Introduction to Intel Threading Building Blocks

Andrea Bocci

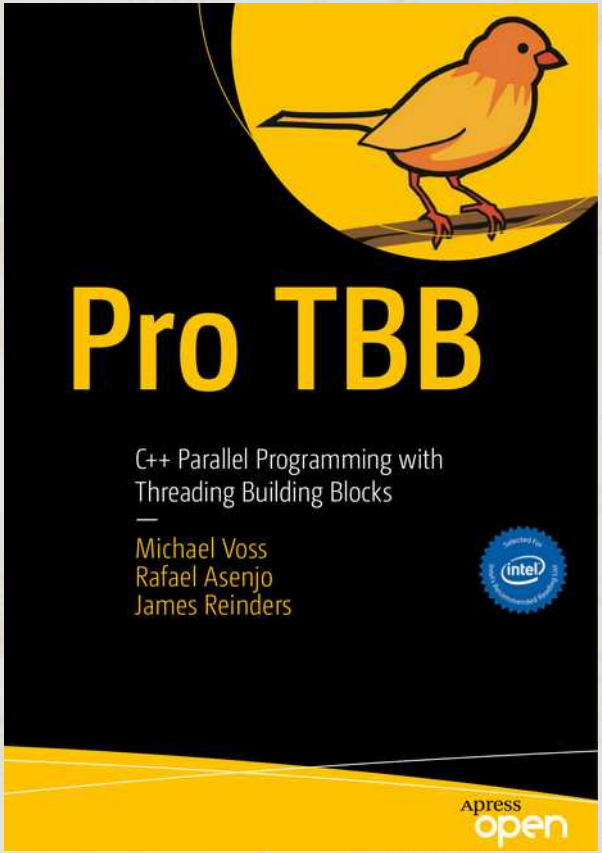
CERN

parallelism with Intel TBB

- Intel Threading Building Blocks
  - now part of the oneAPI ecosystem: 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 !

setting up oneTBB

- the latest version of oneTBB is installed on the ESC machines under: `/opt/intel/oneapi/tbb/latest`
  - make it available to the compiler and to the program at run time:

```
$ source /opt/intel/oneapi/tbb/latest/env/vars.sh
```

  - check that it worked:

```
$ echo $TBBROOT
/opt/intel/oneapi/tbb/2022.2/env/..
```

- we need gcc 14 for some of the exercises
  - load the gcc 14 environment:

```
$ source scl_source enable gcc-toolset-14
$ gcc -version
gcc (GCC) 14.2.1 20250110 (Red Hat 14.2.1-7)
...
```

tbb::parallel_for

- **parallel_for** is the simplest parallel algorithm

```
template<typename Index, typename Func>
void parallel_for(Index first, Index last, Index step, const Func& func);
```

  - `step` is optional, defaults to 1
  - `func` can be a lambda !
- replace

```
for (Index i = first; i < last; i += step) {
    // … loop body …
}
```

- with

```
tbb::parallel_for<Index>(first, last, step, [&](Index i){
    // … loop body …
});
```

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

parallel_for partitioner

- **`parallel_for`**
  - splits the input range in *chunks*
  - executes the loop body over each chunk in parallel
- a *partitioner* specifies a strategy for splitting into chunks and executing the loop:
  - `static_partitioner`
    - split the work in chunks of approximately equal size, to keep all threads busy
    - may be more efficient if all items take approximately the same time
  - `auto_partitioner` (default)
    - similar approach, but may split the work further if some items take longer than others
  - `affinity_partitioner`
    - similar approach, tries to maintain cache affinity across multiple loops
  - `simple_partitioner`
    - split the input range as much as possible
    - useful with a `blocked_range` to process 1-, 2-, or 3-dimensional chunks

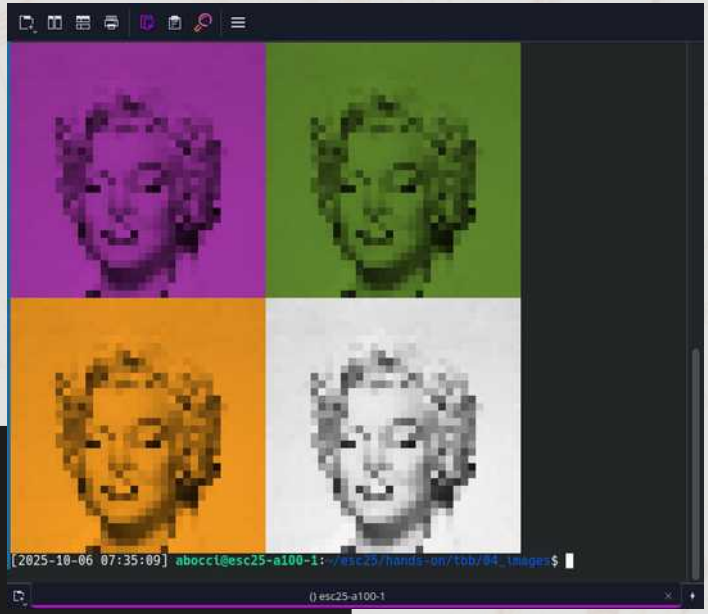
saxpy with different partitioners

- try to use the various partitioners
  - what gives the best performance ?
  - what is a good chunk size for the `simple_partitioner` ?
  - does the `affinity_partitioner` make sense in this case ?

Art @ ESC24 !

dependencies

- stb_image.h and stb_image_write.h reading and writing image files
- {fmt} for formatted output
  - c++20 includes `std::format`
  - **{fmt}** includes a lot more !
- both libraries can be used in header-only mode
  - increases compilation times
  - easier to set up

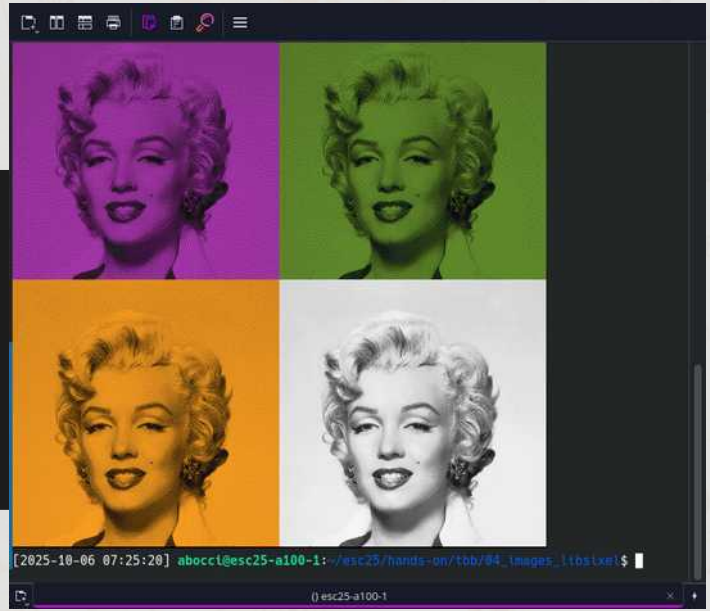



```
all: test

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 $@
```

an alternative graph library

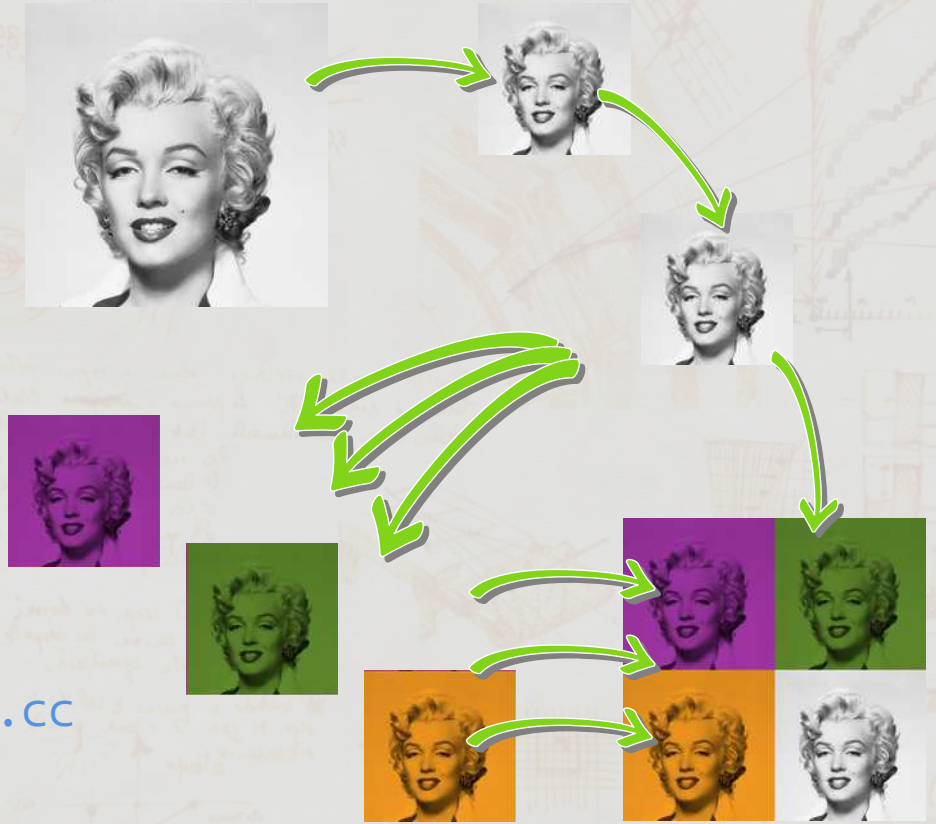
- libsixel
  - provides encoder/decoder implementation for DEC SIXEL graphics, and some converter programs
  - lets you display graphical images directly on the terminal
    - if your terminal supports it
  - needs to be build and installed

```
git clone git@github.com:saitoha/libsixel.git build/libsixel
cd build/libsixel
./configure --without-libcurl --without-jpeg --without-png \
    --without-pkgconfigdir --without-bashcompletiondir \
    --without-zshcompletiondir --disable-python \
    --prefix=$(realpath ../../libsixel)
make -j8 install
cd ../../
rm -rf build
```



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



- `hands-on/tbb/04_images_libsixel/test.cc`
  - libsixel version, otherwise same functionality

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
    - writing to disk 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
..		
03_tbb_parallel_for_saxpy	Import TBB hands-on from ESC24 and update for gcc 14	yesterday
04_images	Import TBB hands-on from ESC24 and update for gcc 14	yesterday
04_images_libsixel	Add alternative image display using libsixel	yesterday
05_tbb_parallel_for_images	Import TBB hands-on from ESC24 and update for gcc 14	yesterday
06_tbb_graph	Import TBB hands-on from ESC24 and update for gcc 14	yesterday
07_tbb_parallel_for_local	Import TBB hands-on from ESC24 and update for gcc 14	yesterday
08_tbb_hierarchical	Import TBB hands-on from ESC24 and update for gcc 14	yesterday
.clang-format	Import TBB hands-on from ESC24 and update for gcc 14	yesterday
.gitignore	Import TBB hands-on from ESC24 and update for gcc 14	yesterday

questions ?