Modern C++ in Computational Science

David S. Hollman, Mark Hoemmen, Daniel Sunderland, Christian R. Trott

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U. S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This document is SAND #2019-2042 C.

bit.ly/cpp-siamcse2019

Why should we use C++ for HPC?

Why should we use C++ for HPC?

- Decades ago, HPC hardware crossed the threshold where it was necessary to primarily use commodity parts

Why should we use C++ for HPC?

- Decades ago, HPC hardware crossed the threshold where it was necessary to primarily use commodity parts
- Similarly, we are long past the point where it makes sense to use non-commodity *languages*

Why should we use C++ for HPC?

- Decades ago, HPC hardware crossed the threshold where it was necessary to primarily use commodity parts
- Similarly, we are long past the point where it makes sense to use non-commodity *languages*
- C++ dominates the performance sensitive code at many companies with much larger budgets than we have (Google, Facebook, …)

Why should we use C++ for HPC?

- Decades ago, HPC hardware crossed the threshold where it was necessary to primarily use commodity parts
- Similarly, we are long past the point where it makes sense to use non-commodity *languages*
- C++ dominates the performance sensitive code at many companies with much larger budgets than we have (Google, Facebook, ...)
- By developing scientific applications in "commodity" languages like C++, we get:

Why should we use C++ for HPC?

- Decades ago, HPC hardware crossed the threshold where it was necessary to primarily use commodity parts
- Similarly, we are long past the point where it makes sense to use non-commodity *languages*
- C++ dominates the performance sensitive code at many companies with much larger budgets than we have (Google, Facebook, ...)
- By developing scientific applications in "commodity" languages like C++, we get:
  - Compiler development and optimizations from industry

Why should we use C++ for HPC?

- Decades ago, HPC hardware crossed the threshold where it was necessary to primarily use commodity parts
- Similarly, we are long past the point where it makes sense to use non-commodity *languages*
- C++ dominates the performance sensitive code at many companies with much larger budgets than we have (Google, Facebook, ...)
- By developing scientific applications in "commodity" languages like C++, we get:
  - Compiler development and optimizations from industry
  - Community knowledge and teaching materials

Why should we use C++ for HPC?

- Decades ago, HPC hardware crossed the threshold where it was necessary to primarily use commodity parts
- Similarly, we are long past the point where it makes sense to use non-commodity *languages*
- C++ dominates the performance sensitive code at many companies with much larger budgets than we have (Google, Facebook, ...)
- By developing scientific applications in "commodity" languages like C++, we get:
  - Compiler development and optimizations from industry
  - Community knowledge and teaching materials
  - The ability to hire "commodity" developers

Why should we use C++ for HPC?

- Decades ago, HPC hardware crossed the threshold where it was necessary to primarily use commodity parts
- Similarly, we are long past the point where it makes sense to use non-commodity *languages*
- C++ dominates the performance sensitive code at many companies with much larger budgets than we have (Google, Facebook, ...)
- By developing scientific applications in "commodity" languages like C++, we get:
  - Compiler development and optimizations from industry
  - Community knowledge and teaching materials
  - The ability to hire "commodity" developers
- Critically, though, we lose much of this if we do not keep our codebases up to date with modern C++ patterns, features, and idioms.

Disclaimer

There are a ton of things happening on the C++ committee that both directly and indirectly benefit computational science and HPC. (There are a ton of things happening in general—our 2018-10 mailing was larger than the entire works of Shakespeare.) Many of the "big ticket" items have *enormous* implications for HPC, but we will not be talking about them here. Many other small things other people are doing on the committee also have a big impact on HPC, and we don't want to diminish their contributions, but it's our talk and we're going to talk about the stuff we worked on

The "Big Ticket" Items: In Brief

The "Big Ticket" Items: In Brief

- For C++ 20, we got:

The "Big Ticket" Items: In Brief

- For C++ 20, we got:
  - Concepts

The "Big Ticket" Items: In Brief

- For C++ 20, we got:
  - Concepts
  - Contracts

The "Big Ticket" Items: In Brief

- For C++ 20, we got:
  - Concepts
  - Contracts
  - Coroutines

The "Big Ticket" Items: In Brief

- For C++ 20, we got:
  - Concepts
  - Contracts
  - Coroutines
  - Modules

The "Big Ticket" Items: In Brief

- For C++ 20, we got:
  - Concepts
  - Contracts
  - Coroutines
  - Modules
- We did not get (but coming in C++23):

The "Big Ticket" Items: In Brief

- For C++ 20, we got:
  - Concepts
  - Contracts
  - Coroutines
  - Modules
- We did not get (but coming in C++23):
  - Networking

The "Big Ticket" Items: In Brief

- For C++ 20, we got:
  - Concepts
  - Contracts
  - Coroutines
  - Modules
- We did not get (but coming in C++23):
  - Networking
  - Reflection

The "Big Ticket" Items: In Brief

- For C++ 20, we got:
  - Concepts
  - Contracts
  - Coroutines
  - Modules
- We did not get (but coming in C++23):
  - Networking
  - Reflection
  - Executors

`std::mdspan`

`std::mdspan`: Multidimensional Arrays in C++

ISO-C++ PROPOSAL

P0009
http://wg21.link/P0009r9

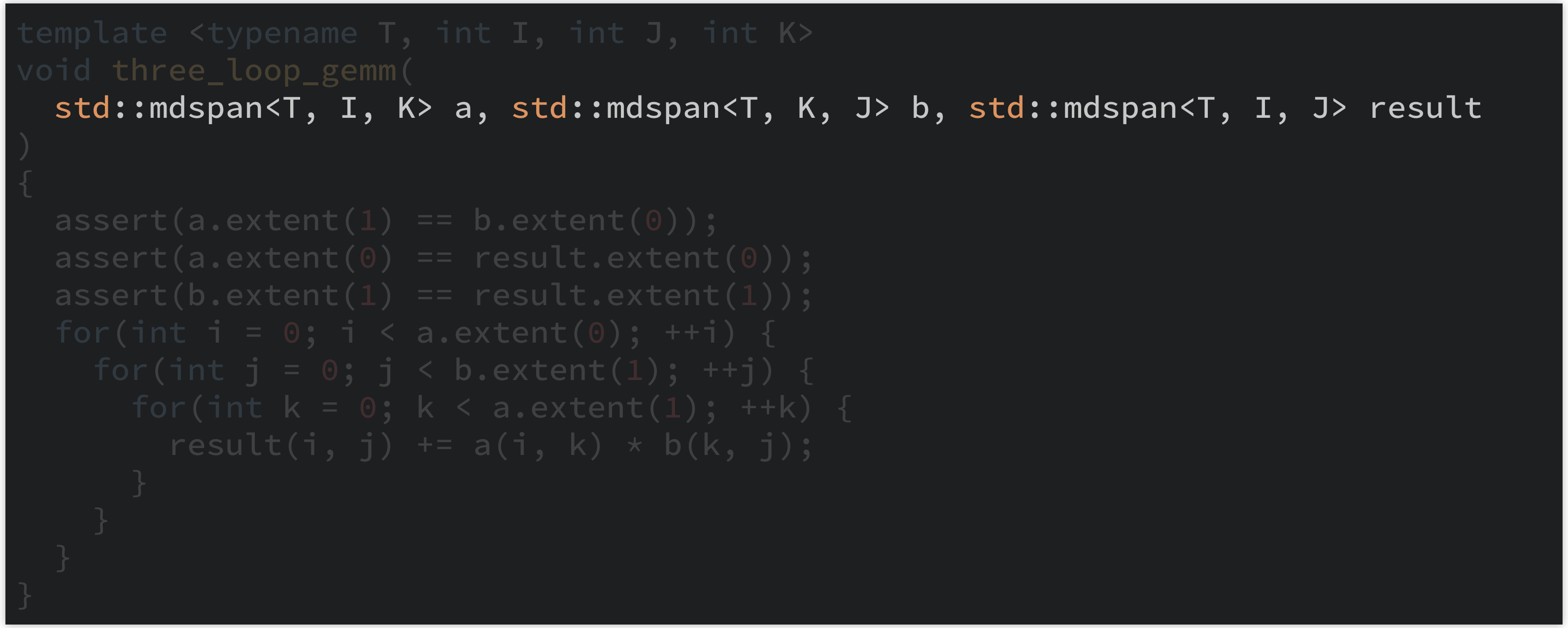
```
template <typename T, int I, int J, int K>
void three_loop_gemm(
  std::mdspan<T, I, K> a, std::mdspan<T, K, J> b, std::mdspan<T, I, J> result
)
{
  assert(a.extent(1) == b.extent(0));
  assert(a.extent(0) == result.extent(0));
  assert(b.extent(1) == result.extent(1));
  for(int i = 0; i < a.extent(0); ++i) {
    for(int j = 0; j < b.extent(1); ++j) {
      for(int k = 0; k < a.extent(1); ++k) {
        result(i, j) += a(i, k) * b(k, j);
      }
    }
  }
}
```

std::mdspan: Multidimensional Arrays in C++

ISO-C++ Proposal

P0009

http://wg21.link/P0009r9



```
template <typename T, int I, int J, int K>
void three_loop_gemm(
  std::mdspan<T, I, K> a, std::mdspan<T, K, J> b, std::mdspan<T, I, J> result
)
{
  assert(a.extent(1) == b.extent(0));
  assert(a.extent(0) == result.extent(0));
  assert(b.extent(1) == result.extent(1));
  for(int i = 0; i < a.extent(0); ++i) {
    for(int j = 0; j < b.extent(1); ++j) {
      for(int k = 0; k < a.extent(1); ++k) {
        result(i, j) += a(i, k) * b(k, j);
      }
    }
  }
}
```

`std::mdspan<T, I, K>` is a multidimensional view with extents `I` and `K` (both of which can be runtime-sized, using `std::dynamic_extent`)

`std::mdspan`: Multidimensional Arrays in C++

ISO-C++ PROPOSAL

P0009

http://wg21.link/P0009r9

```cpp
template <typename T, int I, int J, int K>
void three_loop_gemm(
  std::mdspan<T, I, K> a, std::mdspan<T, K, J> b, std::mdspan<T, I, J> result
)
{
  assert(a.extent(1) == b.extent(0));
  assert(a.extent(0) == result.extent(0));
  assert(b.extent(1) == result.extent(1));
  for(int i = 0; i < a.extent(0); ++i) {
    for(int j = 0; j < b.extent(1); ++j) {
      for(int k = 0; k < a.extent(1); ++k) {
        result(i, j) += a(i, k) * b(k, j);
      }
    }
  }
}
```

These assertions can be evaluated at compile time if the extents `I`, `J`, and `K` are static sizes

std::mdspan: Multidimensional Arrays in C++

ISO-C++ Proposal

P0009

http://wg21.link/P0009r9

```
template <typename T, int I, int J, int K>
void three_loop_gemm(
  std::mdspan<T, I, K> a, std::mdspan<T, K, J> b, std::mdspan<T, I, J> result
)
{
  assert(a.extent(1) == b.extent(0));
  assert(a.extent(0) == result.extent(0));
  assert(b.extent(1) == result.extent(1));
  for(int i = 0; i < a.extent(0); ++i) {
    for(int j = 0; j < b.extent(1); ++j) {
      for(int k = 0; k < a.extent(1); ++k) {
        result(i, j) += a(i, k) * b(k, j);
      }
    }
  }
}
```

Indexing uses the call operator for now.
(Work is in progress to also use the subscript operator `[]`, see https://wg21.link/p1161r2)

std::mdspan: Multidimensional Arrays in C++

ISO-C++ Proposal

P0009
http://wg21.link/P0009r9

```
template <typename T, int I, int J, int K>
void three_loop_gemm(
  std::mdspan<T, I, K> a, std::mdspan<T, K, J> b, std::mdspan<T, I, J> result
)
{
  assert(a.extent(1) == b.extent(0));
  assert(a.extent(0) == result.extent(0));
  assert(b.extent(1) == result.extent(1));
  for(int i = 0; i < a.extent(0); ++i) {
    for(int j = 0; j < b.extent(1); ++j) {
      for(int k = 0; k < a.extent(1); ++k) {
        result(i, j) += a(i, k) * b(k, j);
      }
    }
  }
}
```

std::mdspan Does Much More…

- mdspan is just an alias for basic_mdspan (just like string is an alias for basic_string):

```
template<class T, ptrdiff_t... Extents>
   using mdspan = basic_mdspan<T, extents<Extents...>>;
```

std::mdspan Does Much More…

- `mdspan` is just an alias for `basic_mdspan` (just like `string` is an alias for `basic_string`):

```
template<class T, ptrdiff_t... Extents>
   using mdspan = basic_mdspan<T, extents<Extents...>>;
```

- The full form is much more flexible and customizable:

std::mdspan Does Much More…

- `mdspan` is just an alias for `basic_mdspan` (just like `string` is an alias for `basic_string`):

```
template<class T, ptrdiff_t... Extents>
   using mdspan = basic_mdspan<T, extents<Extents...>>;
```

- The full form is much more flexible and customizable:

```
template<class ElementType,
         class Extents,
         class LayoutPolicy = layout_right,
         class AccessorPolicy = accessor_basic<ElementType>>
  class basic_mdspan;
```

std::mdspan Does Much More…

- mdspan is just an alias for basic_mdspan (just like string is an alias for basic_string):

```
template<class T, ptrdiff_t... Extents>
   using mdspan = basic_mdspan<T, extents<Extents...>>;
```

- The full form is much more flexible and customizable:

```
template<class ElementType,
         class Extents,
         class LayoutPolicy = layout_right,
         class AccessorPolicy = accessor_basic<ElementType>>
  class basic_mdspan;
```

ElementType is the element data type

`std::mdspan` Does Much More…

- `mdspan` is just an alias for `basic_mdspan` (just like `string` is an alias for `basic_string`):

```
template<class T, ptrdiff_t... Extents>
  using mdspan = basic_mdspan<T, extents<Extents...>>;
```

- The full form is much more flexible and customizable:

```
template<class ElementType,
         class Extents,
         class LayoutPolicy = layout_right,
         class AccessorPolicy = accessor_basic<ElementType>>
  class basic_mdspan;
```

`Extents` is an instance of a template `std::extents<...>` that contains the shape information.

std::mdspan Does Much More…

- `mdspan` is just an alias for `basic_mdspan` (just like `string` is an alias for `basic_string`):

```
template<class T, ptrdiff_t... Extents>
   using mdspan = basic_mdspan<T, extents<Extents...>>;
```

- The full form is much more flexible and customizable:

```
template<class ElementType,
         class Extents,
         class LayoutPolicy = layout_right,
         class AccessorPolicy = accessor_basic<ElementType>>
  class basic_mdspan;
```

`LayoutPolicy` is a customization point that lets you control how multi-indices are translated into memory offsets.

std::mdspan Does Much More…

- `mdspan` is just an alias for `basic_mdspan` (just like `string` is an alias for `basic_string`):

```
template<class T, ptrdiff_t... Extents>
  using mdspan = basic_mdspan<T, extents<Extents...>>;
```

- The full form is much more flexible and customizable:

```
template<class ElementType,
         class Extents,
         class LayoutPolicy = layout_right,
         class AccessorPolicy = accessor_basic<ElementType>>
  class basic_mdspan;
```

`AccessorPolicy` is a customization point that lets you control how memory offsets are translated into values, references, and pointers.

std::mdspan Does Much More…

- mdspan is just an alias for basic_mdspan (just like string is an alias for basic_string):

```
template<class T, ptrdiff_t... Extents>
   using mdspan = basic_mdspan<T, extents<Extents...>>;
```

- The full form is much more flexible and customizable:

```
template<class ElementType,
         class Extents,
         class LayoutPolicy = layout_right,
         class AccessorPolicy = accessor_basic<ElementType>>
  class basic_mdspan;
```

The `LayoutPolicy` Customization Point

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:
  - `layout_left` (FORTRAN ordering)

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:
  - `layout_left` (FORTRAN ordering)
  - `layout_right` (C ordering)

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:
  - `layout_left` (FORTRAN ordering)
  - `layout_right` (C ordering)
  - `layout_stride` (non-contiguous memory)

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:
  - `layout_left` (FORTRAN ordering)
  - `layout_right` (C ordering)
  - `layout_stride` (non-contiguous memory)
- The customization point is flexible enough to support things like

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:
  - `layout_left` (FORTRAN ordering)
  - `layout_right` (C ordering)
  - `layout_stride` (non-contiguous memory)
- The customization point is flexible enough to support things like
  - tiled layouts

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:
  - `layout_left` (FORTRAN ordering)
  - `layout_right` (C ordering)
  - `layout_stride` (non-contiguous memory)
- The customization point is flexible enough to support things like
  - tiled layouts
  - various forms of symmetric layouts

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:
  - `layout_left` (FORTRAN ordering)
  - `layout_right` (C ordering)
  - `layout_stride` (non-contiguous memory)
- The customization point is flexible enough to support things like
  - tiled layouts
  - various forms of symmetric layouts
  - sparse layouts

The `LayoutPolicy` Customization Point

- The proposal provides three layout policies:
  - `layout_left` (FORTRAN ordering)
  - `layout_right` (C ordering)
  - `layout_stride` (non-contiguous memory)
- The customization point is flexible enough to support things like
  - tiled layouts
  - various forms of symmetric layouts
  - sparse layouts
  - compressed layouts (with the help of an `AccessorPolicy`)

The `AccessorPolicy` Customization Point

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:
  - The reference type to be returned by `basic_mdspan::operator()`

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:
  - The reference type to be returned by `basic_mdspan::operator()`
  - The pointer type through which access occurs

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:
  - The reference type to be returned by `basic_mdspan::operator()`
  - The pointer type through which access occurs
  - A function for converting a pointer and an offset into a reference

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:
  - The reference type to be returned by `basic_mdspan::operator()`
  - The pointer type through which access occurs
  - A function for converting a pointer and an offset into a reference
- With these tools, you can write accessors that do things like:

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:
  - The reference type to be returned by `basic_mdspan::operator()`
  - The pointer type through which access occurs
  - A function for converting a pointer and an offset into a reference
- With these tools, you can write accessors that do things like:
  - Expose non-aliasing semantics (i.e., like `restrict` in C)

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:
  - The reference type to be returned by `basic_mdspan::operator()`
  - The pointer type through which access occurs
  - A function for converting a pointer and an offset into a reference
- With these tools, you can write accessors that do things like:
  - Expose non-aliasing semantics (i.e., like `restrict` in C)
  - Access remote memory

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:
  - The reference type to be returned by `basic_mdspan::operator()`
  - The pointer type through which access occurs
  - A function for converting a pointer and an offset into a reference
- With these tools, you can write accessors that do things like:
  - Expose non-aliasing semantics (i.e., like `restrict` in C)
  - Access remote memory
  - Access data stored in a compressed format of some sort

The `AccessorPolicy` Customization Point

- The `AccessorPolicy` customization point provides:
  - The reference type to be returned by `basic_mdspan::operator()`
  - The pointer type through which access occurs
  - A function for converting a pointer and an offset into a reference
- With these tools, you can write accessors that do things like:
  - Expose non-aliasing semantics (i.e., like `restrict` in C)
  - Access remote memory
  - Access data stored in a compressed format of some sort
  - Access data atomically (using P0019, `atomic_ref`!)

std::atomic_ref

Atomic Operations on Non-Atomic Memory

ISO-C++ PROPOSAL

P0019 http://wg21.link/P0019

```
std::vector<double> my_data;
/* ... */
// Before:
atomic_fetch_add(&my_data[i], 5.0);
// After:
auto a = atomic_ref{my_data[i]};
a += 5.0;
```

Atomic Operations on Non-Atomic Memory

```
template <class T>
void my_function(std::vector<T>& my_data, T value) {
  /* ... */
  // Before:
  ????????
  // After:
  auto a = atomic_ref{my_data[i]};
  a += value;
}
```

Executors

Executors: A Generic Abstraction for Execution

ISO-C++ Proposal

P0443:
https://wg21.link/p0443r10

(and many more…)

Executors: A Generic Abstraction for Execution

ISO-C++ Proposal

P0443:
https://wg21.link/p0443r10

(and many more...)

- Coming in C++23

Executors: A Generic Abstraction for Execution

ISO-C++ PROPOSAL

P0443:
https://wg21.link/p0443r10

(and many more...)

- Coming in C++23
  - (Why should you care about something coming that far away?)

Executors: A Generic Abstraction for Execution

ISO-C++ PROPOSAL

P0443:
https://wg21.link/p0443r10

(and many more…)

- Coming in C++23
  - (Why should you care about something coming that far away?)
- One of the most ambitious generic programming exercises ISO-C++ has ever undertaken

Executors: A Generic Abstraction for Execution

ISO-C++ Proposal

P0443:
https://wg21.link/p0443r10

(and many more...)

- Coming in C++23
  - (Why should you care about something coming that far away?)
- One of the most ambitious generic programming exercises ISO-C++ has ever undertaken
- Provides a generic abstraction for the execution model in the presence of a restricted programming model

Executor Example

```
template <class DataContainer>
DataContainer my_algorithm(DataContainer& data) {
  apply_transformation(data);
  DataContainer result = allocate_result_container_for(data);
  apply_reduction(data, result);
  return result;
}
```

Executor Example

```
template <class DataContainer>
DataContainer my_algorithm(DataContainer& data) {
  apply_transformation(data);
  DataContainer result = allocate_result_container_for(data);
  apply_reduction(data, result);
  return result;
}
```

Executor Example

```
template <class DataContainer>
DataContainer my_algorithm(DataContainer& data) {
  apply_transformation(data);
  DataContainer result = allocate_result_container_for(data);
  apply_reduction(data, result);
  return result;
}
```

Executor Example

```
template <class DataContainer>
DataContainer my_algorithm(DataContainer& data) {
  apply_transformation(data);
  DataContainer result = allocate_result_container_for(data);
  apply_reduction(data, result);
  return result;
}
```

Executor Example

```
template <class DataContainer>
DataContainer my_algorithm(DataContainer& data) {
  apply_transformation(data);
  DataContainer result = allocate_result_container_for(data);
  apply_reduction(data, result);
  return result;
}
```

Executor Example

```
template <class Executor, class DataContainer>
DataContainer my_algorithm(Executor ex, DataContainer& data) {
  apply_transformation(ex, data);
  DataContainer result = allocate_result_container_for(ex, data);
  apply_reduction(ex, data, result);
  return result;
}
```

Executor Example

```
template <class Executor, class DataContainer>
DataContainer my_algorithm(Executor ex, DataContainer& data) {
  apply_transformation(ex, data);
  DataContainer result = allocate_result_container_for(ex, data);
  apply_reduction(ex, data, result);
  return result;
}
```

Executor Example

```
template <class Executor, class DataContainer>
DataContainer my_algorithm(Executor ex, DataContainer& data) {
  apply_transformation(ex, data);
  DataContainer result = allocate_result_container_for(ex, data);
  apply_reduction(ex, data, result);
  return result;
}
```

Executor Example

```
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  if(data.size() > THRESHOLD) {
    auto gpu_executor = get_nearest_gpu_executor(network_executor);
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    // Otherwise, do it in place
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

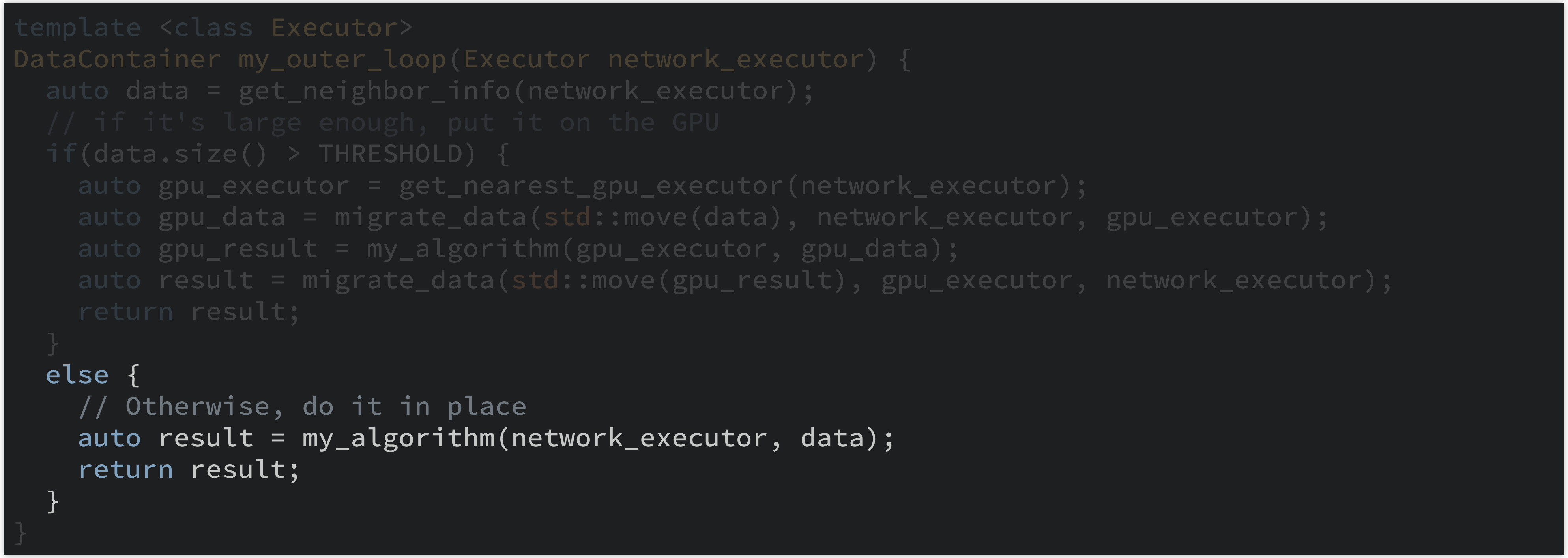
Executor Example

```
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  if(data.size() > THRESHOLD) {
    auto gpu_executor = get_nearest_gpu_executor(network_executor);
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    // Otherwise, do it in place
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example

```
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  if(data.size() > THRESHOLD) {
    auto gpu_executor = get_nearest_gpu_executor(network_executor);
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    // Otherwise, do it in place
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example



```
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  if(data.size() > THRESHOLD) {
    auto gpu_executor = get_nearest_gpu_executor(network_executor);
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    // Otherwise, do it in place
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example

```
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  if(data.size() > THRESHOLD) {
    auto gpu_executor = get_nearest_gpu_executor(network_executor);
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    // Otherwise, do it in place
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

But this isn't very generic...

Executor Example

```cpp
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  if(data.size() > THRESHOLD) {
    auto gpu_executor = get_nearest_gpu_executor(network_executor);
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    // Otherwise, do it in place
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example

```cpp
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  auto gpu_executor = get_nearest_gpu_executor(network_executor);
  auto threshold = std::query(gpu_executor, transfer_threshold(data, network_executor));
  if(data.size() > threshold) {
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example

```
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  auto gpu_executor = get_nearest_gpu_executor(network_executor);
  auto threshold = std::query(gpu_executor, transfer_threshold(data, network_executor));
  if(data.size() > threshold) {
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Solution: Put the customization on the executor!

Executor Example

```
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  auto gpu_executor = get_nearest_gpu_executor(network_executor);
  auto threshold = std::query(gpu_executor, transfer_threshold(data, network_executor));
  if(data.size() > threshold) {
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example

```
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  auto gpu_executor = get_nearest_gpu_executor(network_executor);
  auto cost_model = std::query(my_algorithm, cost_model(network_executor, gpu_executor));
  auto should_transfer = std::query(cost_model, transfer_recommendation(data));
  if(should_transfer) {
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example

```cpp
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  auto gpu_executor = get_nearest_gpu_executor(network_executor);
  auto cost_model = std::query(my_algorithm, cost_model(network_executor, gpu_executor));
  auto should_transfer = std::query(cost_model, transfer_recommendation(data));
  if(should_transfer) {
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Even better: ask the algorithm!

Executor Example

```cpp
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  auto gpu_executor = get_nearest_gpu_executor(network_executor);
  auto cost_model = std::query(my_algorithm, cost_model(network_executor, gpu_executor));
  auto should_transfer = std::query(cost_model, transfer_recommendation(data));
  if(should_transfer) {
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example

```cpp
template <class Executor>
DataContainer my_outer_loop(Executor network_executor) {
  auto data = get_neighbor_info(network_executor);
  // if it's large enough, put it on the GPU
  auto gpu_executor = get_nearest_gpu_executor(network_executor);
  auto cost_model = std::query(my_algorithm, cost_model(network_executor, gpu_executor));
  auto should_transfer = std::query(cost_model, transfer_recommendation(data));
  if(should_transfer) {
    auto gpu_data = migrate_data(std::move(data), network_executor, gpu_executor);
    auto gpu_result = my_algorithm(gpu_executor, gpu_data);
    auto result = migrate_data(std::move(gpu_result), gpu_executor, network_executor);
    return result;
  }
  else {
    auto result = my_algorithm(network_executor, data);
    return result;
  }
}
```

Executor Example

What happens if I have a network-capable GPU direct executor?

```
void my_program() {
  auto gpu_direct_executor = /*...*/;
  // ...
  for(auto iter : my_iterations) {
    // ...
    auto result = my_outer_loop(gpu_direct_executor);
    // ...
  }
}
```

Executor Example

What happens if I have a network-capable GPU direct executor?

```
void my_program() {
  auto gpu_direct_executor = /*...*/;
  // ...
  for(auto iter : my_iterations) {
    // ...
    auto result = my_outer_loop(gpu_direct_executor);
    // ...
  }
}
```

Executor Example

What happens if I have a network-capable GPU direct executor?

```
void my_program() {
  auto gpu_direct_executor = /*...*/;
  // ...
  for(auto iter : my_iterations) {
    // ...
    auto result = my_outer_loop(gpu_direct_executor);
    // ...
  }
}
```

How does this change how you should write code?

How does this change how you should write code?

- Use algorithms, not loops

How does this change how you should write code?

- Use algorithms, not loops
- Write to the most restricted programming model you can

How does this change how you should write code?

- Use algorithms, not loops
- Write to the most restricted programming model you can
- Use Kokkos (or something similar that is tracking standards for you)

Questions?