Dustin Heither, Maximilian Achenbach, and Robert Kagan

Optimizing Deep Learning Performance:

A Hybrid CPU-GPU Framework with Multithreading, SIMD,
and Evaluation of Efficiency Metrics

Outline

6. Presentation

Tweaks & Enhancements

Overview

- **Optimizing Code Efficiency**
  - **Goal:** Reduce code complexity and binary size
  - **Solution:** Removed unused functions
- **Improving Data Processing Speed**
  - **Goal:** Eliminate unnecessary transformation step
  - **Solution:** Integrated transpose into flatten
- **Clang and C++11 Extensions**
  - **Goal:** Reduce boilerplate and boost functionality
  - **Solution:** Use clang-specific and C++11 features

```
DATA_TYPE *malloc_cuda(int N, int M) {
    size_t bytes = N * M * sizeof(DATA_TYPE);

    DATA_TYPE *d_matrix;
    cudaMalloc(&d_matrix, bytes);

    return d_matrix;
}

for(int i = 0; i < a->x / len; i++) {
    for(int j = 0; j < a->y; j++) {
        for(int m = 0; m < len; m++) {
            int idx = i * a->y * len + j * len + m;
                c->m[get_idx(0, idx, c->y)] = a->m[get_idx(i, j, a->y) + m *
                  ((a->x / len) * a->y)];
            }
        }
}
```

Hardware & Benchmark
Hardware

CPU	Release date	TDP (W)	Number of (performance) cores	Number of threads
AMD Ryzen 7 3800XT	7. Juli 2020	105	8	16
Apple M3 Pro 11-Core	30. Oktober 2023	27	5	11
Intel Core i7 1065G7	1. Juni 2019	15	4	8

GPU	Release date	TDP (W)	Number of CUDA cores	Base Clock (MHz)
NVIDIA GeForce RTX 2080	20. September 2018	215	2944	1515
NVIDIA GeForce MX350	10. February 2020	20	640	1354

Hardware & Benchmark

Benchmark

- **Batch size:** 1
- **Epochs:** 10

- **Unit:**
  - Total time
  - In microseconds (µs)
  - Averaged over 12 runs
  - Discarding first 2 runs
  - Lower is better

- **Old:**
  - Last presentation

- **Quantization:**
  - **Data type:** int32

- **XL:**
  - **Image dimensions:** (**32**x30)^2

- **XXL:**
  - **Image dimensions:** (**64**x30)^2

CUDA Tuning

Overview

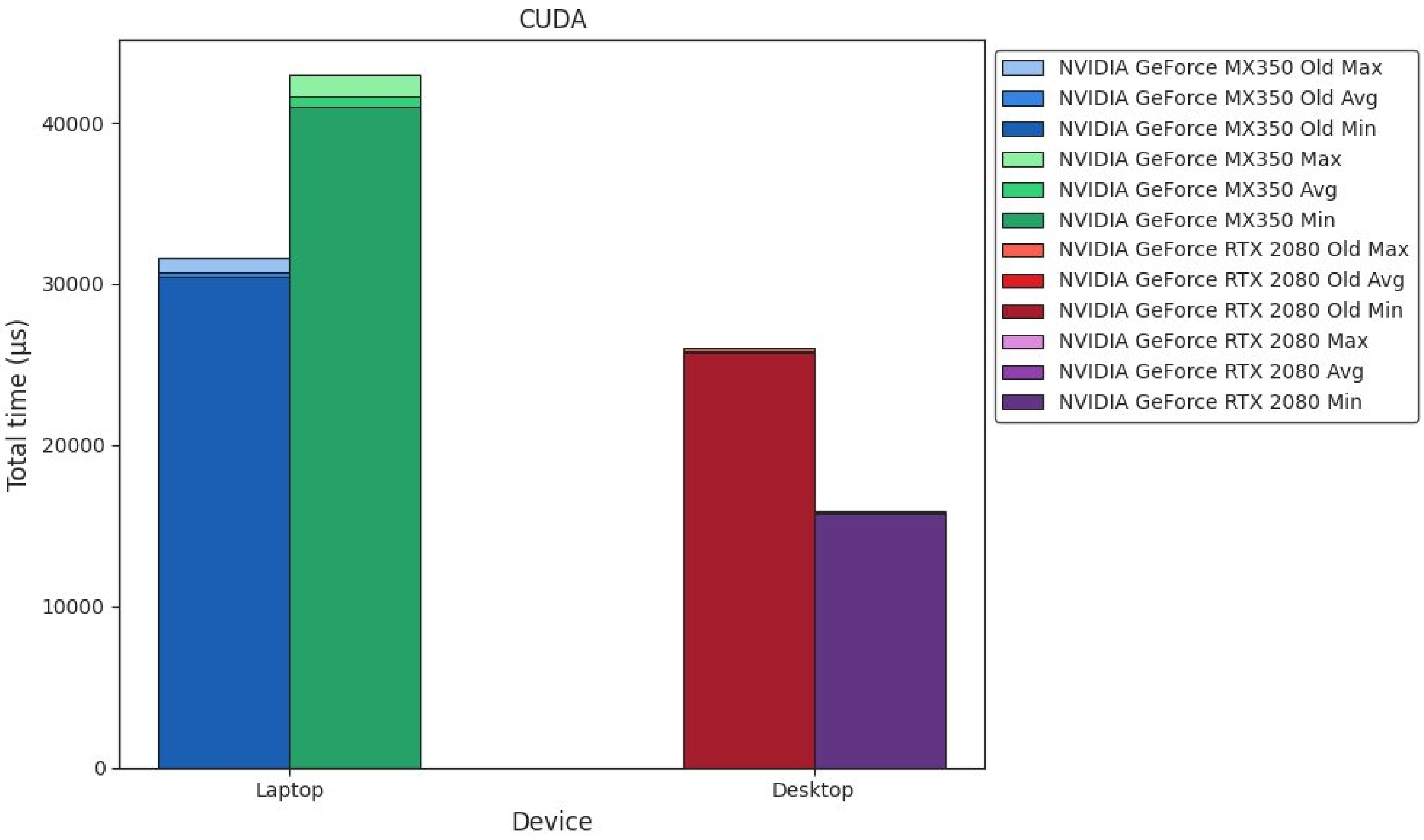
- **Removed transposing of fc_weights:**
  - **Change:** Eliminated unnecessary transposition of fully connected layer weights
  - **Benefit:** Reduces overhead and improves memory efficiency

- **Implemented CUDA Shared Memory for Matrix Multiplication:**
  - **Change:** Implemented shared memory utilization for XL matrix multiplication
  - **Benefit:** Boosts computational efficiency on CUDA hardware

- **Eliminated cudaDeviceSynchronize call:**
  - **Change:** Removed an unnecessary synchronization call in CUDA code
  - **Benefit:** Reduces latency and enhances execution performance

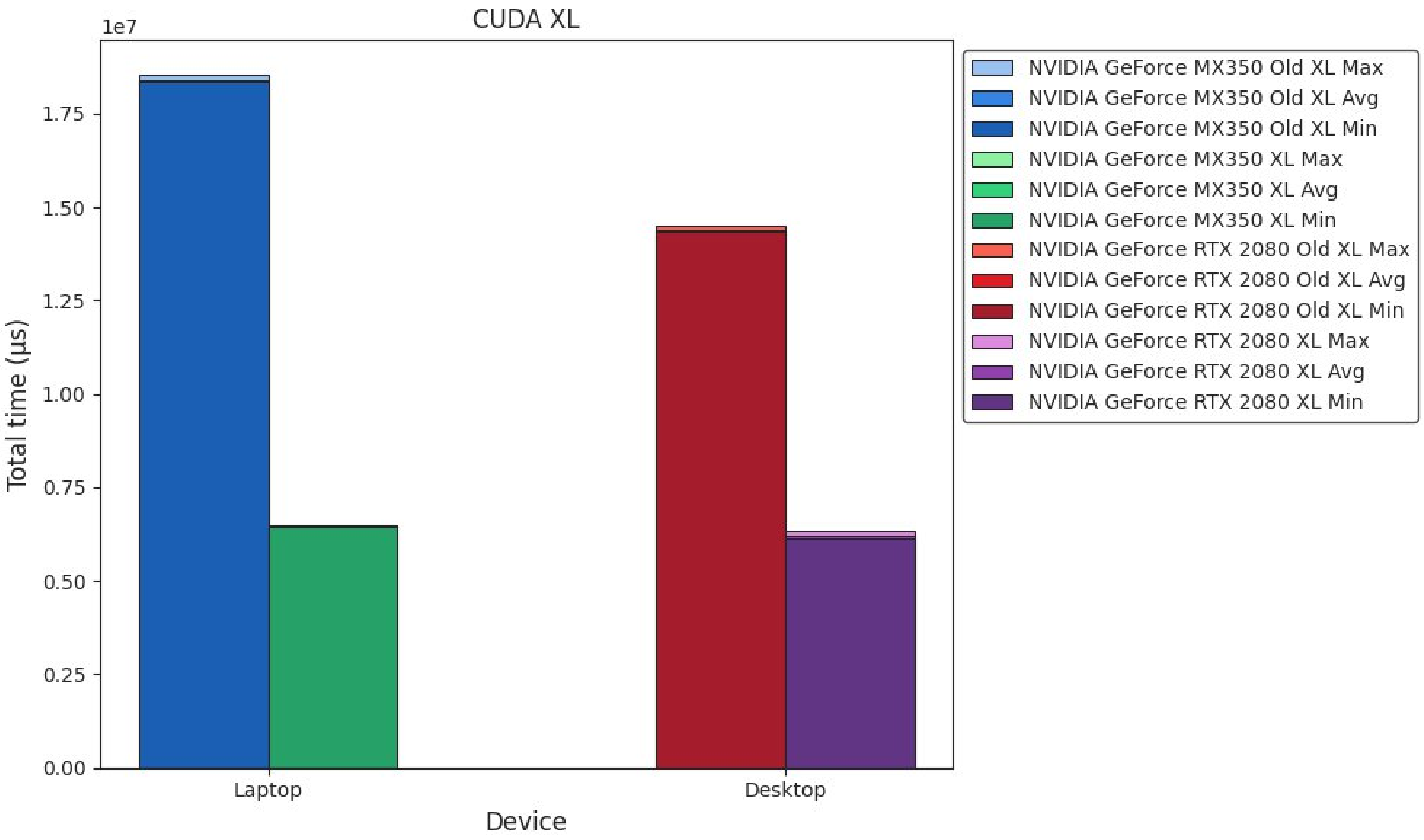
Benchmark

CUDA



Benchmark

CUDA XL



Multithreading

1 / 3

- **Separated NVIDIA, OpenMP, and Multithreading**
  - **Change:** Modularized different parallel frameworks
  - **Benefit:** Clearer separation and easier maintenance
- **Fixed Smart Multithreading**
  - **Change:** Refined adaptive thread management logic
  - **Benefit:** More robust and efficient multithreading

```
__attribute__((always_inline)) inline void run_nvidia(int i);
__attribute__((always_inline)) inline void run_omp(int i);
__attribute__((always_inline)) inline void run(int t, int i);

mt_arg arg[instance->THREADS];
   for(int i = 0; i < c->x; i++) {
      for(int j = 0; j < instance->THREADS; j++) {
         arg[j].a = &a;
         arg[j].b = &b;
         arg[j].c = &c;
         arg[j].i = i;
         if(instance->THREADS < c->y) {
            arg[j].single_core = 1;
            instance->matmul_mt(&arg[i]);
            break;
         }
         arg[j].single_core = 0;
         arg[j].start_routine = instance->matmul_mt_wrapper;
         instance->push_mt(&arg[j]);
      }
      if(instance->THREADS >= c->y) {
         instance->wait_mt();
      }
}
```

Multithreading

2 / 3

- **Implemented Dedicated Multithreading Class**
  - **Change:** Centralized thread management in one class
  - **Benefit:** Improved reusability and consistency

- **Merged Multithreading Files into New Class Constructor**
  - **Change:** Combined two separate multithreading files into one unified class constructor
  - **Benefit:** Simplified initialization process and improved maintainability

```
class mt {
  public:
    int COUNTER;
    int THREADS;
    GAsyncQueue *queue;
    pthread_cond_t cond;
    pthread_mutex_t mutex;
    pthread_t tids[(int)(CHAR_BIT * sizeof(void*))];

    mt() : mt(1) {}

    mt(int threads) {
      COUNTER = 0;
      THREADS = threads;
      queue = g_async_queue_new();
      pthread_mutex_init(&mutex, NULL);
      pthread_cond_init(&cond, NULL);
      mt_struct arg[THREADS];
      for(int i = 0; i < THREADS; i++) {
        arg[i].instance = this;
        arg[i].idx = i;
        pthread_create(&tids[i], NULL, start_mt, &arg[i]);
      }
      wait_mt();
    }

    ~mt() {
      mt_arg arg[THREADS];
      for(long i = 0; i < THREADS; i++) {
        arg[i].start_routine = stop_mt;
        push_mt(&arg[i]);
      }
      for(long i = 0; i < THREADS; i++) {
        pthread_join(tids[i], NULL);
      }
      pthread_cond_destroy(&cond);
      pthread_mutex_destroy(&mutex);
      g_async_queue_unref(queue);
    }
}
```

Multithreading

3 / 3

- **Implemented Multithreaded Image Processing**
  - **Change:** Introduced a concurrent image processing pipeline
  - **Benefit:** Enhanced performance through parallel processing

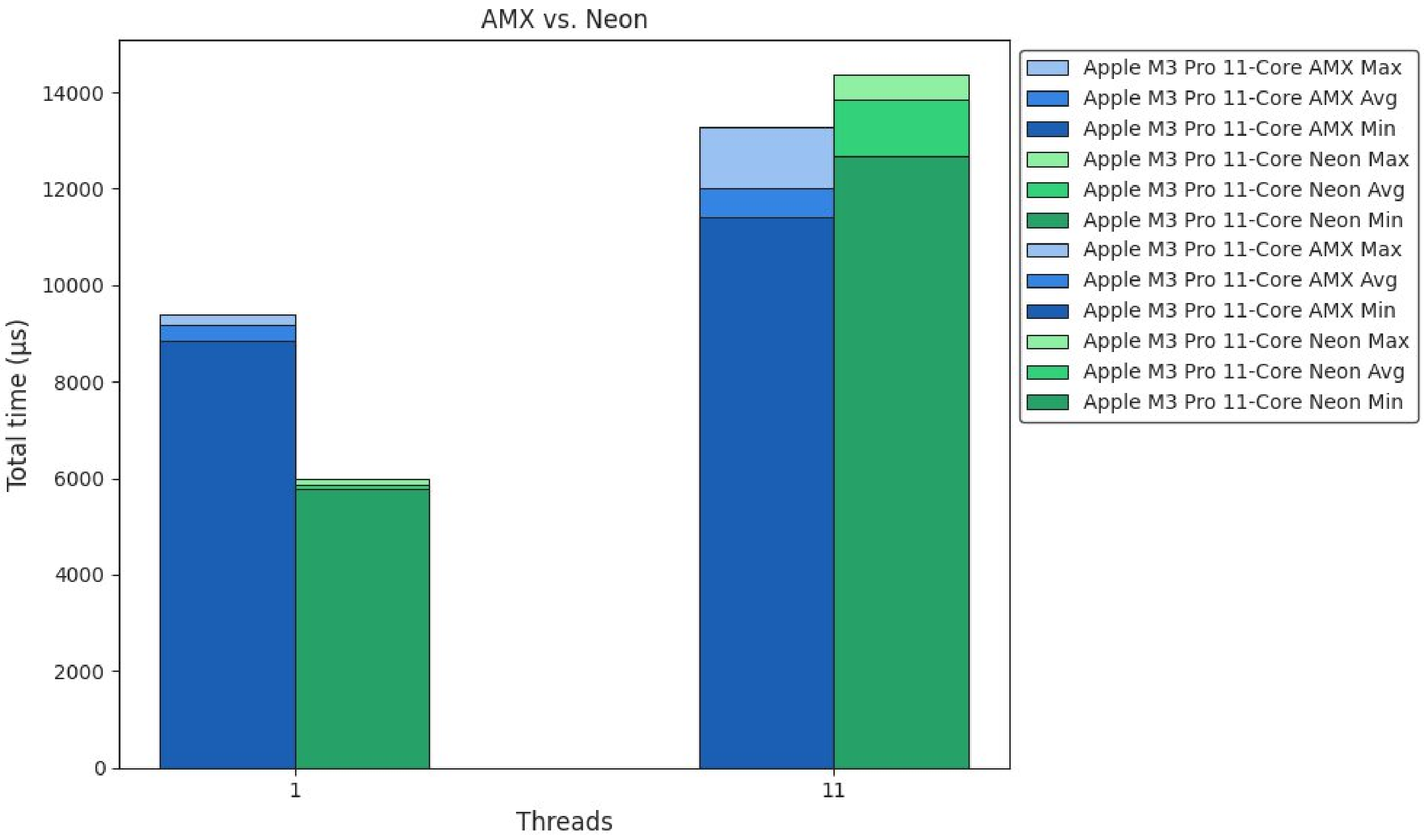
```
#ifndef IMG_HPP
  #define IMG_HPP

  #include "tf.hpp"

  #ifndef NVIDIA
    __attribute__((always_inline)) inline void process_images(void *instance,
            mt_arg *arg) {
      mt *parent_instance = (mt*)instance;
      mt child_instance(parent_instance->THREADS);
      // malloc
      for(int i = arg->idx * (arg->io->image_len / parent_instance->THREADS); i
            < ((arg->idx + 1) * (arg->io->image_len /
            parent_instance->THREADS)) + (arg->idx ==
            parent_instance->THREADS - 1 ? arg->io->image_len %
            parent_instance->THREADS : 0); i++) {
        // tf
      }
      // free
      parent_instance->wait_mt();
    }
  #endif
#endif
```

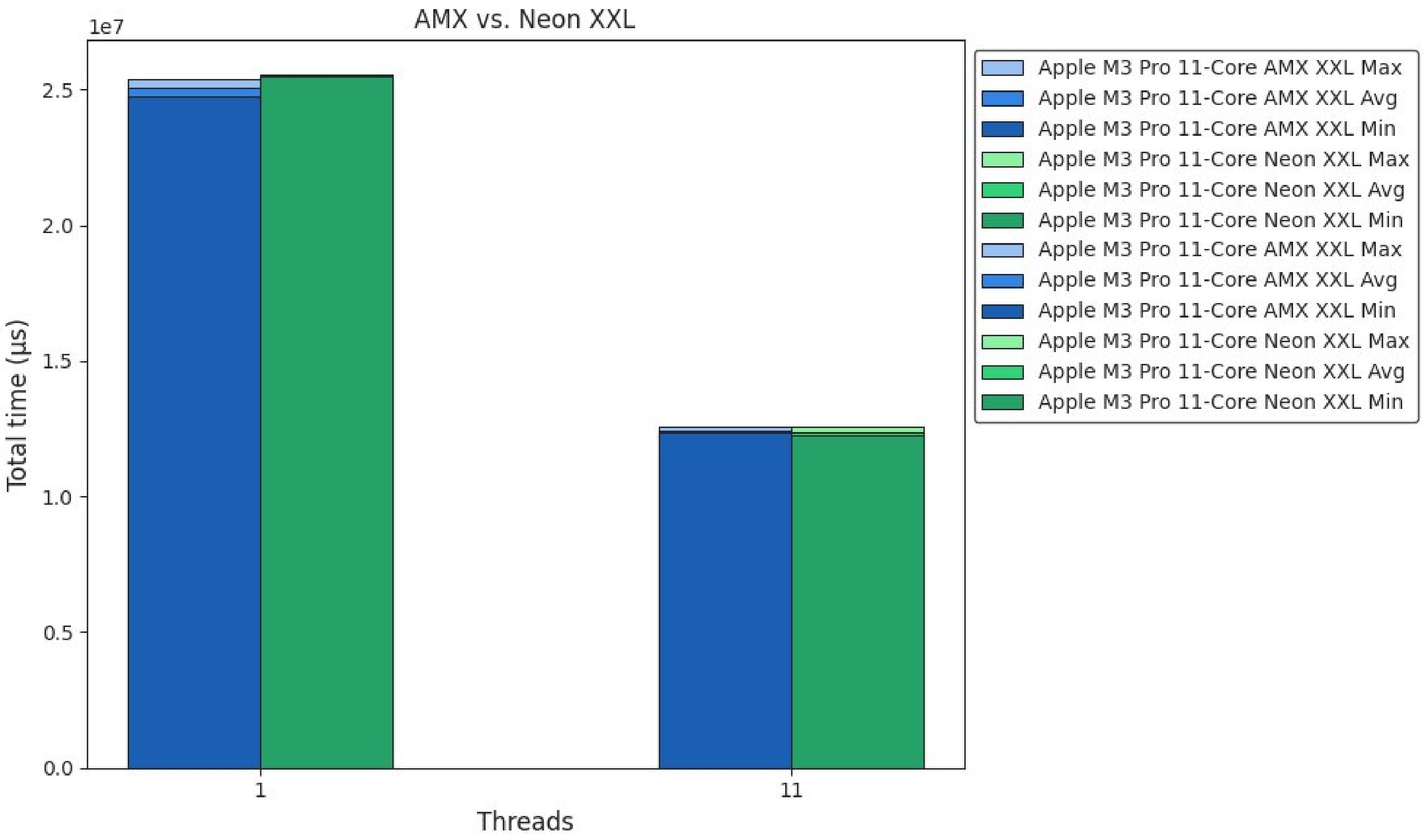
Benchmark

AMX vs. Neon



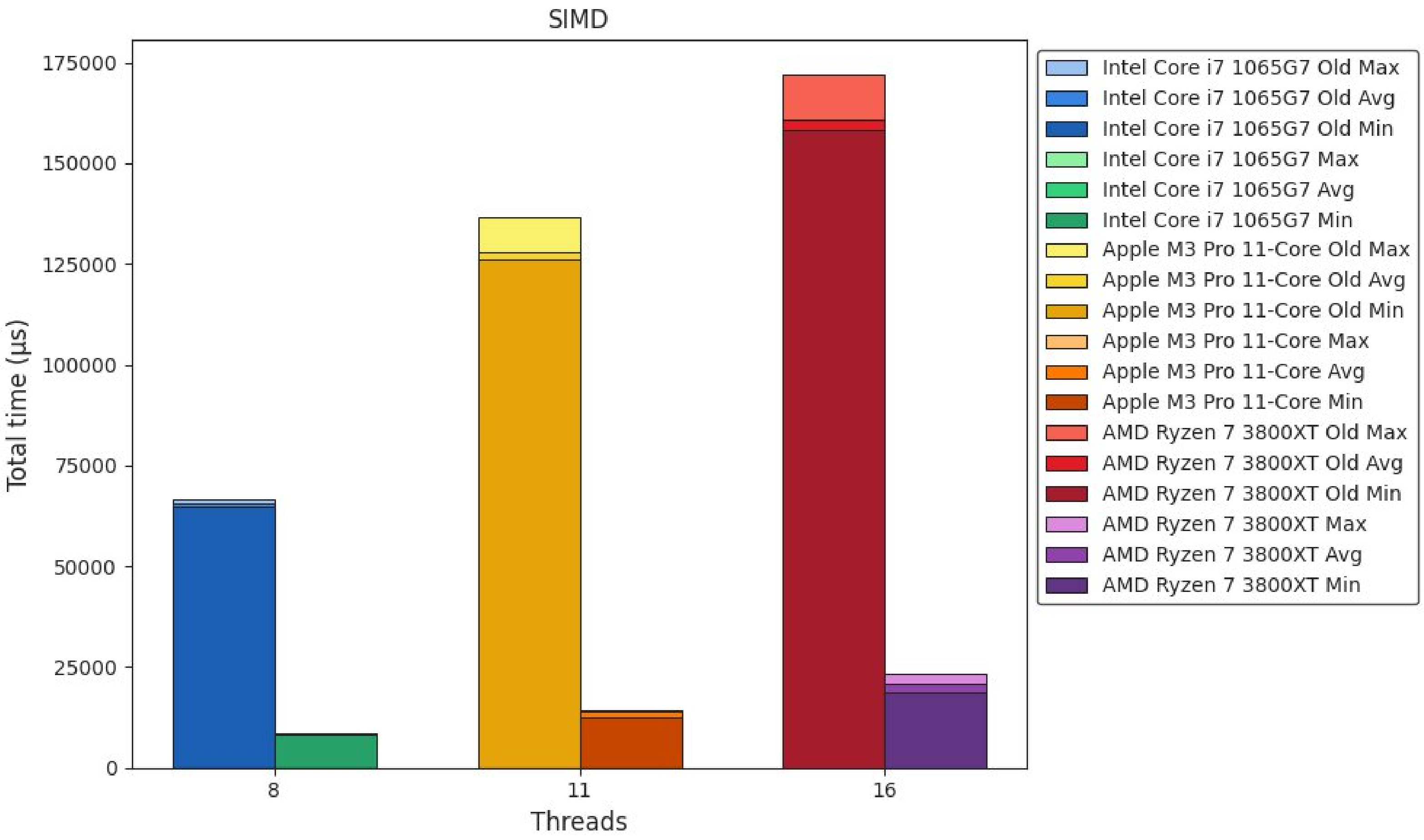
Benchmark

AMX vs. Neon XXL



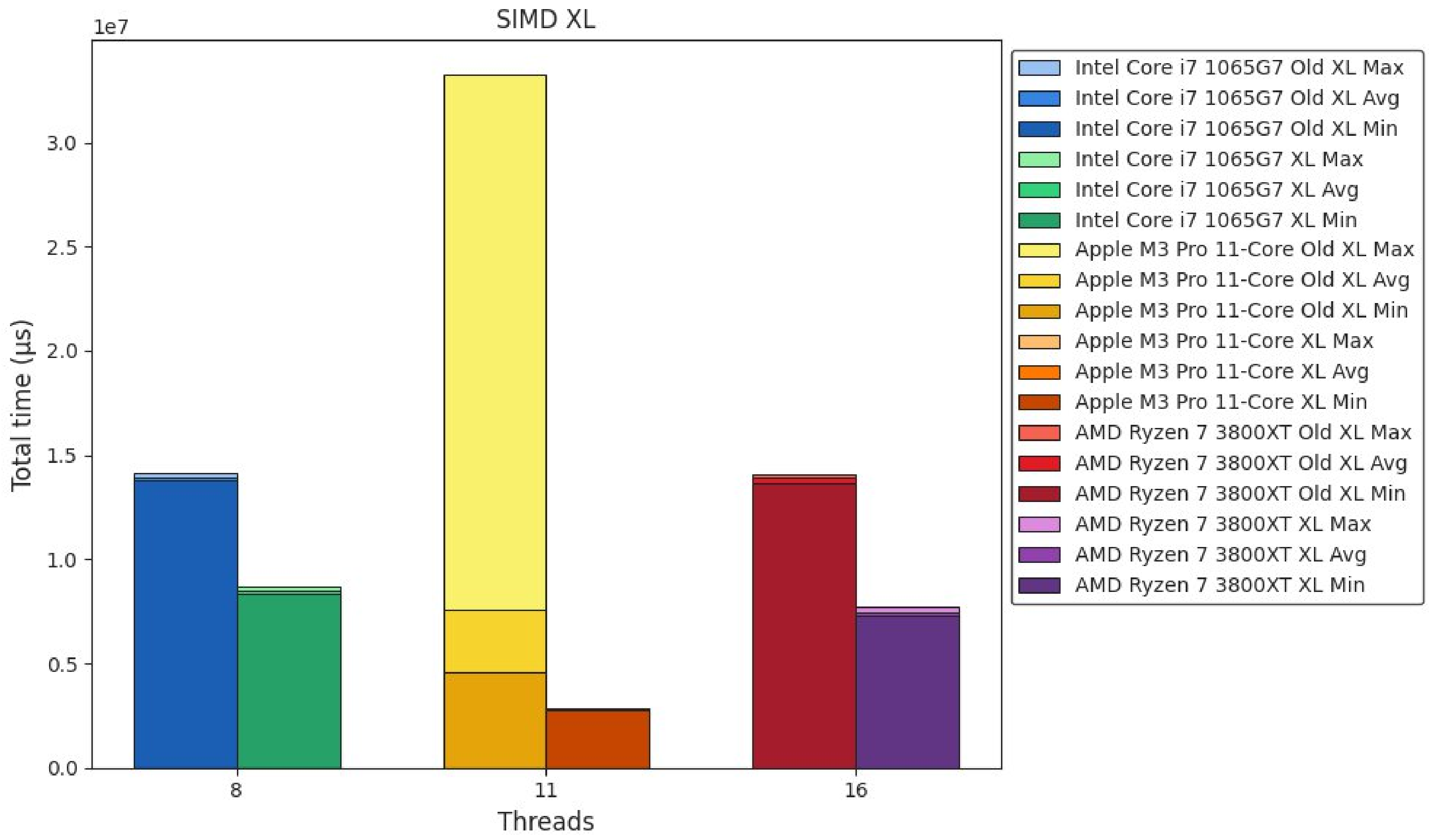
Benchmark

SIMD



Benchmark

SIMD XL



Quantization

Overview

- **Quantization in export_image.py and export_xl.py**
  - **Change:** Implemented quantization in image export scripts
  - **Benefit:** Reduces file size and memory usage for exported images

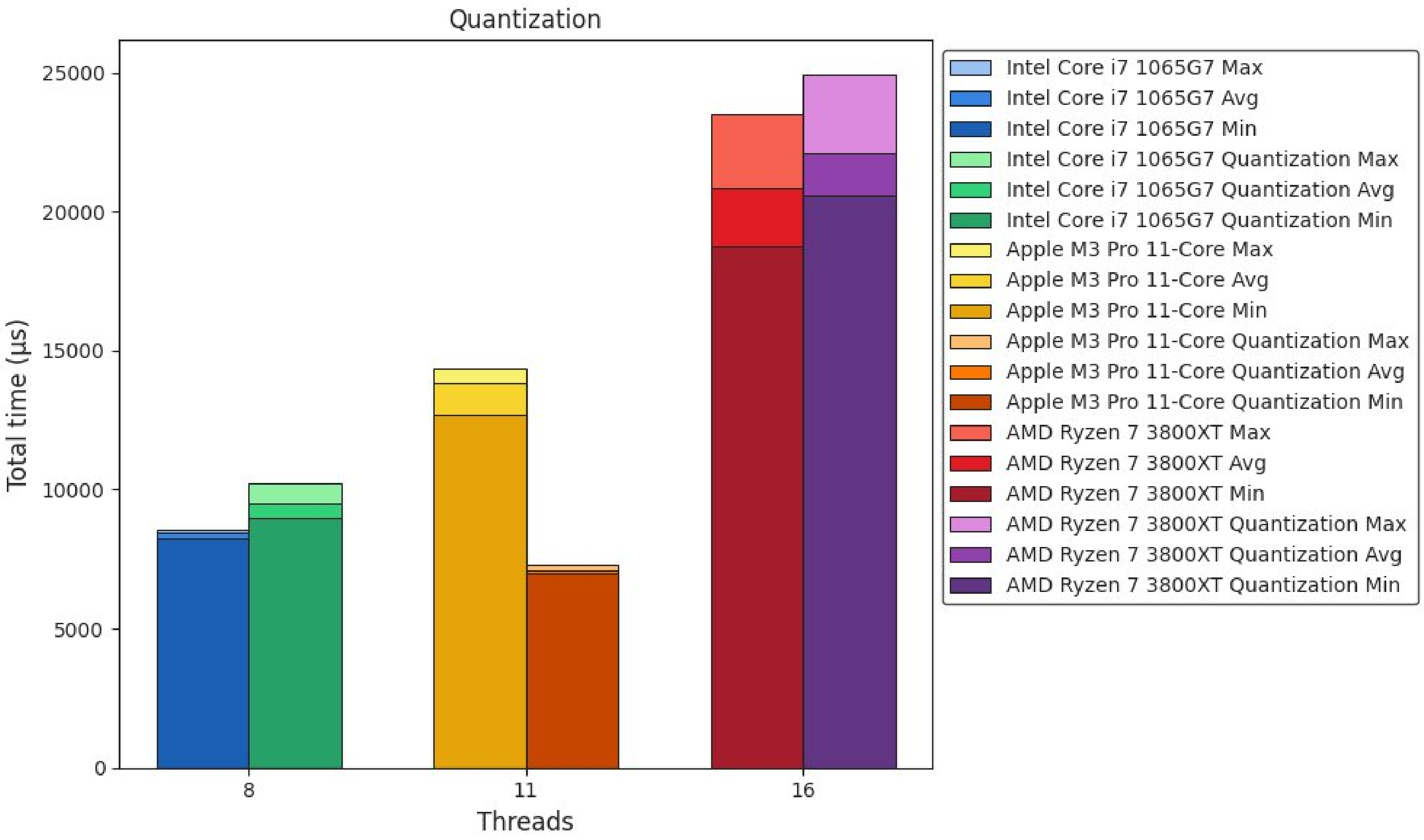
- **Quantization in tf.py**
  - **Change:** Integrated quantization into the training process
  - **Benefit:** Improves model efficiency and reduces inference time

- **Fixed quantized SIMD**
  - **Change:** Corrected SIMD-based quantization
  - **Benefit:** Enhances performance and accuracy of vectorized computations

- **Exported quantized weights**
  - **Change:** Saved model weights in a quantized format
  - **Benefit:** Reduces storage requirements and speeds up model loading

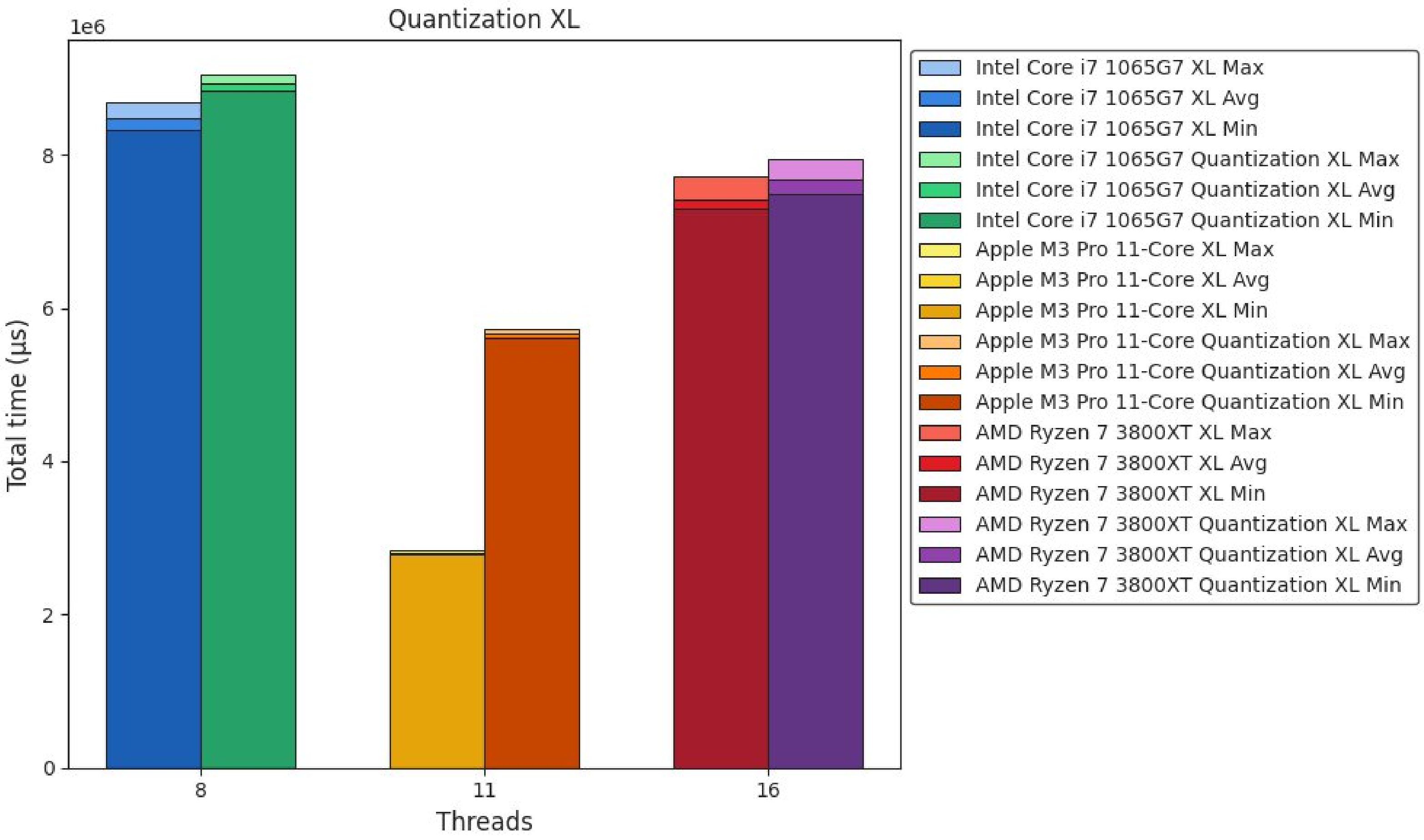
Benchmark

Quantization



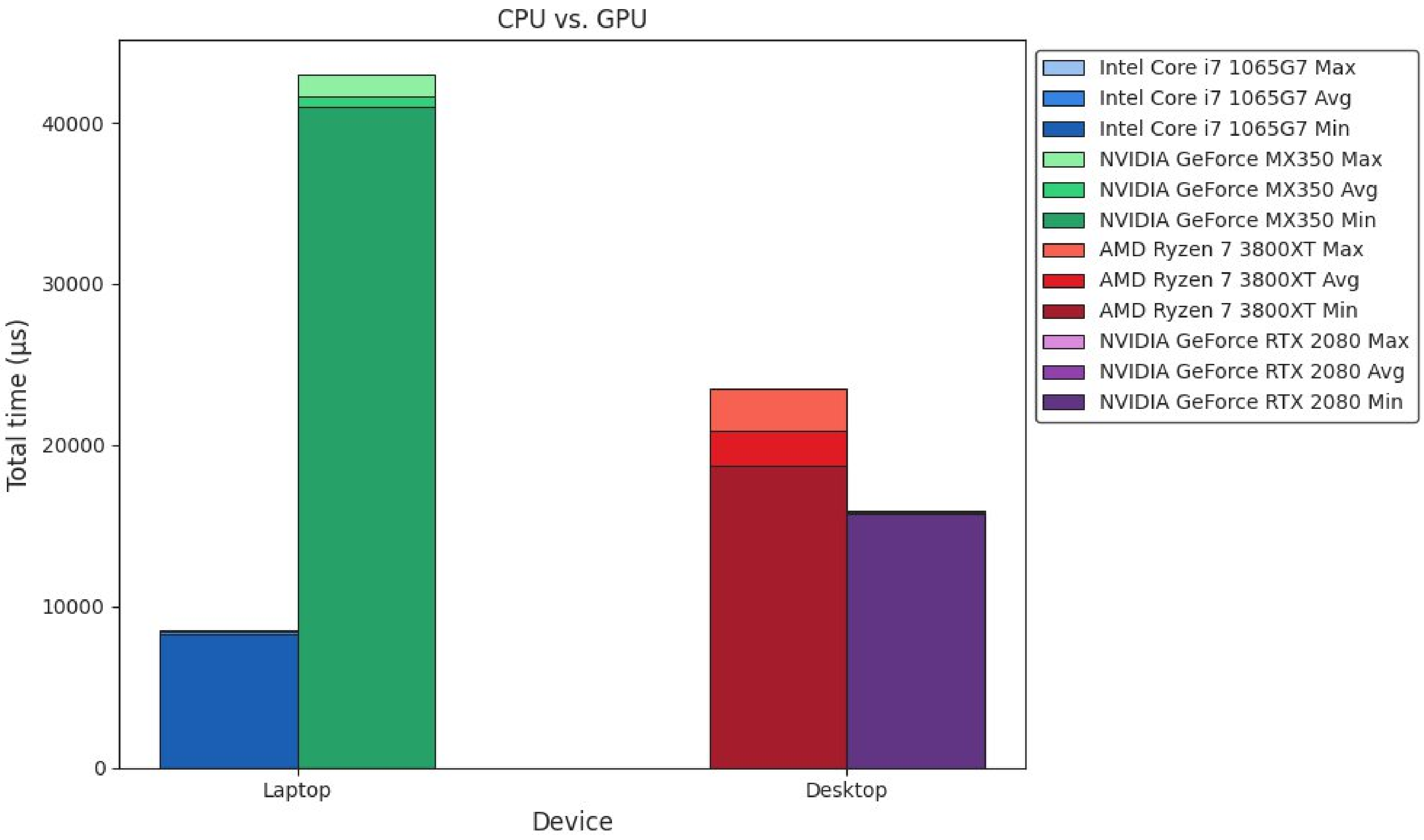
Benchmark

Quantization XL



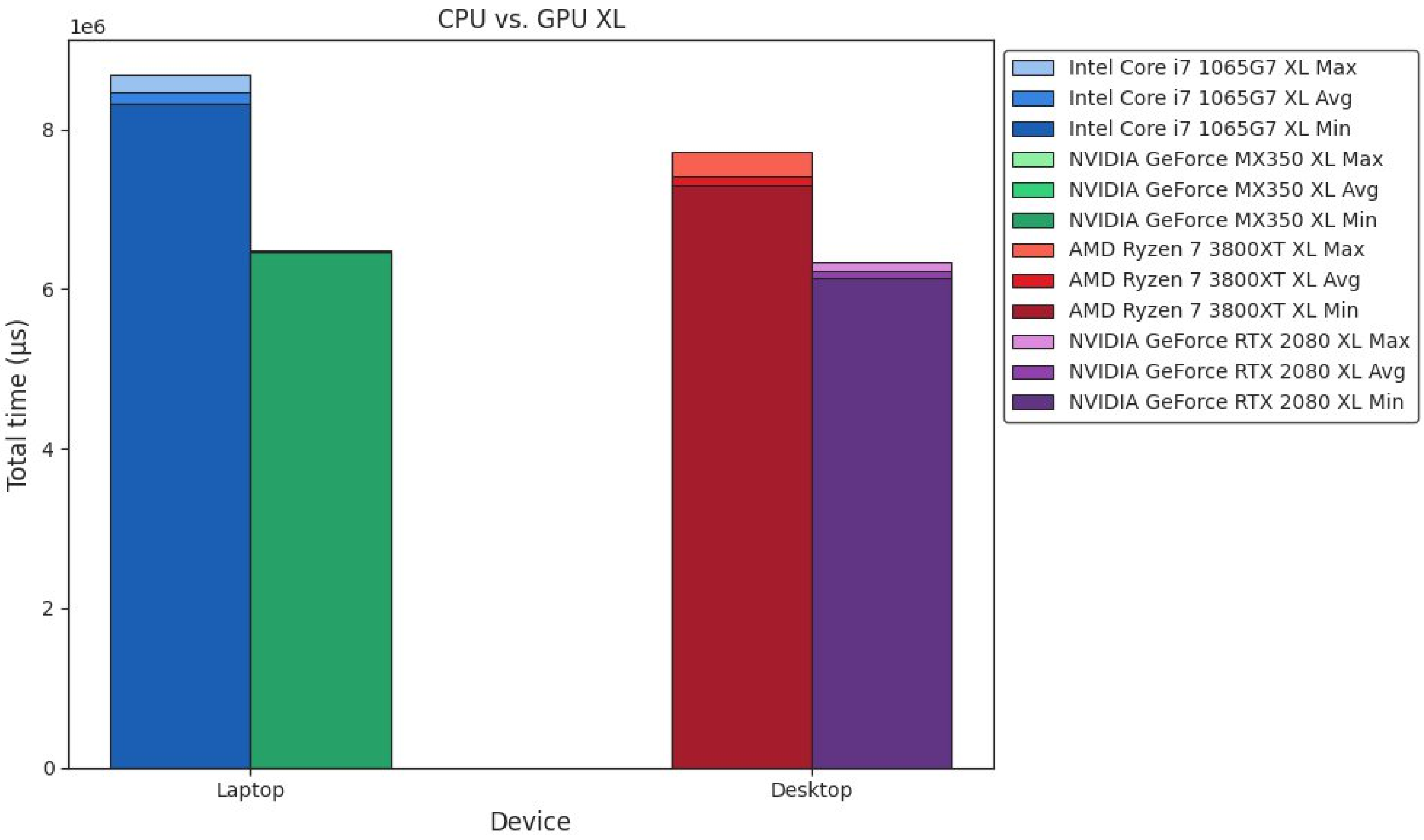
Benchmark

CPU vs. GPU



Benchmark

CPU vs. GPU XL



Outlook

Overview

Work Completed ✅

- One Framework
- CUDA Tuning
- ICPX vs. Clang Comparison
- OpenMP Integration
- Multithreading Optimization
- SIMD Implementation
- Quantization Support

To Do‘s 🚧

- Final Presentation
- Final Paper Submission 💀