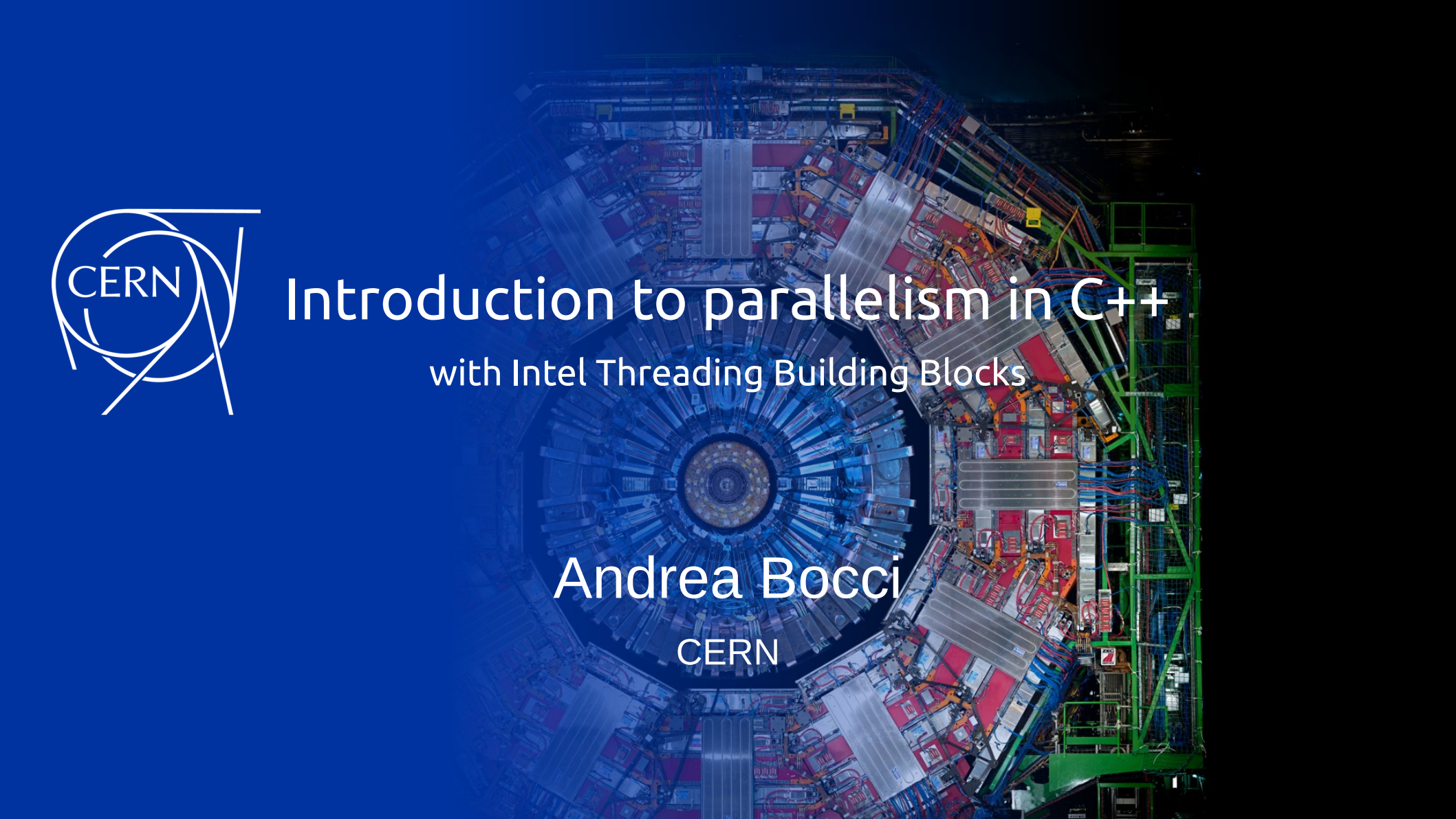


Introduction to parallelism in C++

with Intel Threading Building Blocks

Andrea Bocci

CERN

parallelism in C++ 11

parallelism in C++ 11

- C++ 11 introduced the building blocks to express parallelism in C++
  - threads:
    - `std::thread`
    - `std::jthread` (since C++20)
  - critical sections: “mutual exclusion”
    - `std::mutex`, …

    and locks:
    - `std::lock_guard`,
    - `std::scoped_lock` (since C++17), …
  - atomic operations:
    - `std::atomic<T>`,
    - `std::atomic_ref<T>` (since C++20)

parallelism in C++ 17

- C++ 17 introduced *parallel algorithms* and *execution policies* to express parallelism
  - `std::execution::sequenced_policy`
    - may not be parallelized
    - serial execution, same as the legacy algorithms (?)
  - `std::execution::parallel_policy`
    - may be parallelized
    - may run in the calling thread or in other threads managed by te library
  - `std::execution::parallel_unsequenced_policy`
    - may be parallelized, vectorised, or migrated across threads
  - `std::execution::unsequenced_policy` (since C++20)
    - may be vectorised, *e.g.* with SSE, AVX2, AVX512, *etc.*

  - if you can express your problem using algorithms,
    parallel algorithms give you a simple way to leverage parallelism to speed up your code

hands-on exercises

- `hands-on/tbb/`:

Name	Last commit message	Last commit date
..		
01_parallel_stl_sort	Move solutions to a separate directory	5 minutes ago
02_parallel_stl_saxpy	Move solutions to a separate directory	5 minutes ago
03_tbb_parallel_for_saxpy	Move solutions to a separate directory	5 minutes ago
04_images	Apply consistent formatting	6 hours ago
05_tbb_parallel_for_images	Apply consistent formatting	6 hours ago
06_tbb_graph	Use typedefs, add comments	21 minutes ago
07_tbb_parallel_for_local	Apply consistent formatting	6 hours ago
08_tbb_hierarchical	Use typedefs, add comments	21 minutes ago
.clang-format	Apply consistent formatting	6 hours ago

parallel algorithms: sorting

- hands-on/tbb/01_parallel_stl_sort/test.cc:
  - generate 1’000’000 random numbers
  - measure how long it takes to sort them
    - repeatedly

```
void measure(bool verbose, std::vector<std::uint64_t> v) {
  const auto start = std::chrono::steady_clock::now();
  std::sort(v.begin(), v.end());
  const auto finish = std::chrono::steady_clock::now();
  if (verbose) {
    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>(finish - start).count() << "ms\n";
  }
  assert(is_sorted(v));
};
```

  - use the parallel STL to speed up the sorting

parallel algorithms: user functions

- `hands-on/tbb/02_parallel_stl_saxpy/test.cc`:
  - generate a random scalar number `x`
  - generate two vectors of 100'000'000 random numbers `A` and `B`
  - measure how log it takes to apply the "saxpy" kernel to the vectors
    - (single precision) `A x + B`

```
template <typename T>
void axpy(T a, T x, T y, T& z) {
  z = a * x + y;
}

template <typename T>
void sequential_axpy(T a, std::vector<T> const& x, std::vector<T> const& y, std::vector<T>& z) {
  std::transform(x.begin(), x.end(), y.begin(), z.begin(), [a](T x, T y) -> T {
    T z;
    axpy(a, x, y, z);
    return z;
  });
}
```

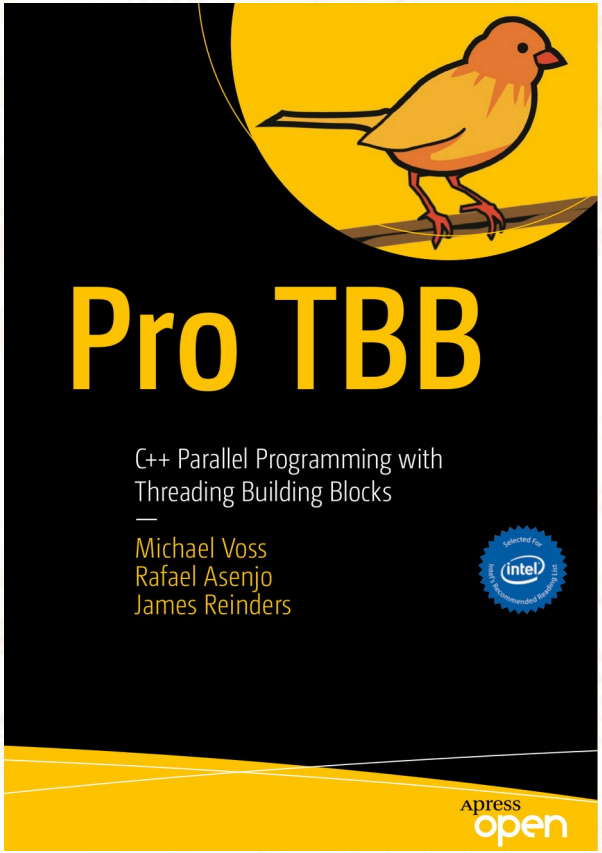
  - use the parallel STL to speed up the operations

parallelism with Intel TBB

parallelism with Intel TBB

- Intel Threading Building Blocks
  - now part of the oneAPI branding: oneTBB
  - including the official documentation and reference
  - migrating from the original TBB to oneTBB requires some small changes
- why TBB ?
  - scalability and load balancing
  - composability
  - multiple levels of parallelism
    - task-based parallelism: `parallel_invoke`, `parallel_pipeline`, various graph types
    - fork-join parallelism: `parallel_for`, various parallel algorithms
  - access to low level interface
    - task_group, task_arena, observers, *etc.*

Pro TBB



- Pro TBB (2019)
  - Voss, Asenjo, Reinders
  - https://doi.org/10.1007/978-1-4842-4398-5
  - open access book
- all examples in the book are on GitHub
  - https://github.com/Apress/pro-TBB
- the book describes the old TBB API
  - prior to the migration to oneTBB
  - use the oneTBB branch !

saxpy with TBB

- `hands-on/tbb/03_tbb_parallel_for_saxpy/test.cc`:
  - generate a random scalar number `x`
  - generate two vectors of 100'000'000 random numbers `A` and `B`
  - measure how log it takes to apply the "saxpy" kernel to the vectors
    - (single precision) `A x + B`

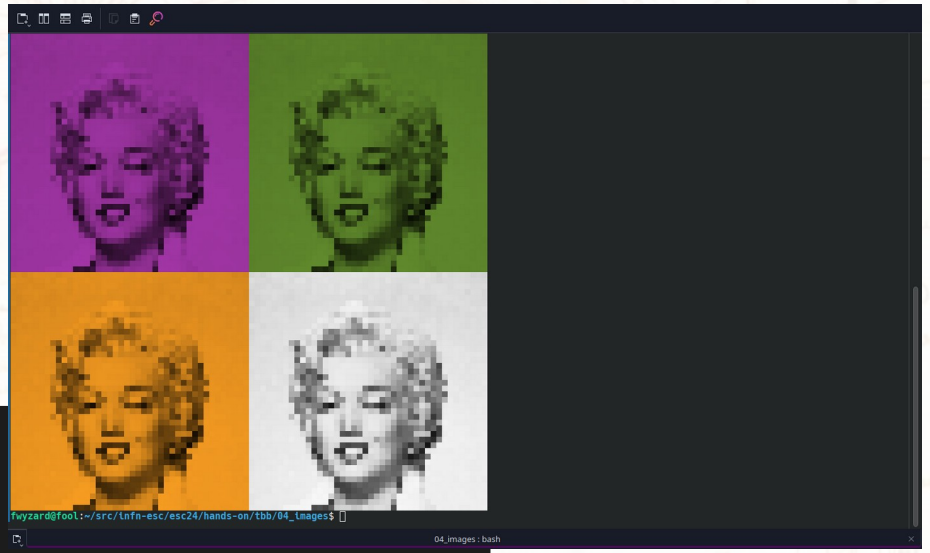
```
template <typename T>
void axpy(T a, T x, T y, T& z) {
  z = a * x + y;
}

template <typename T>
void sequential_axpy(T a, std::vector<T> const& x, std::vector<T> const& y, std::vector<T>& z) {
  std::size_t size = x.size();
  for (std::size_t i = 0; i < size; ++i) {
    axpy(a, x[i], y[i], z[i]);
  }
}
```

  - use `tbb::parallel_for` to speed up the operations

Art @ ESC24 !

dependencies

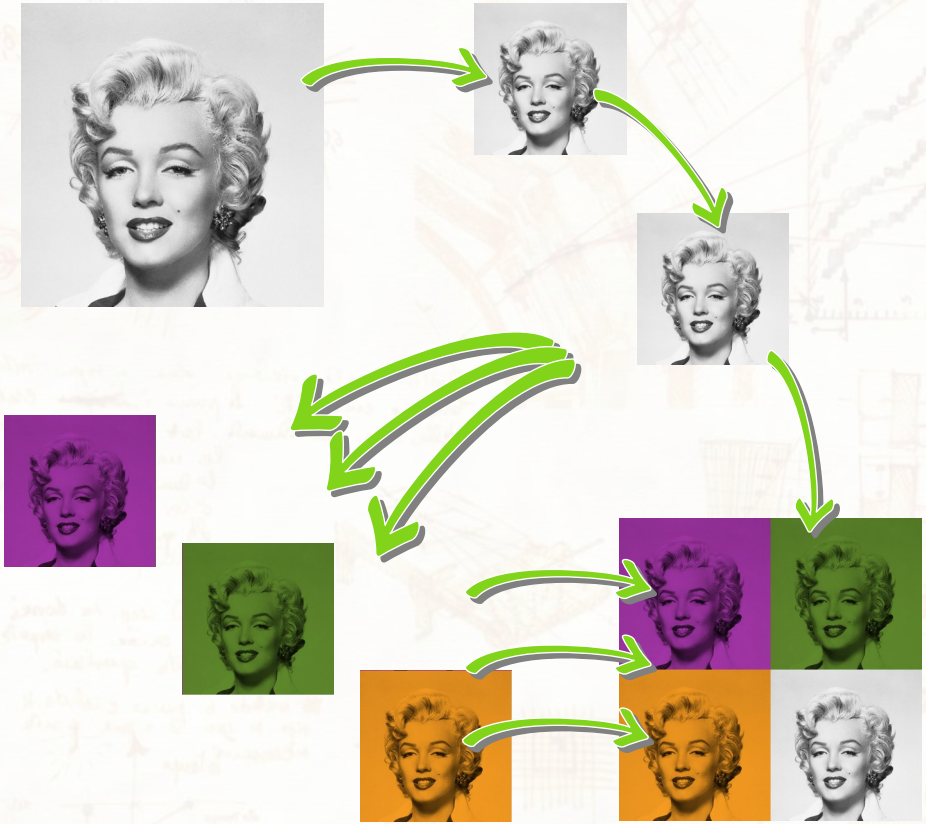
- stb_image.h and stb_image_write.h reading and writing image files
- {fmt} for formatted output
  - gcc 12 does not include c++20 `std::format`
  - **{fmt}** includes a lot more !
- both libraries can be used in header-only mode
  - increases compilation times
  - easier to set up



```
all: test

stb:
	git clone https://github.com/nothings/stb.git

fmt:
	git clone https://github.com/fmtlib/fmt.git

test: test.cc Makefile stb fmt
	g++ -std=c++20 -O3 -g -Istb -Ifmt/include -Wall -march=native -ltbb $< -o $@
```

our task list

- `hands-on/tbb/04_images/test.cc`:
  - read one image from a file
  - display the image on the terminal
  - make a 0.5×0.5 smaller copy of the image
  - convert the image to gray scale
  - make tinted copies
  - combine the gray scale and tinted images into a single image with the same size as the original
  - display the image on the terminal
  - write the image to a file



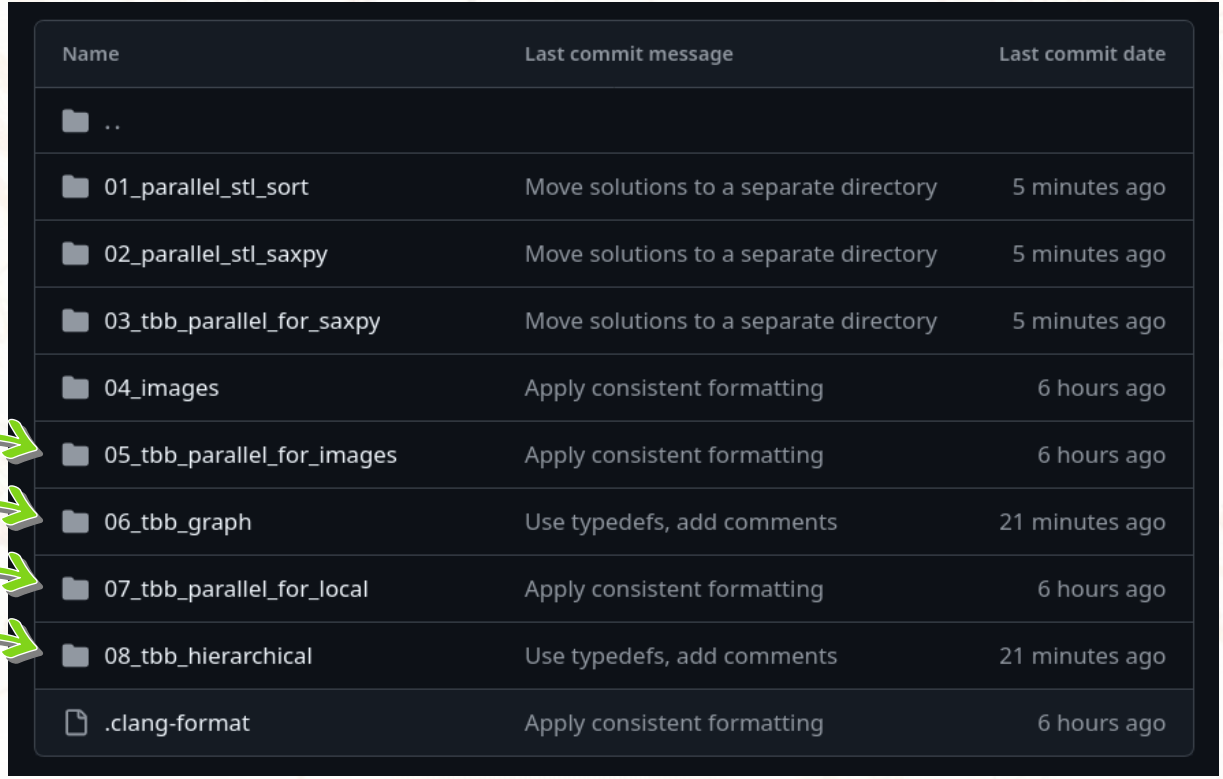
multiple levels of parallelism

- with TBB we can easily (?) express multiple levels of parallelism
  - **algorithmic parallelism**: parallelise the inner loops in the various algorithms
    - scaling
    - gray scaling
    - tinting
    - **very dependent on the algorithms**
  - **task-based parallelism**: parallelise the different tasks working on the same data
    - apply the different tints can be done in parallel
      - **note:** this is not an efficient approach… **why ?**
    - writing to disk in parallel in parallel to displaying on the terminal
    - **very dependent on the workflow**
  - **data parallelism:** process multiple images in parallel
    - weak scaling
    - **often the most efficient approach for large datasets**

- **composability**: you can also apply all of them to the same problem !

hands-on exercises

- `hands-on/tbb/`:



Name	Last commit message	Last commit date
..		
01_parallel_stl_sort	Move solutions to a separate directory	5 minutes ago
02_parallel_stl_saxpy	Move solutions to a separate directory	5 minutes ago
03_tbb_parallel_for_saxpy	Move solutions to a separate directory	5 minutes ago
04_images	Apply consistent formatting	6 hours ago
05_tbb_parallel_for_images	Apply consistent formatting	6 hours ago
06_tbb_graph	Use typedefs, add comments	21 minutes ago
07_tbb_parallel_for_local	Apply consistent formatting	6 hours ago
08_tbb_hierarchical	Use typedefs, add comments	21 minutes ago
.clang-format	Apply consistent formatting	6 hours ago

questions ?