensional code.

We want to write generic multi-dimensional code that is:

- Storage agnostic.
- Rank agnostic.
- Layout agnostic.
- Iteration pattern agnostic.
- Composable.

Today, C++ has no reasonable standard abstraction for multi-dimensional data.

The solution is coming in C++23:

`std::mdspan`

`std::mdspan`

- Non-owning; pointer + metadata.

`std::mdspan`

- Non-owning; pointer + metadata.
- Metadata can be dynamic or static.

`std::mdspan`

- Non-owning; pointer + metadata.
- Metadata can be dynamic or static.
- Parameterizes layout.

`std::mdspan`

- Non-owning; pointer + metadata.
- Metadata can be dynamic or static.
- Parameterizes layout and access.

```
template <typename Index, Index... Extents>
class std::extents;
```

```cpp
template <typename Index, Index... Extents>
class std::extents;

std::extents e0{16, 32};
// Equivalent to:
std::extents<std::dynamic_extent, std::dynamic_extent> e1{16, 32};


e0.rank()     == 2
e0.extent(0) == 16
e0.extent(1) == 32
```

```cpp
template <typename Index, Index... Extents>
class std::extents;

std::extents e0{16, 32};
// Equivalent to:
std::extents<std::dynamic_extent, std::dynamic_extent> e1{16, 32};
std::dextents<2> e2{16, 32};

e0.rank()     == 2
e0.extent(0) == 16
e0.extent(1) == 32
```

```
template <typename Index, Index... Extents>
class std::extents;

std::extents e0{16, 32};
// Equivalent to:
std::extents<std::dynamic_extent, std::dynamic_extent> e1{16, 32};
std::dextents<2> e2{16, 32};

e0.rank()      == 2
e0.extent(0) == 16
e0.extent(1) == 32

std::extents<16, 32> e3;
```

```cpp
template <typename Index, Index... Extents>
class std::extents;

std::extents e0{16, 32};
// Equivalent to:
std::extents<std::dynamic_extent, std::dynamic_extent> e1{16, 32};
std::dextents<2> e2{16, 32};

e0.rank()      == 2
e0.extent(0) == 16
e0.extent(1) == 32

std::extents<16, 32> e3;

std::extents<16, std::dynamic_extent> e4{32};
```

```cpp
template <typename Index, Index... Extents>
class std::extents;

std::extents e0{16, 32};
// Equivalent to:
std::extents<std::dynamic_extent, std::dynamic_extent> e1{16, 32};
std::dextents<2> e2{16, 32};

e0.rank()      == 2
e0.extent(0) == 16
e0.extent(1) == 32

std::extents<16, 32> e3;

std::extents<16, std::dynamic_extent> e4{32};

std::extents e5{16, 32, 48, 4};
```

```
template <

                                                         >
class std::mdspan;
```

```
template <class T,

                                                  >
class std::mdspan;
```

```
template <class T,
          class Extents,


                                                           >
class std::mdspan;
```

```
template <class T,
          class Extents,
          class LayoutPolicy = std::layout_right,
                                                           >
class std::mdspan;
```

```
template <class T,
          class Extents,
          class LayoutPolicy = std::layout_right,
          class AccessorPolicy = std::default_accessor<T>>
class std::mdspan;
```

```
template <class T,
          class Extents,
          class LayoutPolicy = std::layout_right,
          class AccessorPolicy = std::default_accessor<T>>
class std::mdspan;

std::mdspan m0{data, 16, 32};
// Equivalent to:
std::mdspan<double, std::dextents<2>> m1{data, 16, 32};
```

```
template <class T,
          class Extents,
          class LayoutPolicy = std::layout_right,
          class AccessorPolicy = std::default_accessor<T>>
class std::mdspan;

std::mdspan m0{data, 16, 32};
// Equivalent to:
std::mdspan<double, std::dextents<2>> m1{data, 16, 32};

m0[i, j] == data[i * M + j]
```

```cpp
template <class T,
          class Extents,
          class LayoutPolicy = std::layout_right,
          class AccessorPolicy = std::default_accessor<T>>
class std::mdspan;

std::mdspan m0{data, 16, 32};
// Equivalent to:
std::mdspan<double, std::dextents<2>> m1{data, 16, 32};

m0[i, j] == data[i * M + j]

std::mdspan m2{data, std::extents<16, 32>{}};
// Equivalent to:
std::mdspan<double, std::extents<16, 32>> m3{data};

std::mdspan m4{data, std::extents<16, std::dynamic_extent>{32}};
```

Row-Major AKA Right

- C++, NumPy (default)
- Rightmost extent is contiguous

```
mdspan A{data, N, M};
mdspan A{data, layout_right::mapping{N, M}};

A[i, j]      == data[i * M + j]
A.stride(0) == M
A.stride(1) == 1
```

Row-Major AKA Right

- C++, NumPy (default)
- Rightmost extent is contiguous

```
mdspan A{data, N, M};
mdspan A{data, layout_right::mapping{N, M}};

A[i, j]      == data[i * M + j]
A.stride(0) == M
A.stride(1) == 1
```

Location	Element
0	a_{11}
1	a_{12}
2	a_{21}
3	a_{22}

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Row-Major AKA Right

- C++, NumPy (default)
- Rightmost extent is contiguous

```
mdspan A{data, N, M};
mdspan A{data, layout_right::mapping{N, M}};

A[i, j]      == data[i * M + j]
A.stride(0) == M
A.stride(1) == 1
```

Location	Element
0	a_{11}
1	a_{12}
2	a_{21}
3	a_{22}

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Column-Major AKA Left

- Fortran, MATLAB
- Leftmost extent is contiguous

```
mdspan B{data, layout_left::mapping{N, M}};

B[i, j]      == data[i + j * N]
B.stride(0) == 1
B.stride(1) == N
```

Row-Major AKA Right

- C++, NumPy (default)
- Rightmost extent is contiguous

```
mdspan A{data, N, M};
mdspan A{data, layout_right::mapping{N, M}};

A[i, j]      == data[i * M + j]
A.stride(0) == M
A.stride(1) == 1
```

Location	Element
0	a_{11}
1	a_{12}
2	a_{21}
3	a_{22}

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Column-Major AKA Left

- Fortran, MATLAB
- Leftmost extent is contiguous

```
mdspan B{data, layout_left::mapping{N, M}};

B[i, j]      == data[i + j * N]
B.stride(0) == 1
B.stride(1) == N
```

Location	Element
0	a_{11}
1	a_{21}
2	a_{12}
3	a_{22}

Row-Major AKA Right

- C++, NumPy (default)
- Rightmost extent is contiguous

```
mdspan A{data, N, M};
mdspan A{data, layout_right::mapping{N, M}};

A[i, j]      == data[i * M + j]
A.stride(0) == M
A.stride(1) == 1
```

Column-Major AKA Left

- Fortran, MATLAB
- Leftmost extent is contiguous

```
mdspan B{data, layout_left::mapping{N, M}};

B[i, j]      == data[i + j * N]
B.stride(0) == 1
B.stride(1) == N
```

User-Defined Strides

```
mdspan C{data, layout_stride::mapping{extents{N, M}, {X, Y}}};

A[i, j]      == data[i * X + j * Y]
A.stride(0) == X
A.stride(1) == Y
```

Layouts map $(i, j, k, \dots)$ to a storage location.

Layouts map $(i, j, k, \ldots)$ to a storage location.

Anyone can define a layout.

Layouts map $(i, j, k, \ldots)$ to a storage location.

Anyone can define a layout.

Layouts may:

- Be non-contiguous.

Layouts map $(i, j, k, \ldots)$ to a storage location.

Anyone can define a layout.

Layouts may:

- Be non-contiguous.
- Map multiple indices to the same location.

Layouts map $(i, j, k, \ldots)$ to a storage location.

Anyone can define a layout.

Layouts may:

- Be non-contiguous.
- Map multiple indices to the same location.
- Perform complicated computations.

Layouts map $(i, j, k, \ldots)$ to a storage location.

Anyone can define a layout.

Layouts may:

- Be non-contiguous.
- Map multiple indices to the same location.
- Perform complicated computations.
- Have or refer to state.

Parametric layouts enables generic multi-dimensional algorithms.

```
void your_function(Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic>& m);
```

```
void your_function(Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic>& m);

your_function(Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic>{…});
```

```
void your_function(Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic>& m);

your_function(Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic>{…});
your_function(boost::numeric::ublas::matrix<double>{…});
your_function(Mat{…}); // PETSc
your_function(blaze::DynamicMatrix<double, blaze::rowMajor>{…});
your_function(cutlass::HostTensor<float, cutlass::layout::ColumnMajor>{…});
// …
```

```
void your_function(std::mdspan<T, Extents, Layout, Accessor> m);

your_function(Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic>{…});
your_function(boost::numeric::ublas::matrix<double>{…});
your_function(Mat{…}); // PETSc
your_function(blaze::DynamicMatrix<double, blaze::rowMajor>{…});
your_function(cutlass::HostTensor<float, cutlass::layout::ColumnMajor>{…});
// …
```

```cpp
struct my_matrix {
public:
  my_matrix(std::size_t N, std::size_t M)
    : num_rows_(N), num_cols_(M), storage_(num_rows_ * num_cols_) {}

  double& operator()(size_t i, size_t j)
  { return storage_[i * num_cols_ + j]; }
  const double& operator()(size_t i, size_t j) const
  { return storage_[i * num_cols_ + j]; }

  std::size_t num_rows() const { return num_rows_; }
  std::size_t num_cols() const { return num_cols_; }




private:
  std::size_t         num_rows_, num_cols_;
  std::vector<double> storage_;
};
```

```
struct my_matrix {
public:
  my_matrix(std::size_t N, std::size_t M)
    : num_rows_(N), num_cols_(M), storage_(num_rows_ * num_cols_) {}

  double& operator()(size_t i, size_t j)
  { return storage_[i * num_cols_ + j]; }
  const double& operator()(size_t i, size_t j) const
  { return storage_[i * num_cols_ + j]; }

  std::size_t num_rows() const { return num_rows_; }
  std::size_t num_cols() const { return num_cols_; }

  operator std::mdspan<double, std::dextents<2>>() const
  { return {storage_, num_rows_, num_cols_}; }

private:
  std::size_t         num_rows_, num_cols_;
  std::vector<double> storage_;
};
```

```
std::span A{input,  N * M};
std::span B{output, M * N};

auto v = stdv::cartesian_product(
  stdv::iota(0, N),
  stdv::iota(0, M));

std::for_each(ex::par_unseq,
  begin(v), end(v),
  [=] (auto idx) {
    auto [i, j] = idx;
    B[i + j * N] = A[i * M + j];
  });
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};

auto v = stdv::cartesian_product(
  stdv::iota(0, A.extent(0)),
  stdv::iota(0, A.extent(1)));

std::for_each(ex::par_unseq,
  begin(v), end(v),
  [=] (auto idx) {
    auto [i, j] = idx;
    B[j, i] = A[i, j];
  });
```

```
std::mdspan A{input,  N, M, O};

std::mdspan B{output, N, M, O};



auto v = stdv::cartesian_product(
  stdv::iota(1, A.extent(0) - 1),
  stdv::iota(1, A.extent(1) - 1),
  stdv::iota(1, A.extent(2) - 1));

std::for_each(ex::par_unseq,
  begin(v), end(v),
  [=] (auto idx) {
    auto [i, j, k] = idx;
    B[i, j, k] =   ( A[i, j, k-1] +
                     A[i-1, j, k] +
      A[i, j-1, k] + A[i, j,   k] + A[i, j+1, k]
                   + A[i+1, j, k]
                   + A[i, j, k+1] ) / 7.0
  });
```

```
std::mdspan A{input,
               std::layout_left::mapping{N, M, O}};
std::mdspan B{output,
               std::layout_left::mapping{N, M, O}};


auto v = stdv::cartesian_product(
  stdv::iota(1, A.extent(0) - 1),
  stdv::iota(1, A.extent(1) - 1),
  stdv::iota(1, A.extent(2) - 1));

std::for_each(ex::par_unseq,
  begin(v), end(v),
  [=] (auto idx) {
    auto [i, j, k] = idx;
    B[i, j, k] =   ( A[i, j, k-1] +
                     A[i-1, j, k] +
      A[i, j-1, k] + A[i, j,   k] + A[i, j+1, k]
                   + A[i+1, j, k]
                   + A[i, j, k+1] ) / 7.0
  });
```

```
submdspan(mdspan<…> m, SliceSpecifiers... ss)
                      -> mdspan<…>
```

```
submdspan(mdspan<…> m, SliceSpecifiers... ss)
    -> mdspan<…>
```

Slice Specifier	Argument	Reduces Rank?
Single Index	`Integral`	☑

```
submdspan(mdspan<…> m, SliceSpecifiers... ss)
    -> mdspan<…>
```

Slice Specifier	Argument	Reduces Rank?
Single Index	Integral	☑
Range of Indices	std::pair<Integral, Integral> std::tuple<Integral, Integral>	☒

```
submdspan(mdspan<…> m, SliceSpecifiers... ss)
    -> mdspan<…>
```

Slice Specifier	Argument	Reduces Rank?
Single Index	Integral	☑
Range of Indices	std::pair<Integral, Integral> std::tuple<Integral, Integral>	☒
All Indices	std::full_extent	☒

```cpp
std::mdspan m0{data, 64, 128, 32};

auto m1 = std::submdspan(m0, std::tuple{15, 23},
                             std::tuple{31, 39},
                             std::tuple{ 7, 15});
```

```cpp
std::mdspan m0{data, 64, 128, 32};

auto m1 = std::submdspan(m0, std::tuple{15, 23},
                             std::tuple{31, 39},
                             std::tuple{ 7, 15});

m1.rank()    == 3
```

```
std::mdspan m0{data, 64, 128, 32};

auto m1 = std::submdspan(m0, std::tuple{15, 23},
                             std::tuple{31, 39},
                             std::tuple{ 7, 15});
m1.rank()     == 3
m1.extent(0) == 8
m1.extent(1) == 8
m1.extent(2) == 8
```

```
std::mdspan m0{data, 64, 128, 32};

auto m1 = std::submdspan(m0, std::tuple{15, 23},
                             std::tuple{31, 39},
                             std::tuple{ 7, 15});

m1.rank()     == 3
m1.extent(0)  == 8
m1.extent(1)  == 8
m1.extent(2)  == 8
m1[i, j, k]   == m0[i + 15, j + 31, k + 7]
```

```cpp
std::mdspan m0{data, 64, 128, 32};

auto m1 = std::submdspan(m0, std::tuple{15, 23},
                             std::tuple{31, 39},
                             std::tuple{ 7, 15});
m1.rank()     == 3
m1.extent(0) == 8
m1.extent(1) == 8
m1.extent(2) == 8
m1[i, j, k]  == m0[i + 15, j + 31, k + 7]

auto m2 = std::submdspan(m0, 15,
                             std::full_extent,
                             31);
```

```cpp
std::mdspan m0{data, 64, 128, 32};

auto m1 = std::submdspan(m0, std::tuple{15, 23},
                             std::tuple{31, 39},
                             std::tuple{ 7, 15});
m1.rank()     == 3
m1.extent(0) == 8
m1.extent(1) == 8
m1.extent(2) == 8
m1[i, j, k]  == m0[i + 15, j + 31, k + 7]

auto m2 = std::submdspan(m0, 15,
                             std::full_extent,
                             31);
m2.rank()     == 1
```

```cpp
std::mdspan m0{data, 64, 128, 32};

auto m1 = std::submdspan(m0, std::tuple{15, 23},
                             std::tuple{31, 39},
                             std::tuple{ 7, 15});
m1.rank()     == 3
m1.extent(0) == 8
m1.extent(1) == 8
m1.extent(2) == 8
m1[i, j, k]  == m0[i + 15, j + 31, k + 7]

auto m2 = std::submdspan(m0, 15,
                             std::full_extent,
                             31);
m2.rank()     == 1
m2.extent(0) == 128
```

```cpp
std::mdspan m0{data, 64, 128, 32};

auto m1 = std::submdspan(m0, std::tuple{15, 23},
                             std::tuple{31, 39},
                             std::tuple{ 7, 15});
m1.rank()     == 3
m1.extent(0)  == 8
m1.extent(1)  == 8
m1.extent(2)  == 8
m1[i, j, k]   == m0[i + 15, j + 31, k + 7]

auto m2 = std::submdspan(m0, 15,
                             std::full_extent,
                             31);
m2.rank()     == 1
m2.extent(0)  == 128
m2[j]         == m0[15, j, 31]
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

auto outer = stdv::cartesian_product(stdv::iota(0, (N + T - 1) / T),
                                     stdv::iota(0, (M + T - 1) / T));
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

auto outer = stdv::cartesian_product(stdv::iota(0, (N + T - 1) / T),
                                     stdv::iota(0, (M + T - 1) / T));

std::for_each(ex::par_unseq, begin(outer), end(outer),
  [=] (auto tile) {
    auto [x, y] = tile;
    …
  });
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

auto outer = stdv::cartesian_product(stdv::iota(0, (N + T - 1) / T),
                                     stdv::iota(0, (M + T - 1) / T));

std::for_each(ex::par_unseq, begin(outer), end(outer),
  [=] (auto tile) {
    auto [x, y] = tile;
    std::tuple selectN{T * x, std::min(T * (x + 1), N)};
    std::tuple selectM{T * y, std::min(T * (y + 1), M)};

    …
  });
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

auto outer = stdv::cartesian_product(stdv::iota(0, (N + T - 1) / T),
                                     stdv::iota(0, (M + T - 1) / T));

std::for_each(ex::par_unseq, begin(outer), end(outer),
  [=] (auto tile) {
    auto [x, y] = tile;
    std::tuple selectN{T * x, std::min(T * (x + 1), N)};
    std::tuple selectM{T * y, std::min(T * (y + 1), M)};

    auto TA = std::submdspan(A, selectN, selectM);
    auto TB = std::submdspan(B, selectM, selectN);

    …
  });
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

auto outer = stdv::cartesian_product(stdv::iota(0, (N + T - 1) / T),
                                     stdv::iota(0, (M + T - 1) / T));

std::for_each(ex::par_unseq, begin(outer), end(outer),
  [=] (auto tile) {
    auto [x, y] = tile;
    std::tuple selectN{T * x, std::min(T * (x + 1), N)};
    std::tuple selectM{T * y, std::min(T * (y + 1), M)};

    auto TA = std::submdspan(A, selectN, selectM);
    auto TB = std::submdspan(B, selectM, selectN);

    auto inner = stdv::cartesian_product(stdv::iota(0, TA.extent(0)),
                                         stdv::iota(0, TA.extent(1)));

    …
  });
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

auto outer = stdv::cartesian_product(stdv::iota(0, (N + T - 1) / T),
                                     stdv::iota(0, (M + T - 1) / T));

std::for_each(ex::par_unseq, begin(outer), end(outer),
  [=] (auto tile) {
    auto [x, y] = tile;
    std::tuple selectN{T * x, std::min(T * (x + 1), N)};
    std::tuple selectM{T * y, std::min(T * (y + 1), M)};

    auto TA = std::submdspan(A, selectN, selectM);
    auto TB = std::submdspan(B, selectM, selectN);

    auto inner = stdv::cartesian_product(stdv::iota(0, TA.extent(0)),
                                         stdv::iota(0, TA.extent(1)));

    for (auto [i, j] : inner)
      TB[j, i] = TA[i, j];
  });
```

Standard Algorithms

adjacent_difference	is_sorted[_until]	rotate[_copy]
adjacent_find	lexicographical_compare	search[_n]
all_of	max_element	set_difference
any_of	merge	set_intersection
copy[_if|_n]	min_element	set_symmetric_difference
count[_if]	minmax_element	set_union
equal	mismatch	sort
fill[_n]	move	stable_partition
find[_end|_first_of|_if|_if_not]	none_of	stable_sort
for_each	nth_element	swap_ranges
generate[_n]	partial_sort[_copy]	transform
includes	partition[_copy]	uninitialized_copy[_n]
inplace_merge	remove[_copy|_copy_if|_if]	uninitialized_fill[_n]
is_heap[_until]	replace[_copy|_copy_if|_if]	unique
is_partitioned	reverse[_copy]	unique_copy

```
std::mdspan A{…, N, M};
std::mdspan x{…, M};
std::mdspan y{…, N};

// y = 3.0 A x + 2.0 y
std::matrix_vector_product(
  ex::par_unseq,
  std::scaled(3.0, A), x,
  std::scaled(2.0, y), y);
```

```
std::mdspan A{…, N, M};
std::mdspan x{…, M};
std::mdspan y{…, N};

// y = 3.0 A x + 2.0 y
std::matrix_vector_product(
  ex::par_unseq,
  std::scaled(3.0, A), x,
  std::scaled(2.0, y), y);
```

```
std::mdspan A{…, N, M};
std::mdspan x{…, M};
std::mdspan y{…, N};

// y = 3.0 A x + 2.0 y
std::matrix_vector_product(
  ex::par_unseq,
  std::scaled(3.0, A), x,
  std::scaled(2.0, y), y);
```

```
std::mdspan A{…, N, M};
std::mdspan x{…, M};
std::mdspan b{…, N};

// Solve A x = b where A = U^T U

// Solve U^T c = b, using x to store c
std::triangular_matrix_vector_solve(ex::par_unseq,
                                    std::transposed(A),
                                    std::upper_triangle, std::explicit_diagonal,
                                    b, x);

// Solve U x = c, overwriting x with result
std::triangular_matrix_vector_solve(ex::par_unseq,
                                    A,
                                    std::upper_triangle, std::explicit_diagonal,
                                    x);
```

```
std::mdspan A{…, N, M};
std::mdspan x{…, M};
std::mdspan b{…, N};

// Solve A x = b where A = U^T U

// Solve U^T c = b, using x to store c
std::triangular_matrix_vector_solve(ex::par_unseq,
                                    std::transposed(A),
                                    std::upper_triangle, std::explicit_diagonal,
                                    b, x);

// Solve U x = c, overwriting x with result
std::triangular_matrix_vector_solve(ex::par_unseq,
                                    A,
                                    std::upper_triangle, std::explicit_diagonal,
                                    x);
```

`mdspan` doesn’t provide ranges and iterators that enumerate its elements.

`mdspan` doesn’t provide ranges and iterators that enumerate its elements.

Why not?

`mdspan` doesn’t provide ranges and iterators that enumerate its elements.

Why not?

Performance.

Why do we want ranges and iterators?

Why do we want ranges and iterators?

- Parameterization.

Why do we want ranges and iterators?

- Parameterization.
- Composability.

Why do we want ranges and iterators?

- Parameterization.
- Composability.

They enable generic programming.

```
void f(stdr::range auto r) {
  for (auto x : r)
    …
}
```

```
std::vector<T> v{…};

f(v);
```

```
void f(stdr::range auto r) {
  for (auto x : r)
    …
}
```

```cpp
std::vector<T> v{…};

f(v);

f(v | stdv::reverse);
```

```cpp
void f(stdr::range auto r) {
  for (auto x : r)
    …
}
```

```
void f(stdr::range auto r) {
  for (auto x : r)
    …
}
```

```
std::vector<T> v{…};

f(v);

f(v | stdv::reverse);

f(v | stdv::stride(2));
```

```cpp
void f(stdr::range auto r) {
  for (auto x : r)
    …
}
```

```cpp
std::vector<T> v{…};

f(v);

f(v | stdv::reverse);

f(v | stdv::stride(2));

f(v | stdv::reverse
    | stdv::stride(2));
```

```
void f(stdr::range auto r) {
  for (auto x : r)
    …
}
```

```
std::vector<T> v{…};

f(v);

f(v | stdv::reverse);

f(v | stdv::stride(2));

f(v | stdv::reverse
    | stdv::stride(2));

f(stdv::iota(0, N));
```

We want to write generic multi-dimensional code that is:

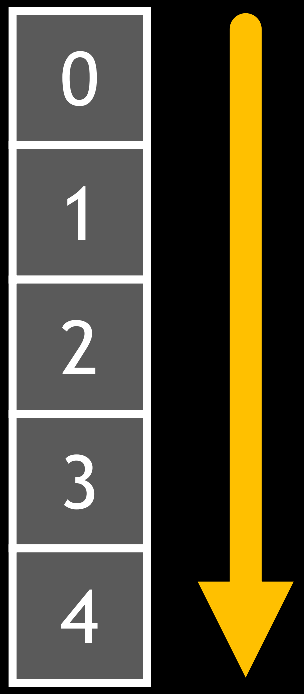
- Storage agnostic.
- Rank agnostic.
- Layout agnostic.
- Iteration pattern agnostic.
- Composable.

```
std::vector<T> v{…};
```

```
std::vector<T> v{…};
```

Forward

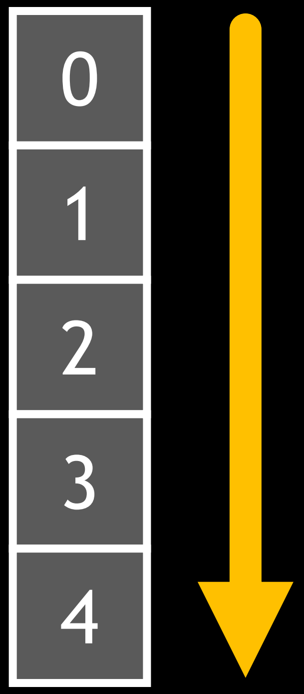
v



```
std::vector<T> v{…};
```

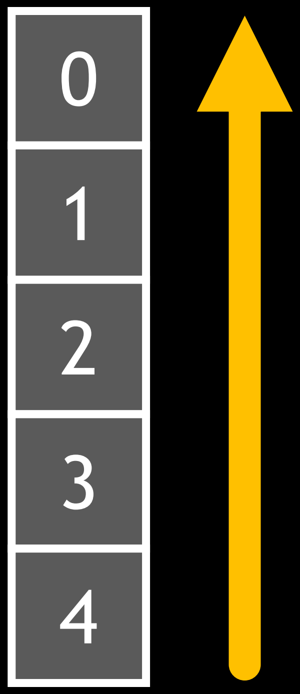
Forward

`v`



Reverse

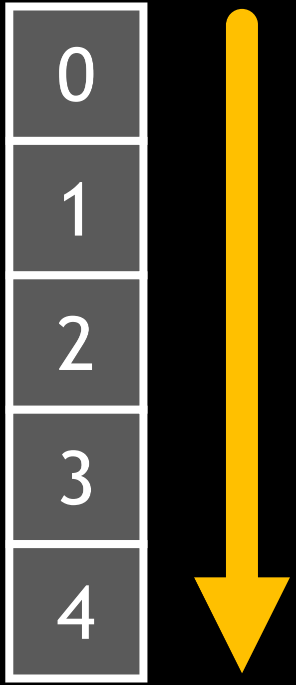
`v | reverse`



```
std::vector<T> v{…};
```

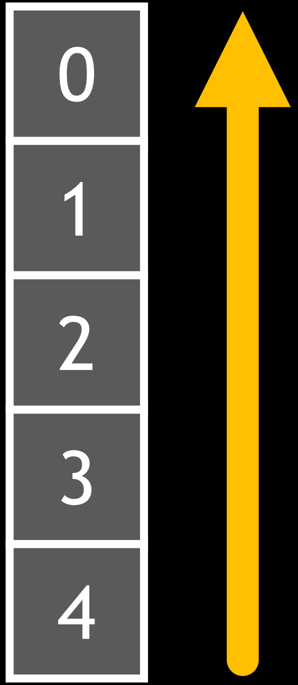
Forward

```
v
```



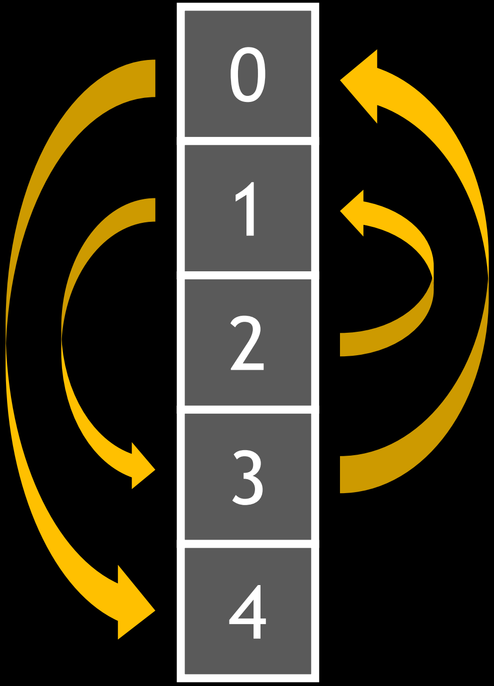
Reverse

```
v | reverse
```



Spiral

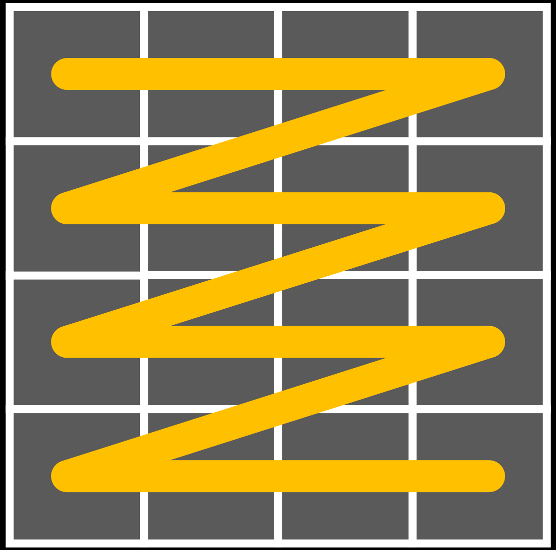
```
v | spiral(2)
```



```
std::mdspan A{…, N, M};
```

```
std::mdspan A{…, N, M};
```

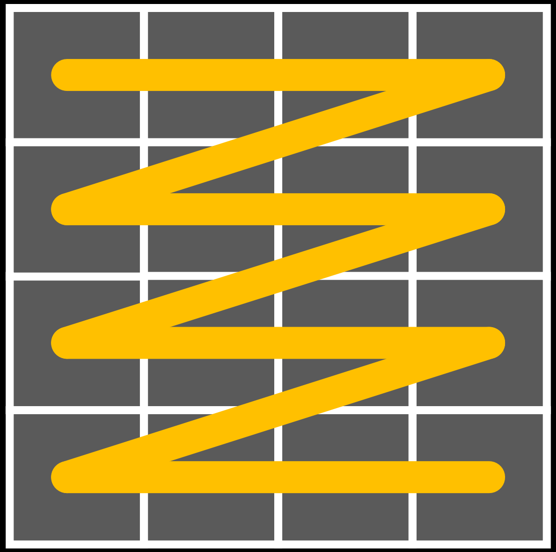
Row-Major
AKA Right



```
std::mdspan A{…, N, M};
```

Row-Major
AKA Right

Column-Major
AKA Left

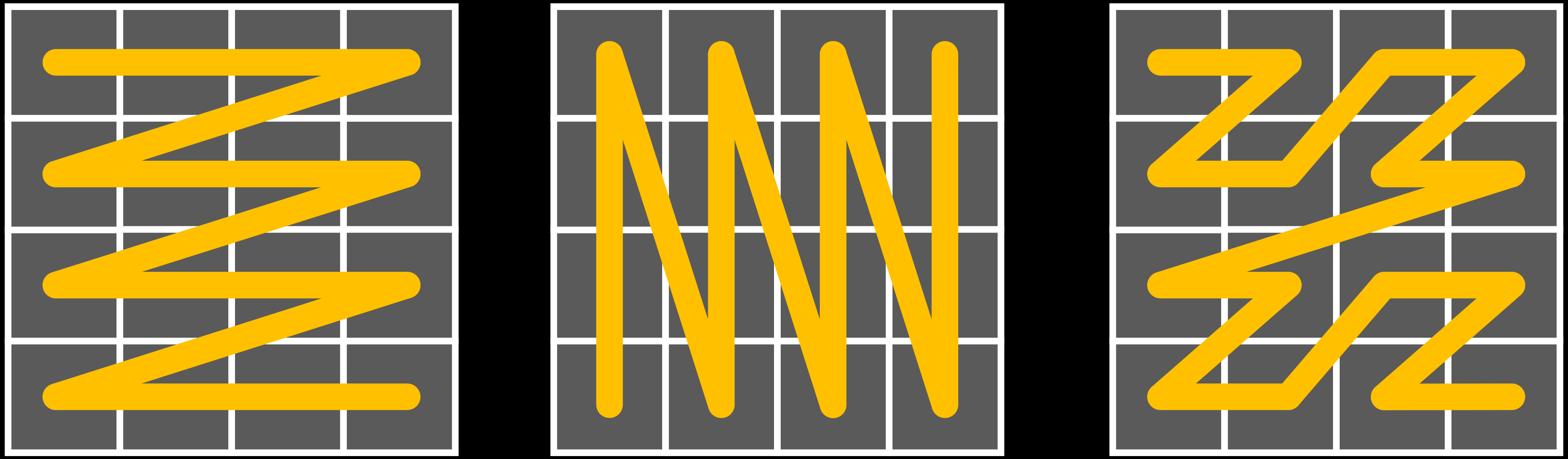


```
std::mdspan A{…, N, M};
```

Row-Major
AKA Right

Column-Major
AKA Left

Morton Order
AKA Z-Order Curve



A range is one way to iterate a sequence.

There may be other ways to iterate that sequence; those would be different ranges.

Indexwise

Elementwise

Indexwise

- Iterate through positions in MD space.
- Often uses multiple elements.
- Examples:
  - Matrix multiplication.
  - Sum rows/columns.
  - Stencils.

Elementwise

Indexwise

- Iterate through positions in MD space.
- Often uses multiple elements.
- Examples:
    - Matrix multiplication.
    - Sum rows/columns.
    - Stencils.

Elementwise

- Iterate through elements.
- Position in MD space doesn’t matter.
- Examples:
    - Multiply/add by scalar.
    - Count non-zeros.
    - Commutative reduction.

Multi-Dimensional Index Space
$(i, j, k, \ldots)$

Linear Storage Location Space
x

Multi-Dimensional Index Space

$(i, j, k, \ldots)$

MD to storage is typically fast, but loses information.

Linear Storage Location Space

x

MD to storage is typically fast, but loses information.

Multi-Dimensional Index Space

$(i, j, k, \ldots)$

Linear Storage Location Space

x

Storage to MD is usually slow, because information must be reconstructed.

C++ iterators operate in this space.

MD to storage is typically fast, but loses information.

Multi-Dimensional Index Space
$(i, j, k, \ldots)$

Linear Storage Location Space
x

Storage to MD is usually slow, because information must be reconstructed.

C++ iterators operate in this space.

MD to storage is typically fast, but loses information.

Multi-Dimensional Index Space
$(i, j, k, \dots)$

Linear Storage Location Space
x

Storage to MD is usually slow, because information must be reconstructed.

Maybe C++ MD iterators aren't such a good idea.

```
for (int64_t j = 0; j != M; ++j)
  for (int64_t i = 0; i != N; ++i)
    A[i + j * N] = 0.0F;
```

```cpp
for (int64_t j = 0; j != M; ++j)
  for (int64_t i = 0; i != N; ++i)
    A[i + j * N] = 0.0F;
```

```asm
        pushq   %r15
        pushq   %r14
        pushq   %r12
        pushq   %rbx
        testq   %rsi, %rsi
        je      .LBB0_12
        testq   %rdi, %rdi
        je      .LBB0_12
        leaq    -32(%rdi), %r9
        movq    %r9, %r8
        shrq    $5, %r8
        addq    $1, %r8
        movl    %r8d, %r10d
        andl    $3, %r10d
        andq    $-4, %r8
        leaq    960(%rdx), %rax
        leaq    (,%rdi,8), %r14
        leaq    192(%rdx), %r11
        xorl    %r15d, %r15d
        vxorps  %xmm0, %xmm0, %xmm0
        xorl    %r12d, %r12d
        jmp     .LBB0_3
```

Setup

```asm
.LBB0_11:
        addq    $1, %r12
        addq    %r14, %rax
        addq    %rdi, %r15
        addq    %r14, %rdx
        cmpq    %rsi, %r12
        je      .LBB0_12
.LBB0_3:
        testq   %rdi, %rdi
        movq    %rdx, %rcx
        movq    %rdi, %rbx
        je      .LBB0_13
        cmpq    $96, %r9
        jae     .LBB0_6
        xorl    %ebx, %ebx
        jmp     .LBB0_8
.LBB0_13:
        movq    $0, (%rcx)
        addq    $8, %rcx
        addq    $-1, %rbx
        jne     .LBB0_13
        jmp     .LBB0_11
```

Outer Loop (Scalar)

```asm
.LBB0_6:
        movq    %r8, %rcx
        xorl    %ebx, %ebx
.LBB0_7:
        vmovups %zmm0, -960(%rax,%rbx,8)
        vmovups %zmm0, -896(%rax,%rbx,8)
        vmovups %zmm0, -832(%rax,%rbx,8)
        vmovups %zmm0, -768(%rax,%rbx,8)
        vmovups %zmm0, -704(%rax,%rbx,8)
        vmovups %zmm0, -640(%rax,%rbx,8)
        vmovups %zmm0, -576(%rax,%rbx,8)
        vmovups %zmm0, -512(%rax,%rbx,8)
        vmovups %zmm0, -448(%rax,%rbx,8)
        vmovups %zmm0, -384(%rax,%rbx,8)
        vmovups %zmm0, -320(%rax,%rbx,8)
        vmovups %zmm0, -256(%rax,%rbx,8)
        vmovups %zmm0, -192(%rax,%rbx,8)
        vmovups %zmm0, -128(%rax,%rbx,8)
        vmovups %zmm0, -64(%rax,%rbx,8)
        vmovups %zmm0, (%rax,%rbx,8)
        subq    $-128, %rbx
        addq    $-4, %rcx
        jne     .LBB0_7
```

Primary Inner Loop (Vectorized)
Width: 16 x 512 bit = 8192 bit

```asm
.LBB0_8:
        testq   %r10, %r10
        je      .LBB0_11
        addq    %r15, %rbx
        leaq    (%r11,%rbx,8), %rcx
        movq    %r10, %rbx
.LBB0_10:
        vmovups %zmm0, -192(%rcx)
        vmovups %zmm0, -128(%rcx)
        vmovups %zmm0, -64(%rcx)
        vmovups %zmm0, (%rcx)
        addq    $256, %rcx
        addq    $-1, %rbx
        jne     .LBB0_10
        jmp     .LBB0_11
```

Remainder Inner Loop (Vectorized)
Width: 4 x 512 bit = 2048 bit

```asm
.LBB0_12:
        popq    %rbx
        popq    %r12
        popq    %r14
        popq    %r15
        retq
```

Cleanup

Peel Loop
Scalar.
Attempts to align and dispatch to a better loop.

Primary Loop
Vectorized.
Assumes certain alignment and # of items remaining.

Remainder Loop(s)
Scalar or vectorized.
May assume certain alignment and # of items remaining.

```
auto r = stdv::cartesian_product(
  stdv::iota(0, N),
  stdv::iota(0, M));
for (auto [i, j] : r)
  A[i, j] = 0.0F;
```

```cpp
auto r = stdv::cartesian_product(
  stdv::iota(0, N),
  stdv::iota(0, M));
for (auto [i, j] : r)
  A[i, j] = 0.0F;
```

```asm
        xorl    %r8d, %r8d
        testq   %rsi, %rsi
        movl    %edi, %ecx
        cmovnel %r8d, %ecx
        movslq  %ecx, %r10
        cmpq    %rdi, %r10
        je      .LBB4_3
        movq    (%rdx), %r9
        xorl    %edx, %edx
.LBB4_2:
        movslq  %edx, %rax
        movq    %rax, %rdx
        imulq   %rdi, %rdx
        addq    %r10, %rdx
        movq    $0, (%r9,%rdx,8)
        addl    $1, %eax
        movslq  %eax, %rdx
        xorl    %eax, %eax
        cmpq    %rsi, %rdx
        cmovel  %r8d, %edx
        sete    %al
        addl    %eax, %ecx
        movslq  %ecx, %r10
        cmpq    %rdi, %r10
        jne     .LBB4_2
.LBB4_3:
        retq
```

```cpp
struct index_2d_iterator { // Column-Major AKA Left
private:
  std::array<std::int64_t, 2> indices;
  std::array<std::int64_t, 2> extents;

public:
  index_2d_iterator(index_2d_iterator const&);
  index_2d_iterator& operator=(index_2d_iterator const&);
  bool operator<=>(index_2d_iterator const&) const;

  index_2d_iterator& operator++() {
    ++indices[0];                     // Inner loop iteration-expression.

    if (extents[0] == indices[0]) { // Inner loop condition.
      ++indices[1];                   // Outer loop increment.
      indices[0] = 0;                 // Inner loop init-statement.
    }
    return *this;
  }

  std::array<std::int64_t, 2> operator*() const { return indices; }
};
```


NVIDIA

```
for (auto [i, j] : index_2d_range(N, M))
  A[i, j] = 0.0F;
```

```cpp
for (auto [i, j] : index_2d_range(N, M))
  A[i, j] = 0.0F;
```

```asm
        movq    (%rdx), %r8
        xorl    %eax, %eax
        xorl    %edx, %edx
.LBB2_1:
        movq    %rax, %rcx
        imulq   %rdi, %rcx
        addq    %rdx, %rcx
        movq    $0, (%r8,%rcx,8)
        addq    $1, %rdx
        xorl    %r9d, %r9d
        cmpq    %rdi, %rdx
        sete    %r9b
        movl    $0, %ecx
        cmoveq  %rcx, %rdx
        addq    %r9, %rax
        cmpq    %rsi, %rax
        jne     .LBB2_1
        retq
```

```cpp
struct storage_2d_iterator { // Column-Major AKA Left
private:
  std::int64_t                location;
  std::array<std::int64_t, 2> extents;

public:
  storage_2d_iterator(storage_2d_iterator const&);
  storage_2d_iterator& operator=(storage_2d_iterator const&);
  bool operator<=>(storage_2d_iterator const&) const;

  storage_2d_iterator& operator++() {
    ++location;
    return *this;
  }

  std::array<std::int64_t, 2> operator*() const {
    return {location % extents[1], location / extents[1]};
  }
};
```

```
for (auto [i, j] : storage_2d_range(N, M))
  A[i, j] = 0.0F;
```

```cpp
for (auto [i, j] : storage_2d_range(N, M))
  A[i, j] = 0.0F;
```

```asm
        pushq   %rbp
        pushq   %r15
        pushq   %r14
        pushq   %r13
        pushq   %r12
        pushq   %rbx
        movq    %rsi, %rbx
        imulq   %rdi, %rbx
        testq   %rbx, %rbx
        je      .LBB5_3
        movq    (%rdx), %rax
        movq    %rax, -24(%rsp)
        vpbroadcastq    %rsi, %zmm0
        vpbroadcastq    %rdi, %zmm1
        addq    $-8, %rbx
        shrq    $3, %rbx
        addq    $1, %rbx
        vmovdqa64       .LCPI5_0(%rip), %zmm2
        vextracti32x4   $3, %zmm0, %xmm3
        vpextrq $1, %xmm3, -32(%rsp)
        vmovq   %xmm3, -40(%rsp)
        vextracti32x4   $2, %zmm0, %xmm3
        vpextrq $1, %xmm3, -48(%rsp)
        vmovq   %xmm3, -56(%rsp)
        vpsrlq  $32, %zmm0, %zmm12
        vpsrlq  $32, %zmm1, %zmm4
        vxorpd  %xmm9, %xmm9, %xmm9
        vpbroadcastq    .LCPI5_1(%rip), %zmm10
        vpbroadcastq    .LCPI5_2(%rip), %zmm11
        movq    -48(%rsp), %r14
.LBB5_2:
        vextracti32x4   $3, %zmm2, %xmm5
        vpextrq $1, %xmm5, %rax
        cqto
        movq    -32(%rsp), %r8
        idivq   %r9
        movq    %rax, -16(%rsp)
        vmovq   %xmm5, %rax
        cqto
        movq    -40(%rsp), %r9
        idivq   %r9
        movq    %rax, %rbp
        vextracti32x4   $2, %zmm2, %xmm5
        vpextrq $1, %xmm5, %rax
        cqto
        idivq   %r14
        movq    %rax, %r10
        vmovq   %xmm5, %rax
        cqto
        idivq   -56(%rsp)
        movq    %rax, %r11
        vextracti128    $1, %ymm2, %xmm5
        vpextrq $1, %xmm5, %rax
        vextracti128    $1, %ymm0, %xmm6
        vpextrq $1, %xmm6, %rcx
        movq    %rcx, -8(%rsp)
        cqto
        idivq   %rcx
        movq    %rax, %rsi
        vmovq   %xmm5, %rax
        vmovq   %xmm6, %r15
        cqto
        idivq   %r15
        movq    %rax, %rdi
        vpextrq $1, %xmm2, %rax
        vpextrq $1, %xmm0, %r12
        cqto
        idivq   %r12
        movq    %rax, %rcx
        vmovq   %xmm2, %rax
        vmovq   %xmm0, %r13
        cqto
        idivq   %r13
        vmovq   -16(%rsp), %xmm5
        vmovq   %rbp, %xmm6
        vpunpcklqdq     %xmm5, %xmm6, %xmm5
        vmovq   %r10, %xmm6
        vmovq   %r11, %xmm7
        vpunpcklqdq     %xmm6, %xmm7, %xmm6
        vinserti128     $1, %xmm5, %ymm6, %ymm5
        vmovq   %rsi, %xmm6
        vmovq   %rdi, %xmm7
        vpunpcklqdq     %xmm6, %xmm7, %xmm6
        vmovq   %rcx, %xmm7
        vmovq   %rax, %xmm3
        vpunpcklqdq     %xmm7, %xmm3, %xmm3
        vinserti128     $1, %xmm6, %ymm3, %ymm3
        vinserti64x4    $1, %ymm5, %zmm3, %zmm3
        vpmuludq        %zmm12, %zmm3, %zmm5
        vpsrlq  $32, %zmm3, %zmm6
        vpmuludq        %zmm0, %zmm6, %zmm7
        vpaddq  %zmm7, %zmm5, %zmm5
        vpmuludq        %zmm0, %zmm3, %zmm7
        vpmuludq        %zmm4, %zmm3, %zmm8
        vpmuludq        %zmm1, %zmm6, %zmm6
        vpsllq  $32, %zmm5, %zmm5
        vpaddq  %zmm6, %zmm8, %zmm6
        vpsllq  $32, %zmm6, %zmm6
        vpmuludq        %zmm1, %zmm3, %zmm3
        vpaddq  %zmm5, %zmm7, %zmm5
        vpaddq  %zmm6, %zmm3, %zmm3
        kxnorw  %k0, %k0, %k1
        vpaddq  %zmm10, %zmm2, %zmm8
        vpsubq  %zmm5, %zmm2, %zmm5
        vextracti32x4   $3, %zmm8, %xmm6
        vpextrq $1, %xmm6, %rax
        vpaddq  %zmm5, %zmm3, %zmm3
        movq    -24(%rsp), %rbp
        vscatterqpd     %zmm9, (%rbp,%zmm3,8) {%k1}
        cqto
        idivq   %r8
        movq    %rax, %r8
        vmovq   %xmm6, %rax
        cqto
        idivq   %r9
        movq    %rax, %r9
        vextracti32x4   $2, %zmm8, %xmm3
        vpextrq $1, %xmm3, %rax
        cqto
        idivq   %r14
        movq    %rax, %r10
        vmovq   %xmm3, %rax
        cqto
        idivq   -56(%rsp)
        movq    %rax, %r11
        vextracti128    $1, %ymm8, %xmm3
        vpextrq $1, %xmm3, %rax
        cqto
        idivq   -8(%rsp)
        movq    %rax, %rsi
        vmovq   %xmm3, %rax
        cqto
        idivq   %r15
        movq    %rax, %rdi
        vpextrq $1, %xmm8, %rax
        cqto
        idivq   %r12
        movq    %rax, %rcx
        vmovq   %xmm8, %rax
        cqto
        idivq   %r13
        vmovq   %r8, %xmm3
        vmovq   %r9, %xmm5
        vpunpcklqdq     %xmm3, %xmm5, %xmm3
        vmovq   %r10, %xmm5
        vmovq   %r11, %xmm6
        vpunpcklqdq     %xmm5, %xmm6, %xmm5
        vinserti128     $1, %xmm3, %ymm5, %ymm3
        vmovq   %rsi, %xmm5
        vmovq   %rdi, %xmm6
        vpunpcklqdq     %xmm5, %xmm6, %xmm5
        vmovq   %rcx, %xmm6
        vmovq   %rax, %xmm7
        vpunpcklqdq     %xmm6, %xmm7, %xmm6
        vinserti128     $1, %xmm5, %ymm6, %ymm5
        vinserti64x4    $1, %ymm3, %zmm5, %zmm3
        vpmuludq        %zmm12, %zmm3, %zmm5
        vpsrlq  $32, %zmm3, %zmm6
        vpmuludq        %zmm0, %zmm6, %zmm7
        vpaddq  %zmm7, %zmm5, %zmm5
        vpsllq  $32, %zmm5, %zmm5
        vpmuludq        %zmm0, %zmm3, %zmm7
        vpaddq  %zmm5, %zmm7, %zmm5
        vpmuludq        %zmm4, %zmm3, %zmm7
        vpmuludq        %zmm1, %zmm6, %zmm6
        vpaddq  %zmm6, %zmm7, %zmm6
        vpsubq  %zmm5, %zmm8, %zmm5
        vpsllq  $32, %zmm6, %zmm6
        vpmuludq        %zmm1, %zmm3, %zmm3
        vpaddq  %zmm6, %zmm3, %zmm3
        vpaddq  %zmm5, %zmm3, %zmm3
        kxnorw  %k0, %k0, %k1
        vpaddq  %zmm11, %zmm2, %zmm2
        addq    $-2, %rbx
        vscatterqpd     %zmm9, (%rbp,%zmm3,8) {%k1}
        jne     .LBB5_2
.LBB5_3:
        popq    %rbx
        popq    %r12
        popq    %r13
        popq    %r14
        popq    %r15
        popq    %rbp
        retq
```

```
vextracti32x4 $3, %zmm2, %xmm5
vpextrq $1, %xmm5, %rax
cqto
movq -32(%rsp), %r8
idivq %r9
movq %rax, -16(%rsp)
vmovq %xmm5, %rax
cqto
movq -40(%rsp), %r9
idivq %r9
movq %rax, %rbp
vextracti32x4 $2, %zmm2, %xmm5
vpextrq $1, %xmm5, %rax
cqto
idivq %r14
movq %rax, %r10
vmovq %xmm5, %rax
cqto
idivq -56(%rsp)
movq %rax, %r11
vextracti128 $1, %ymm2, %xmm5
vpextrq $1, %xmm5, %rax
vextracti128 $1, %ymm0, %xmm6
vpextrq $1, %xmm6, %rcx
movq %rcx, -8(%rsp)
cqto
idivq %rcx
movq %rax, %rsi
vmovq %xmm5, %rax
vmovq %xmm6, %r15
cqto
idivq %r15
movq %rax, %rdi
vpextrq $1, %xmm2, %rax
vpextrq $1, %xmm0, %r12
cqto
idivq %r12
movq %rax, %rcx
vmovq %xmm2, %rax
vmovq %xmm0, %r13
cqto
idivq %r13
vmovq -16(%rsp), %xmm5
vmovq %rbp, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vmovq %r10, %xmm6
vmovq %r11, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vinserti128 $1, %xmm5, %ymm6, %ymm5
vmovq %rsi, %xmm6
vmovq %rdi, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vmovq %rcx, %xmm7
vmovq %rax, %xmm3
vpunpcklqdq %xmm7, %xmm3, %xmm3
vinserti128 $1, %xmm6, %ymm3, %ymm3
vinserti64x4 $1, %ymm5, %zmm3, %zmm3
vpmuludq %zmm12, %zmm3, %zmm5
vpsrlq $32, %zmm3, %zmm6
vpmuludq %zmm0, %zmm6, %zmm7
vpaddq %zmm7, %zmm5, %zmm5
vpmuludq %zmm0, %zmm3, %zmm7
vpmuludq %zmm4, %zmm3, %zmm8
vpmuludq %zmm1, %zmm6, %zmm6
vpsllq $32, %zmm5, %zmm5
vpaddq %zmm6, %zmm8, %zmm6
vpsllq $32, %zmm6, %zmm6
vpmuludq %zmm1, %zmm3, %zmm3
vpaddq %zmm5, %zmm7, %zmm5
vpaddq %zmm6, %zmm3, %zmm3
kxnorw %k0, %k0, %k1
vpaddq %zmm10, %zmm2, %zmm8
vpsubq %zmm5, %zmm2, %zmm5
vextracti32x4 $3, %zmm8, %xmm6
vpextrq $1, %xmm6, %rax
vpaddq %zmm5, %zmm3, %zmm3
movq -24(%rsp), %rbp
vscatterqpd %zmm9, (%rbp,%zmm3,8) {%k1}
cqto
idivq %r8
movq %rax, %r8
vmovq %xmm6, %rax
cqto
idivq %r9
movq %rax, %r9
vextracti32x4 $2, %zmm8, %xmm3
vpextrq $1, %xmm3, %rax
cqto
idivq %r14
movq %rax, %r10
vmovq %xmm3, %rax
cqto
idivq -56(%rsp)
movq %rax, %r11
vextracti128 $1, %ymm8, %xmm3
vpextrq $1, %xmm3, %rax
cqto
idivq -8(%rsp)
movq %rax, %rsi
vmovq %xmm3, %rax
cqto
idivq %r15
movq %rax, %rdi
vpextrq $1, %xmm8, %rax
cqto
idivq %r12
movq %rax, %rcx
vmovq %xmm8, %rax
cqto
idivq %r13
vmovq %r8, %xmm3
vmovq %r9, %xmm5
vpunpcklqdq %xmm3, %xmm5, %xmm3
vmovq %r10, %xmm5
vmovq %r11, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vinserti128 $1, %xmm3, %ymm5, %ymm3
vmovq %rsi, %xmm5
vmovq %rdi, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vmovq %rcx, %xmm6
vmovq %rax, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vinserti128 $1, %xmm5, %ymm6, %ymm5
vinserti64x4 $1, %ymm3, %zmm5, %zmm3
vpmuludq %zmm12, %zmm3, %zmm5
vpsrlq $32, %zmm3, %zmm6
vpmuludq %zmm0, %zmm6, %zmm7
vpaddq %zmm7, %zmm5, %zmm5
vpsllq $32, %zmm5, %zmm5
vpmuludq %zmm0, %zmm3, %zmm7
vpaddq %zmm5, %zmm7, %zmm5
vpmuludq %zmm4, %zmm3, %zmm7
vpmuludq %zmm1, %zmm6, %zmm6
vpaddq %zmm6, %zmm7, %zmm6
vpsubq %zmm5, %zmm8, %zmm5
vpsllq $32, %zmm6, %zmm6
vpmuludq %zmm1, %zmm3, %zmm3
vpaddq %zmm6, %zmm3, %zmm3
vpaddq %zmm5, %zmm3, %zmm3
kxnorw %k0, %k0, %k1
vpaddq %zmm11, %zmm2, %zmm2
addq $-2, %rbx
vscatterqpd %zmm9, (%rbp,%zmm3,8) {%k1}
jne .LBB5_2
```

```
vextracti32x4 $3, %zmm2, %xmm5
vpextrq $1, %xmm5, %rax
cqto
movq -32(%rsp), %r8
idivq %r9
movq %rax, -16(%rsp)
vmovq %xmm5, %rax
cqto
movq -40(%rsp), %r9
idivq %r9
movq %rax, %rbp
vextracti32x4 $2, %zmm2, %xmm5
vpextrq $1, %xmm5, %rax
cqto
idivq %r14
movq %rax, %r10
vmovq %xmm5, %rax
cqto
idivq -56(%rsp)
movq %rax, %r11
vextracti128 $1, %ymm2, %xmm5
vpextrq $1, %xmm5, %rax
vextracti128 $1, %ymm0, %xmm6
vpextrq $1, %xmm6, %rcx
movq %rcx, -8(%rsp)
cqto
idivq %rcx
movq %rax, %rsi
vmovq %xmm5, %rax
vmovq %xmm6, %r15
cqto
idivq %r15
movq %rax, %rdi
vpextrq $1, %xmm2, %rax
vpextrq $1, %xmm0, %r12
cqto
idivq %r12
movq %rax, %rcx
vmovq %xmm2, %rax
vmovq %xmm0, %r13
cqto
idivq %r13
vmovq -16(%rsp), %xmm5
vmovq %rbp, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vmovq %r10, %xmm6
vmovq %r11, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vinserti128 $1, %xmm5, %ymm6, %ymm5
vmovq %rsi, %xmm6
vmovq %rdi, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vmovq %rcx, %xmm7
vmovq %rax, %xmm3
vpunpcklqdq %xmm7, %xmm3, %xmm3
vinserti128 $1, %xmm6, %ymm3, %ymm3
vinserti64x4 $1, %ymm5, %zmm3, %zmm3
vpmuludq %zmm12, %zmm3, %zmm5
vpsrlq $32, %zmm3, %zmm6
vpmuludq %zmm0, %zmm6, %zmm7
vpaddq %zmm7, %zmm5, %zmm5
vpmuludq %zmm0, %zmm3, %zmm7
vpmuludq %zmm4, %zmm3, %zmm8
vpmuludq %zmm1, %zmm6, %zmm6
vpsllq $32, %zmm5, %zmm5
vpaddq %zmm6, %zmm8, %zmm6
vpsllq $32, %zmm6, %zmm6
vpmuludq %zmm1, %zmm3, %zmm3
vpaddq %zmm5, %zmm7, %zmm5
vpaddq %zmm6, %zmm3, %zmm3
kxnorw %k0, %k0, %k1
vpaddq %zmm10, %zmm2, %zmm8
vpsubq %zmm5, %zmm2, %zmm5
vextracti32x4 $3, %zmm8, %xmm6
vpextrq $1, %xmm6, %rax
vpaddq %zmm5, %zmm3, %zmm3
movq -24(%rsp), %rbp
vscatterqpd %zmm9, (%rbp,%zmm3,8) {%k1}
cqto
idivq %r8
movq %rax, %r8
vmovq %xmm6, %rax
cqto
idivq %r9
movq %rax, %r9
vextracti32x4 $2, %zmm8, %xmm3
vpextrq $1, %xmm3, %rax
cqto
idivq %r14
movq %rax, %r10
vmovq %xmm3, %rax
cqto
idivq -56(%rsp)
movq %rax, %r11
vextracti128 $1, %ymm8, %xmm3
vpextrq $1, %xmm3, %rax
cqto
idivq -8(%rsp)
movq %rax, %rsi
vmovq %xmm3, %rax
cqto
idivq %r15
movq %rax, %rdi
vpextrq $1, %xmm8, %rax
cqto
idivq %r12
movq %rax, %rcx
vmovq %xmm8, %rax
cqto
idivq %r13
vmovq %r8, %xmm3
vmovq %r9, %xmm5
vpunpcklqdq %xmm3, %xmm5, %xmm3
vmovq %r10, %xmm5
vmovq %r11, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vinserti128 $1, %xmm3, %ymm5, %ymm3
vmovq %rsi, %xmm5
vmovq %rdi, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vmovq %rcx, %xmm6
vmovq %rax, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vinserti128 $1, %xmm5, %ymm6, %ymm5
vinserti64x4 $1, %ymm3, %zmm5, %zmm3
vpmuludq %zmm12, %zmm3, %zmm5
vpsrlq $32, %zmm3, %zmm6
vpmuludq %zmm0, %zmm6, %zmm7
vpaddq %zmm7, %zmm5, %zmm5
vpsllq $32, %zmm5, %zmm5
vpmuludq %zmm0, %zmm3, %zmm7
vpaddq %zmm5, %zmm7, %zmm5
vpmuludq %zmm4, %zmm3, %zmm7
vpmuludq %zmm1, %zmm6, %zmm6
vpaddq %zmm6, %zmm7, %zmm6
vpsubq %zmm5, %zmm8, %zmm5
vpsllq $32, %zmm6, %zmm6
vpmuludq %zmm1, %zmm3, %zmm3
vpaddq %zmm6, %zmm3, %zmm3
vpaddq %zmm5, %zmm3, %zmm3
kxnorw %k0, %k0, %k1
vpaddq %zmm11, %zmm2, %zmm2
addq $-2, %rbx
vscatterqpd %zmm9, (%rbp,%zmm3,8) {%k1}
jne .LBB5_2
```

```
vextracti32x4 $3, %zmm2, %xmm5
vpextrq $1, %xmm5, %rax
cqto
movq -32(%rsp), %r8
idivq %r9
movq %rax, -16(%rsp)
vmovq %xmm5, %rax
cqto
movq -40(%rsp), %r9
idivq %r9
movq %rax, %rbp
vextracti32x4 $2, %zmm2, %xmm5
vpextrq $1, %xmm5, %rax
cqto
idivq %r14
movq %rax, %r10
vmovq %xmm5, %rax
cqto
idivq -56(%rsp)
movq %rax, %r11
vextracti128 $1, %ymm2, %xmm5
vpextrq $1, %xmm5, %rax
vextracti128 $1, %ymm0, %xmm6
vpextrq $1, %xmm6, %rcx
movq %rcx, -8(%rsp)
cqto
idivq %rcx
movq %rax, %rsi
vmovq %xmm5, %rax
vmovq %xmm6, %r15
cqto
idivq %r15
movq %rax, %rdi
vpextrq $1, %xmm2, %rax
vpextrq $1, %xmm0, %r12
cqto
idivq %r12
movq %rax, %rcx
vmovq %xmm2, %rax
vmovq %xmm0, %r13
cqto
idivq %r13
vmovq -16(%rsp), %xmm5
vmovq %rbp, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vmovq %r10, %xmm6
vmovq %r11, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vinserti128 $1, %xmm5, %ymm6, %ymm5
vmovq %rsi, %xmm6
vmovq %rdi, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vmovq %rcx, %xmm7
vmovq %rax, %xmm3
vpunpcklqdq %xmm7, %xmm3, %xmm3
vinserti128 $1, %xmm6, %ymm3, %ymm3
vinserti64x4 $1, %ymm5, %zmm3, %zmm3
vpmuludq %zmm12, %zmm3, %zmm5
vpsrlq $32, %zmm3, %zmm6
vpmuludq %zmm0, %zmm6, %zmm7
vpaddq %zmm7, %zmm5, %zmm5
vpmuludq %zmm0, %zmm3, %zmm7
vpmuludq %zmm4, %zmm3, %zmm8
vpmuludq %zmm1, %zmm6, %zmm6
vpsllq $32, %zmm5, %zmm5
vpaddq %zmm6, %zmm8, %zmm6
vpsllq $32, %zmm6, %zmm6
vpmuludq %zmm1, %zmm3, %zmm3
vpaddq %zmm5, %zmm7, %zmm5
vpaddq %zmm6, %zmm3, %zmm3
kxnorw %k0, %k0, %k1
vpaddq %zmm10, %zmm2, %zmm8
vpsubq %zmm5, %zmm2, %zmm5
vextracti32x4 $3, %zmm8, %xmm6
vpextrq $1, %xmm6, %rax
vpaddq %zmm5, %zmm3, %zmm3
movq -24(%rsp), %rbp
vscatterqpd %zmm9, (%rbp,%zmm3,8) {%k1}
cqto
idivq %r8
movq %rax, %r8
vmovq %xmm6, %rax
cqto
idivq %r9
movq %rax, %r9
vextracti32x4 $2, %zmm8, %xmm3
vpextrq $1, %xmm3, %rax
cqto
idivq %r14
movq %rax, %r10
vmovq %xmm3, %rax
cqto
idivq -56(%rsp)
movq %rax, %r11
vextracti128 $1, %ymm8, %xmm3
vpextrq $1, %xmm3, %rax
cqto
idivq -8(%rsp)
movq %rax, %rsi
vmovq %xmm3, %rax
cqto
idivq %r15
movq %rax, %rdi
vpextrq $1, %xmm8, %rax
cqto
idivq %r12
movq %rax, %rcx
vmovq %xmm8, %rax
cqto
idivq %r13
vmovq %r8, %xmm3
vmovq %r9, %xmm5
vpunpcklqdq %xmm3, %xmm5, %xmm3
vmovq %r10, %xmm5
vmovq %r11, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vinserti128 $1, %xmm3, %ymm5, %ymm3
vmovq %rsi, %xmm5
vmovq %rdi, %xmm6
vpunpcklqdq %xmm5, %xmm6, %xmm5
vmovq %rcx, %xmm6
vmovq %rax, %xmm7
vpunpcklqdq %xmm6, %xmm7, %xmm6
vinserti128 $1, %xmm5, %ymm6, %ymm5
vinserti64x4 $1, %ymm3, %zmm5, %zmm3
vpmuludq %zmm12, %zmm3, %zmm5
vpsrlq $32, %zmm3, %zmm6
vpmuludq %zmm0, %zmm6, %zmm7
vpaddq %zmm7, %zmm5, %zmm5
vpsllq $32, %zmm5, %zmm5
vpmuludq %zmm0, %zmm3, %zmm7
vpaddq %zmm5, %zmm7, %zmm5
vpmuludq %zmm4, %zmm3, %zmm7
vpmuludq %zmm1, %zmm6, %zmm6
vpaddq %zmm6, %zmm7, %zmm6
vpsubq %zmm5, %zmm8, %zmm5
vpsllq $32, %zmm6, %zmm6
vpmuludq %zmm1, %zmm3, %zmm3
vpaddq %zmm6, %zmm3, %zmm3
vpaddq %zmm5, %zmm3, %zmm3
kxnorw %k0, %k0, %k1
vpaddq %zmm11, %zmm2, %zmm2
addq $-2, %rbx
vscatterqpd %zmm9, (%rbp,%zmm3,8) {%k1}
jne .LBB5_2
```

```
struct index_2d_generator;

index_2d_generator
generate_indices(std::int64_t N, std::int64_t M) {
  for (std::int64_t j = 0; j != M; ++j)
    for (std::int64_t i = 0; i != N; ++i)
      co_yield std::array{i, j};
}
```

```
for (auto [i, j] : generate_indices(N, M))
  A[i, j] = 0.0F;
```

```cpp
for (auto [i, j] : generate_indices(N, M))
  A[i, j] = 0.0F;
```

```asm
        movq    (%rdx), %r9
        addq    $56, %r9
        leaq    (,%rdi,8), %r8
        xorl    %edx, %edx
        vxorps  %xmm0, %xmm0, %xmm0
.LBB7_1:
        movq    %rdi, %rcx
        movq    %r9, %rax
.LBB7_2:
        vmovups %zmm0, -56(%rax)
        addq    $64, %rax
        addq    $-8, %rcx
        jne     .LBB7_2
        addq    $1, %rdx
        addq    %r8, %r9
        cmpq    %rsi, %rdx
        jne     .LBB7_1
        retq
```

Solutions

- Teach compilers to recognize non-loop control flow patterns and turn them into loops.

Solutions

- Teach compilers to recognize non-loop control flow patterns and turn them into loops.
- Develop compiler builtins that MD iterator authors can use to describe the inner loop.

Solutions

- Teach compilers to recognize non-loop control flow patterns and turn them into loops.
- Develop compiler builtins that MD iterator authors can use to describe the inner loop.
- Provide elementwise ranges and iterators that don’t expose MD position.

Solutions

- Teach compilers to recognize non-loop control flow patterns and turn them into loops.
- Develop compiler builtins that MD iterator authors can use to describe the inner loop.
- Provide elementwise ranges and iterators that don’t expose MD position.
- Don’t use ranges and iterators for MD; build new constructs instead.

The range-based `for` statement:

`for (`*for-range-declaration*`:` *for-range-initializer*`)` *statement*

is equivalent to:

```
{
  auto&& range = for-range-initializer;
  auto begin = stdr::begin(range);
  auto end = stdr::end(range);
  for (; begin != end; ++begin) {
    for-range-declaration = *begin;
    statement
  }
}
```

[stmt.ranged]

The range-based `for` statement:

`for` (*for-range-declaration*`:` *for-range-initializer*`)` *statement*

[stmt.ranged]

The Range Protocol

```
template <typename Range>
auto begin(Range&& range);

template <typename Range>
auto end(Range&& range);
```

The Space Protocol

```
template <std::int64_t N, typename Space, typename... Outer>
auto mdbegin(Space&& space, std::tuple<Outer...>&& outer);

template <std::int64_t N, typename Space, typename... Outer>
auto mdend(Space&& space, std::tuple<Outer...>&& outer);

template <typename Space>
constexpr std::int64_t mdrank;
```

The Space Protocol

```
template <std::int64_t N, typename Space, typename... Outer>
auto mdbegin(Space&& space, std::tuple<Outer...>&& outer);

template <std::int64_t N, typename Space, typename... Outer>
auto mdend(Space&& space, std::tuple<Outer...>&& outer);

template <typename Space>
constexpr std::int64_t mdrank;
```

The Space Protocol

```
template <std::int64_t N, typename Space, typename... Outer>
auto mdbegin(Space&& space, std::tuple<Outer...>&& outer);

template <std::int64_t N, typename Space, typename... Outer>
auto mdend(Space&& space, std::tuple<Outer...>&& outer);

template <typename Space>
constexpr std::int64_t mdrank;
```

The Space Protocol

```
template <std::int64_t N, typename Space, typename... Outer>
auto mdbegin(Space&& space, std::tuple<Outer...>&& outer);

template <std::int64_t N, typename Space, typename... Outer>
auto mdend(Space&& space, std::tuple<Outer...>&& outer);

template <typename Space>
constexpr std::int64_t mdrank;
```

The function:

```
template <typename Space, typename UnaryFunction>
void mdfor(Space&& space, UnaryFunction&& f);
```

is equivalent to:

```
constexpr auto N = mdrank<MDSpace>;
for (auto k = mdbegin<N - 1>(space); k != mdend<N - 1>(space); ++k)
  for (auto j = mdbegin<N – 2>(space, *k); j != mdend<N - 2>(space, *k); ++j)
    …
      for (auto i = mdbegin<0>(space, …); i != mdend<0>(space, …); ++i)
        f(*i);
```

The space-based `for` statement:

```
for (for-space-declaration : for-space-initializer) statement
```

is equivalent to:

```
auto&& space = for-space-initializer;
constexpr auto N = mdrank<MDSpace>;
for (auto k = mdbegin<N - 1>(space); k != mdend<N - 1>(space); ++k)
  for (auto j = mdbegin<N – 2>(space, *k); j != mdend<N - 2>(space, *k); ++j)
    …
      for (auto i = mdbegin<0>(space, …); i != mdend<0>(space, …); ++i)
        for-space-declaration = *i;
        statement
```

```
std::mdspan A{…, N, M, O};

// Just the main diagonal.
A.indices() | stdv::filter([] (auto [i, j, k]) { return i == j && j == k; });
```

```cpp
std::mdspan A{…, N, M, O};

// Just the main diagonal.
A.indices() | stdv::filter([] (auto [i, j, k]) { return i == j && j == k; });

// Just [i, 0, 0].
A.indices() | on_extent<1>(stdv::filter([] (auto [j, k]) { return j == 0; }))
            | on_extent<2>(stdv::filter([] (auto [k]) { return k == 0; }));
```

```
std::mdspan A{…, N, M, O};

// Just the main diagonal.
A.indices() | stdv::filter([] (auto [i, j, k]) { return i == j && j == k; });

// Just [i, 0, 0].
A.indices() | on_extent<1>(stdv::filter([] (auto [j, k]) { return j == 0; }))
            | on_extent<2>(stdv::filter([] (auto [k]) { return k == 0; }));

// Just interior points.
A.indices() | on_extent<0>(stdv::drop(1) | stdv::take(A.extent(0) - 2))
            | on_extent<1>(stdv::drop(1) | stdv::take(A.extent(1) - 2))
            | on_extent<2>(stdv::drop(1) | stdv::take(A.extent(2) - 2));
```

```
std::mdspan A{…, N, M};

// Traditional ranges & iterators for elementwise access.
// Multidimensional indices are NOT exposed from these.
stdr::random_access_range auto    range = A;
stdr::random_access_iterator auto first = A.begin();
stdr::random_access_iterator auto last  = A.end();

space auto indices = A.indices();
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};

auto v = stdv::cartesian_product(
  stdv::iota(0, A.extent(0)),
  stdv::iota(0, A.extent(1)));

std::for_each(ex::par_unseq,
  begin(v), end(v),
  [=] (auto idx) {
    auto [i, j] = idx;
    B[j, i] = A[i, j];
  });
```

```cpp
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};

stdr::for_each(
  ex::par_unseq,
  A.indices(),
  [=] (auto [i, j]) {
    B[j, i] = A[i, j];
  });
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};

ex::sender auto s =
    ex::transfer_just(sch, A.indices())
  | for_each_async(
      [=] (auto [i, j]) {
        B[j, i] = A[i, j];
      });
```

```
std::mdspan A{input,  N, M, O};
std::mdspan B{output, N, M, O};

auto v = stdv::cartesian_product(
  stdv::iota(1, A.extent(0) - 1),
  stdv::iota(1, A.extent(1) - 1),
  stdv::iota(1, A.extent(2) - 1));

std::for_each(ex::par_unseq,
  begin(v), end(v),
  [=] (auto idx) {
    auto [i, j, k] = idx;
    B[i, j, k] =   ( A[i, j, k-1] +
                     A[i-1, j, k] +
      A[i, j-1, k] + A[i, j,   k] + A[i, j+1, k]
                   + A[i+1, j, k]
                   + A[i, j, k+1] ) / 7.0
  });
```

```
std::mdspan A{input,  N, M, O};
std::mdspan B{output, N, M, O};

stdr::for_each(ex::par_unseq,
  A.indices(),
  [=] (auto [i, j, k]) {
    B[i, j, k] =   ( A[i, j, k-1] +
                     A[i-1, j, k] +
      A[i, j-1, k] + A[i, j,   k] + A[i, j+1, k]
                   + A[i+1, j, k]
                   + A[i, j, k+1] ) / 7.0
  });
```

```
std::mdspan A{input,  N, M, O};
std::mdspan B{output, N, M, O};

// Just interior points.
stdr::for_each(ex::par_unseq,
  A.indices() | on_extent<2>(stdv::drop(1) | stdv::take(A.extent(2)-2))
              | on_extent<1>(stdv::drop(1) | stdv::take(A.extent(1)-2))
              | on_extent<0>(stdv::drop(1) | stdv::take(A.extent(0)-2)),
  [=] (auto [i, j, k]) {
    B[i, j, k] =   ( A[i, j, k-1] +
                     A[i-1, j, k] +
      A[i, j-1, k] + A[i, j,   k] + A[i, j+1, k]
                   + A[i+1, j, k]
                   + A[i, j, k+1] ) / 7.0
  });
```

```
std::mdspan A{input,  N, M, O};
std::mdspan B{output, N, M, O};

// Just interior points.
for (auto [i, j, k] : A.indices()
                    | on_extent<0>(stdv::drop(1) | stdv::take(A.extent(0)-2))
                    | on_extent<1>(stdv::drop(1) | stdv::take(A.extent(1)-2))
                    | on_extent<2>(stdv::drop(1) | stdv::take(A.extent(2)-2)))
  B[i, j, k] =   ( A[i, j, k-1] +
                   A[i-1, j, k] +
    A[i, j-1, k] + A[i, j,   k] + A[i, j+1, k]
                 + A[i+1, j, k]
                 + A[i, j, k+1] ) / 7.0;
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

auto outer = stdv::cartesian_product(stdv::iota(0, (N + T - 1) / T),
                                     stdv::iota(0, (M + T - 1) / T));

std::for_each(ex::par_unseq, begin(outer), end(outer),
  [=] (auto tile) {
    auto [x, y] = tile;
    std::tuple selectN{T * x, std::min(T * (x + 1), N)};
    std::tuple selectM{T * y, std::min(T * (y + 1), M)};

    auto TA = std::submdspan(A, selectN, selectM);
    auto TB = std::submdspan(B, selectM, selectN);

    auto inner = stdv::cartesian_product(stdv::iota(0, TA.extent(0)),
                                         stdv::iota(0, TA.extent(1)));

    for (auto [i, j] : inner)
      TB[j, i] = TA[i, j];
  });
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

auto outer = A.indices() | on_extent<1>(stdv::stride(T))
                         | on_extent<0>(stdv::stride(T));

stdr::for_each(ex::par_unseq, outer,
  [=] (auto [x, y]) {
    for (auto [i, j] : mdspace{{T * x, std::min(T * (x + 1), N)},
                               {T * y, std::min(T * (y + 1), M)}})
      B[j, i] = A[i, j];
  });
```

```
std::mdspan A{input,  N, M};
std::mdspan B{output, M, N};
std::size_t T = ...;

for (auto [x, y] : A.indices() | on_extent<1>(stdv::stride(T))
                               | on_extent<0>(stdv::stride(T)))
  for (auto [i, j] : mdspace{{T * x, std::min(T * (x + 1), N)},
                             {T * y, std::min(T * (y + 1), M)}})
    B[j, i] = A[i, j];
```

@blelbach