Rechnerarchitekturen für Deep-Learning Anwendungen (RADL)

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

5. Presentation

Tweaks & Enhancements

Overview

- **Build System Enhancements:**
  - **Goal:** Support new compiler flags
  - **Solution:** Updated make config
- **Data Preprocessing:**
  - **Goal:** Avoid repeated transposing
  - **Solution:** Pre-transposed fc_bias and fc_weights
- **Function Inlining:**
  - **Goal:** Minimize overhead caused by function calls
  - **Solution:** Inlined all remaining functions
- **Optimization Flags:**
  - **Goal:** Enable optimizations and detect memory issues
  - **Solution:** Integrated -O3 and -fsanitize=address
- **AVX-512 Optimization:**
  - **Goal:** Avoid repeated inline assembly runtime checks for AVX-512
  - **Solution:** Relocated the checks for AVX-512 to the Makefile

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:** 128

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

- **Old:**
  - Last presentation
- **Quantization:**
  - **Data type:** int8
- **XL:**
  - **Image dimensions:** (**32**x30)^2

CUDA Tuning

Overview

- **Fixed CUDA Transpose:**
  - **Change:** Corrected the previously incorrect implementation
  - **Benefit:** Ensured accurate functionality and improved code reliability
- **Implemented CUDA Constant Memory for Conv2D:**
  - **Change:** Optimized kernel operations using constant memory for frequently accessed data
  - **Benefit:** Reduced memory latency and improved execution speed
- **Implemented CUDA Pipeline:**
  - **Change:** Introduced efficient pipelining mechanisms for overlapping data transfer and computation
  - **Benefit:** Enhanced GPU utilization and throughput
- **Separated GPU Allocations from CPU:**
  - **Change:** Decoupled memory management between CPU and GPU
  - **Benefit:** Improved resource management

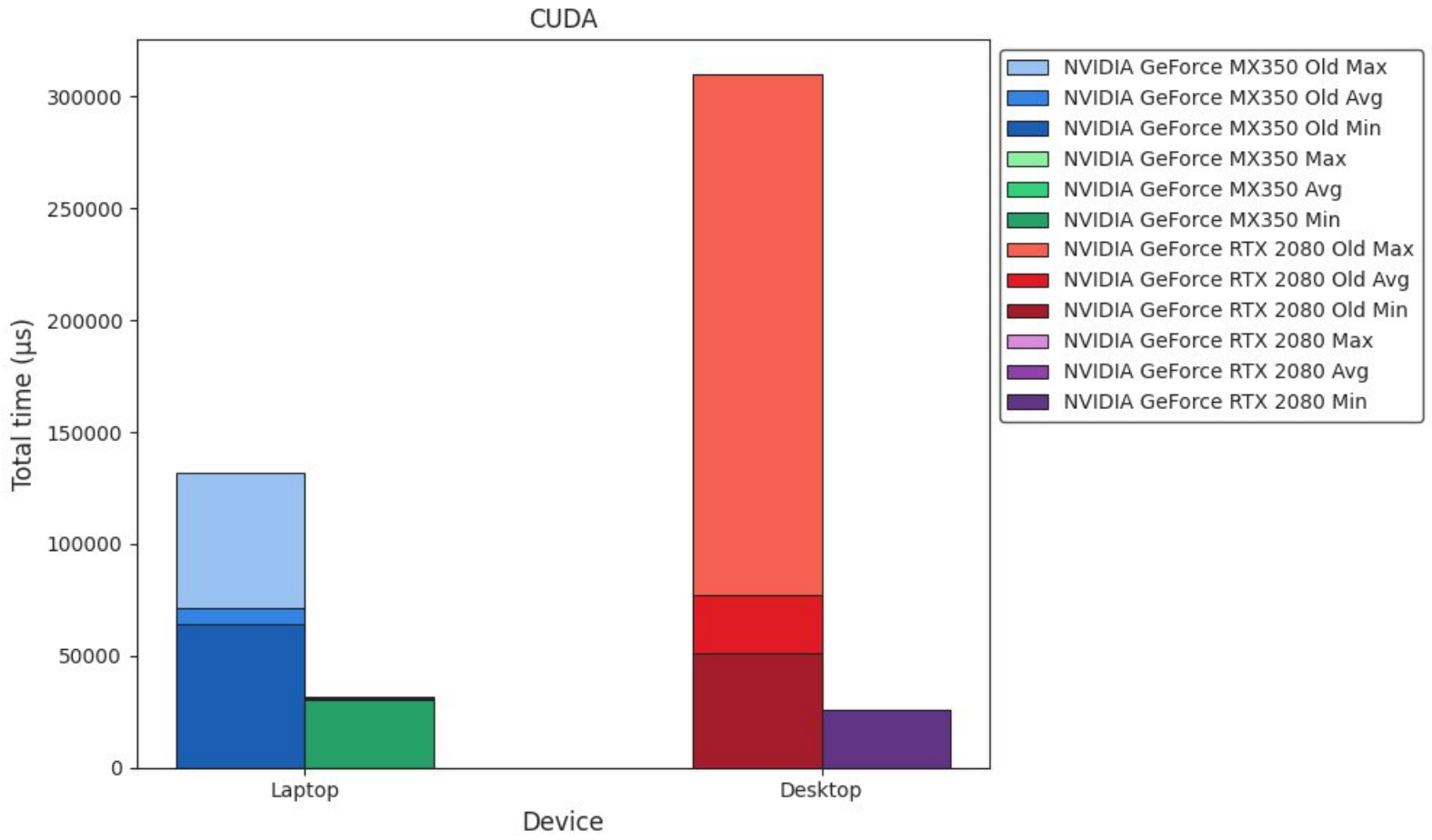
CUDA Tuning

Profiling with nvprof

- **Key Takeaways**
  - **MNIST:**
    - 34% matmul
    - 18% conv2d
    - 12% maxpool, biasing, relu
  - **XL:**
    - 94% matmul
    - <2% other functions
- **Key Insights**
  - matmul becomes the bottleneck in XL
- **Explanation**
  - **matmul complexity:** $O(n^3)$
  - **Other functions:** $O(n^2)$
  - **Scale factor:** 32
  - **Growth Factor:**
    - **matmul:** 32,768
    - **Other functions:** 1024
- **Goal**
  - Improve matmul computation speed

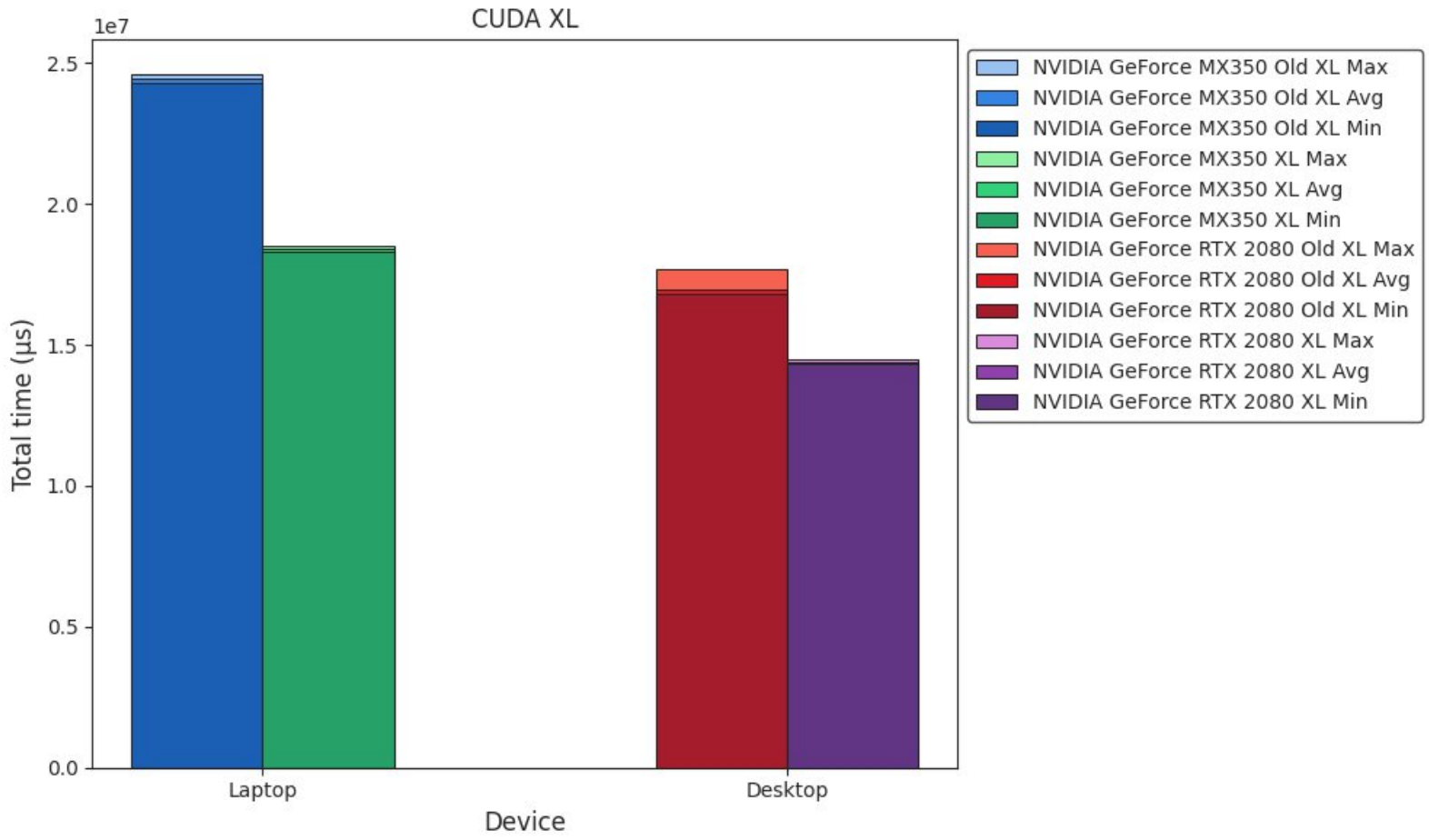
Benchmark

CUDA



Benchmark

CUDA XL



Multithreading & SIMD

Multithreading

- **Fixed Memory Leak**
  - **Change:** Resolved a memory leak that had existed for a long time
  - **Benefit:** Reduced memory usage
- **Enhanced Poison Pill**
  - **Change:** Removed costly if comparisons
  - **Benefit:** Direct thread termination
- **Optimized mt_arg**
  - **Change:**
    - Reduced the size of mt_arg
    - Moved mt_arg to its own header file
  - **Benefit:** Lower memory overhead and improved modularity
- **Enhanced Multithreading**
  - **Change:** Improved smart multithreading and thread distribution
  - **Benefit:** Better performance and workload balance

Before:

```
void create_mt(long threads) {
    ...
    mt_arg *mt = (mt_arg*)malloc(THREADS * sizeof(mt_arg));
    for(long i = 0; i < THREADS; i++) {
        mt[i].idx = i;
        pthread_create(&tids[i], NULL, start_mt, &mt[i]);
    }
}

static void *start_mt(void *arg) {
    mt_arg *mt = (mt_arg*)arg;
    while(1) {
        mt_arg *head = (mt_arg*)g_async_queue_pop(queue);
        if(head->start_routine == stop_mt) {
            break;
        }
        head->idx = mt->idx;
        head->start_routine(head);
    }
    return NULL;
}
```

Multithreading & SIMD

Multithreading

- **Fixed Memory Leak**
  - **Change:** Resolved a memory leak that had existed for a long time
  - **Benefit:** Reduced memory usage
- **Enhanced Poison Pill**
  - **Change:** Removed costly if comparisons
  - **Benefit:** Direct thread termination
- **Optimized mt_arg**
  - **Change:**
    - Reduced the size of mt_arg
    - Moved mt_arg to its own header file
  - **Benefit:** Lower memory overhead and improved modularity
- **Enhanced Multithreading**
  - **Change:** Improved smart multithreading and thread distribution
  - **Benefit:** Better performance and workload balance

After:

```
__attribute__((always_inline)) inline void create_mt(int threads) {
    ...
    int idx[THREADS];
    for(int i = 0; i < THREADS; i++) {
        idx[i] = i;
        pthread_create(&tids[i], NULL, start_mt, &idx[i]);
    }
    wait_mt();
}

__attribute__((always_inline)) inline static void *start_mt(void *arg) {
    int idx = *(int*)arg;
    wait_mt();
    while(1) {
        mt_arg *head = (mt_arg*)g_async_queue_pop(queue);
        head->idx = idx;
        head->start_routine(head);
    }
    return NULL;
}
```

Multithreading & SIMD

Multithreading

- **Fixed Memory Leak**
  - **Change:** Resolved a memory leak that had existed for a long time
  - **Benefit:** Reduced memory usage
- **Enhanced Poison Pill**
  - **Change:** Removed costly if comparisons
  - **Benefit:** Direct thread termination
- **Optimized mt_arg**
  - **Change:**
    - Reduced the size of mt_arg
    - Moved mt_arg to its own header file
  - **Benefit:** Lower memory overhead and improved modularity
- **Enhanced Multithreading**
  - **Change:** Improved smart multithreading and thread distribution
  - **Benefit:** Better performance and workload balance

Before (mt.hpp):

```
typedef struct mt_arg {
    long idx;
    matrix **a_ptr;
    matrix *a;
    matrix **b_ptr;
    matrix *b;
    int len;
    matrix **c_ptr;
    matrix *c;
    ...
} mt_arg;
```

After (mt_arg.hpp):

```
typedef struct mt_arg {
    int idx;
    matrix **a;
    matrix **b;
    matrix **c;
    int len;
    ...
} mt_arg;
```

Multithreading & SIMD

Multithreading

- **Fixed Memory Leak**
  - **Change:** Resolved a memory leak that had existed for a long time
  - **Benefit:** Reduced memory usage
- **Enhanced Poison Pill**
  - **Change:** Removed costly if comparisons
  - **Benefit:** Direct thread termination
- **Optimized mt_arg**
  - **Change:**
    - Reduced the size of mt_arg
    - Moved mt_arg to its own header file
  - **Benefit:** Lower memory overhead and improved modularity
- **Enhanced Multithreading**
  - **Change:** Improved smart multithreading and thread distribution
  - **Benefit:** Better performance and workload balance

Before:

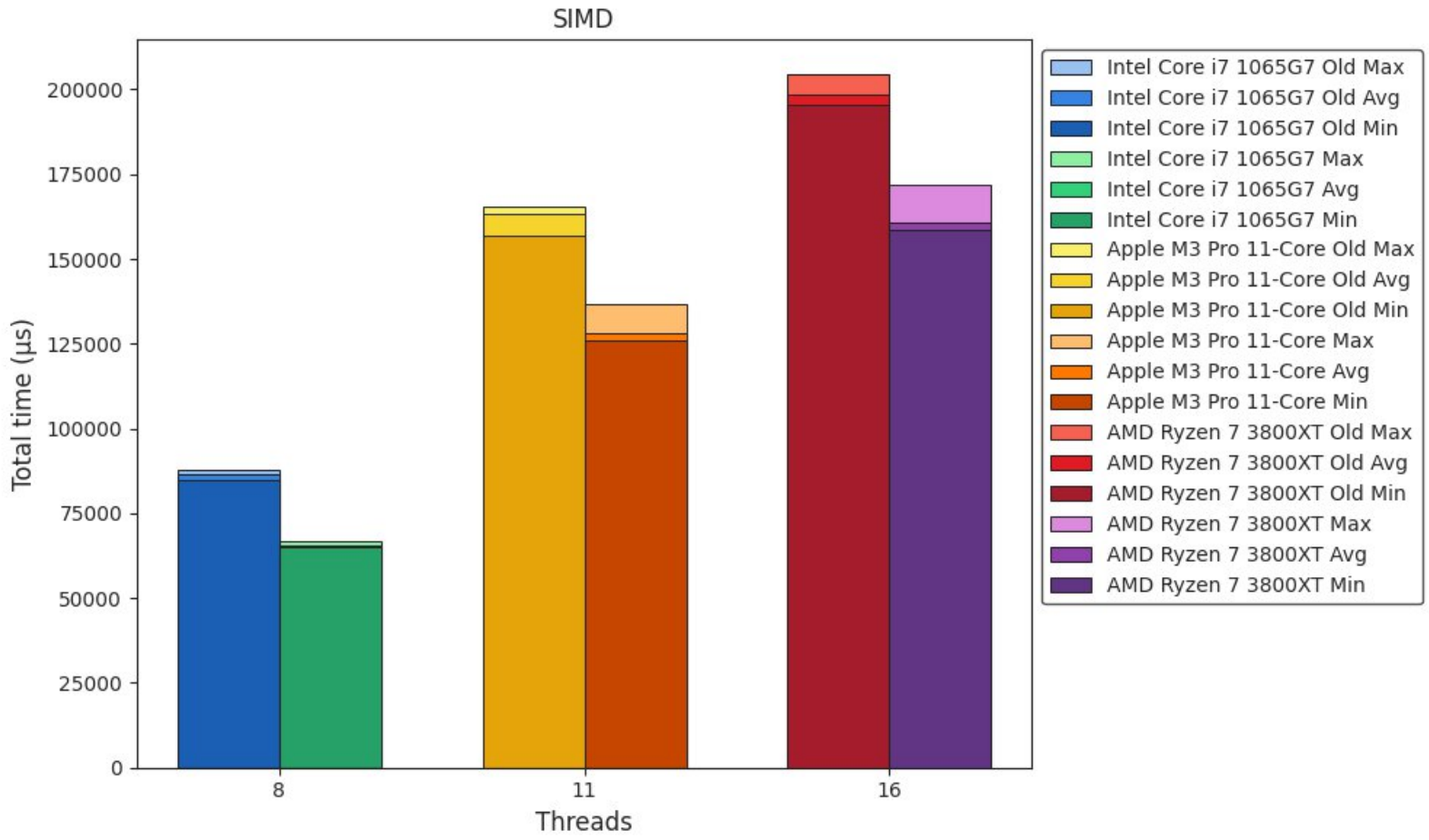
```
void add_mt(mt_arg *mt) {
    ...
    for(int i = mt->idx; i < mt->c->x; i += THREADS) {
        mt->i = i;
        add_simd(mt);
    }
    wait_mt();
}
```

After:

```
__attribute__((always_inline)) inline void add_mt(mt_arg *mt) {
    ...
    for(int i = mt->idx * ((*mt->c)->x / THREADS);
i < ((mt->idx + 1) * ((*mt->c)->x / THREADS)) +
(mt->idx == THREADS - 1 ? (*mt->c)->x % THREADS : 0);
i++) {
        mt->i = i;
        add_simd(mt);
    }
    wait_mt();
}
```

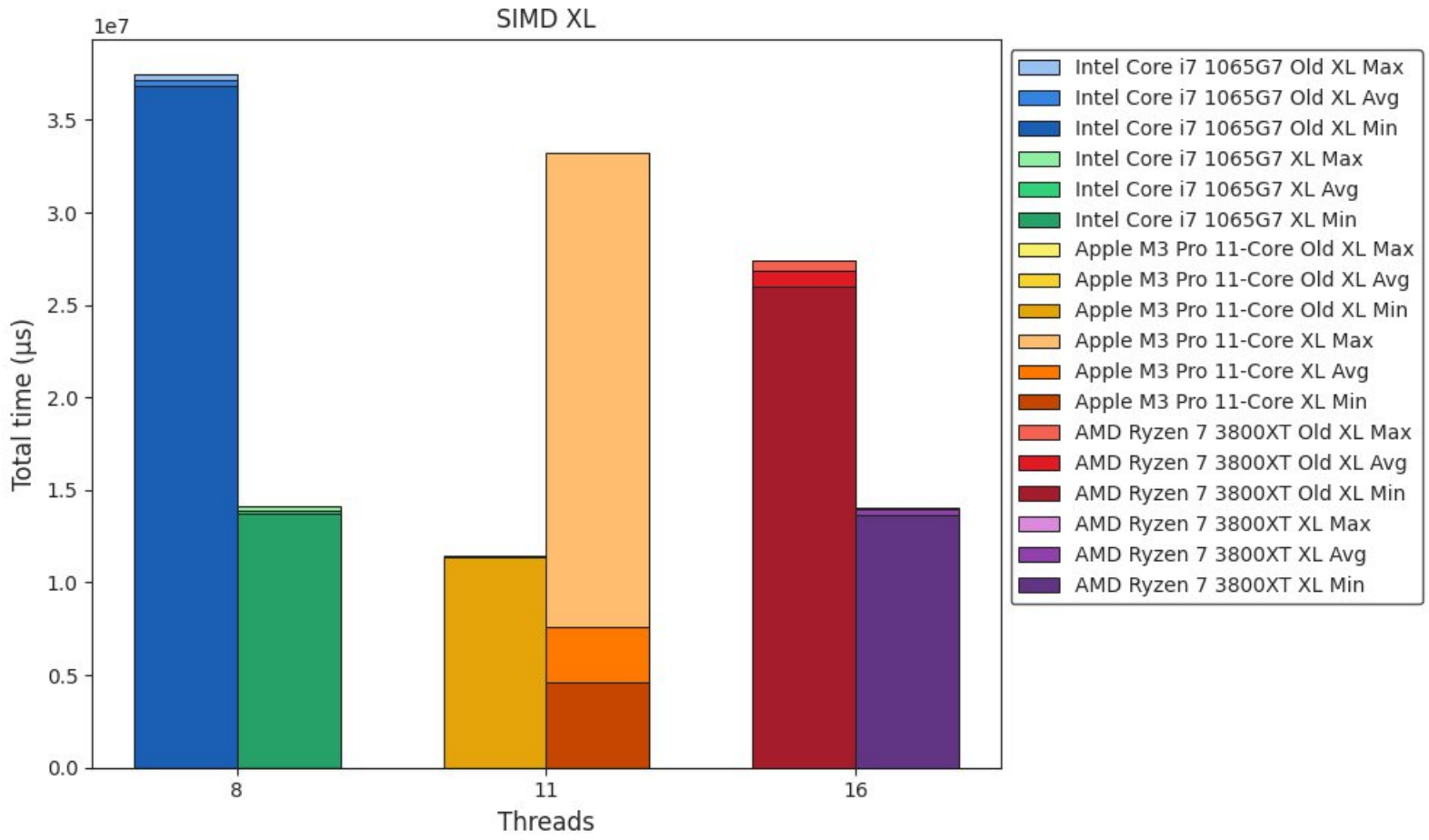
Benchmark

SIMD



Benchmark

SIMD XL



Multithreading & SIMD

AMX (1 / 2)

- **Progress**
  - Fixed **void matmul_simd**(mt_arg *mt);
  - Implemented **void conv2d_simd**(mt_arg *mt);
- **AMX Registers**
  - **X (or Y) Registers:**
    - 4 source registers for computations
    - Containing 64 floating-point values in total
  - **Z Registers:**
    - Serves as the accumulator or output registers
    - Holds a 64x64 floating-point matix
- **AMX Instructions**
  - **Defined in aarch64.h:**
    - Implemented as macros for assembly instructions
    - Utilize bitmasks (bm) to define instruction behavior
  - **Key instructions:**
    - **AMX_SET**(); and **AMX_CLR**();
      Frame an AMX instruction block
    - **AMX_LDX**(bm); or **AMX_LDY**(bm);
      Load values into the X or Y registers
    - **AMX_STZ**(bm);
      Stores values from the Z registers into memory
    - **AMX_FMA32**(bm);:
      Performs fused multiply add
      `(z[j][i] += x[i] * y[j])`

Multithreading & SIMD

AMX (2 / 2)

- **Example AMX Instruction**
  - Extract of **void matmul_simd**(mt_arg *mt);

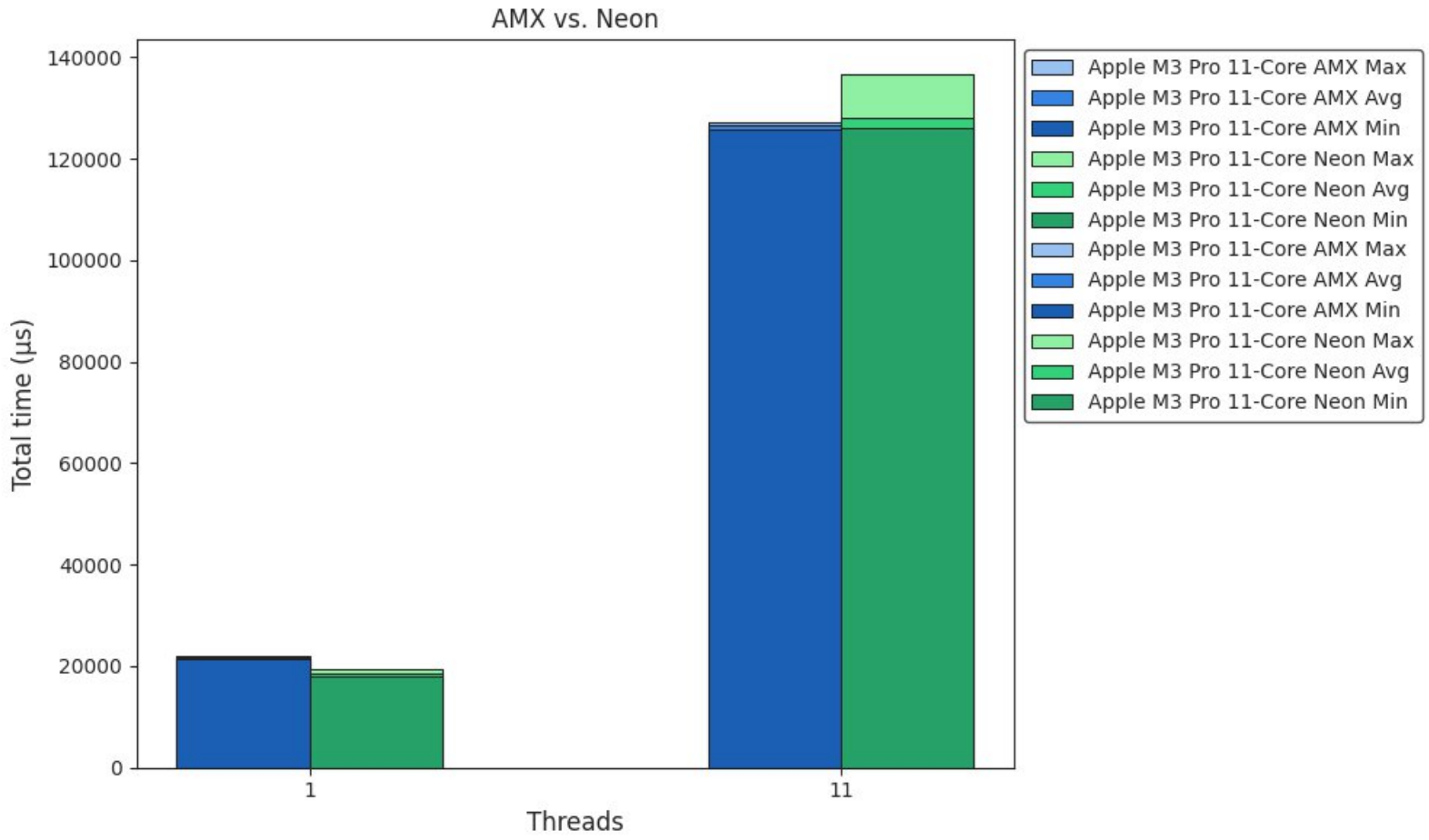
- **Power Consumption**
  - Measured with powermetrics (average values)

Benchmark	mW
Neon	7331
AMX	7778
Neon XL	7340
AMX XL	7218

```
uint64_t ldx = (uint64_t)&(*mt->a)->m[get_idx(mt->i, 0, (*mt->a)->y)];
ldx = ldx | 1ull << 60; // four registers
ldx = ldx | 1ull << 62; // multiple registers
uint64_t fma32 = 1ull << 63; // vector mode
uint64_t stz = (uint64_t)&z_reg;
// ...
for(int k = 0; k + CHUNK_SIZE - 1 < (*mt->a)->y; k += CHUNK_SIZE) {
        size_t k_offset = k * sizeof(DATA_TYPE);
        AMX_LDX(ldx + k_offset);
        AMX_LDY(ldy + k_offset);
         for(int i = 0; i < CHUNK_SIZE / LANE_SIZE; i++) {
                size_t i_offset = LANE_SIZE * i * sizeof(DATA_TYPE);
                AMX_FMA32(fma32 + i_offset + (i_offset << 10)); // x and y offset
         }
}
AMX_STZ(stz);
AMX_CLR():
```

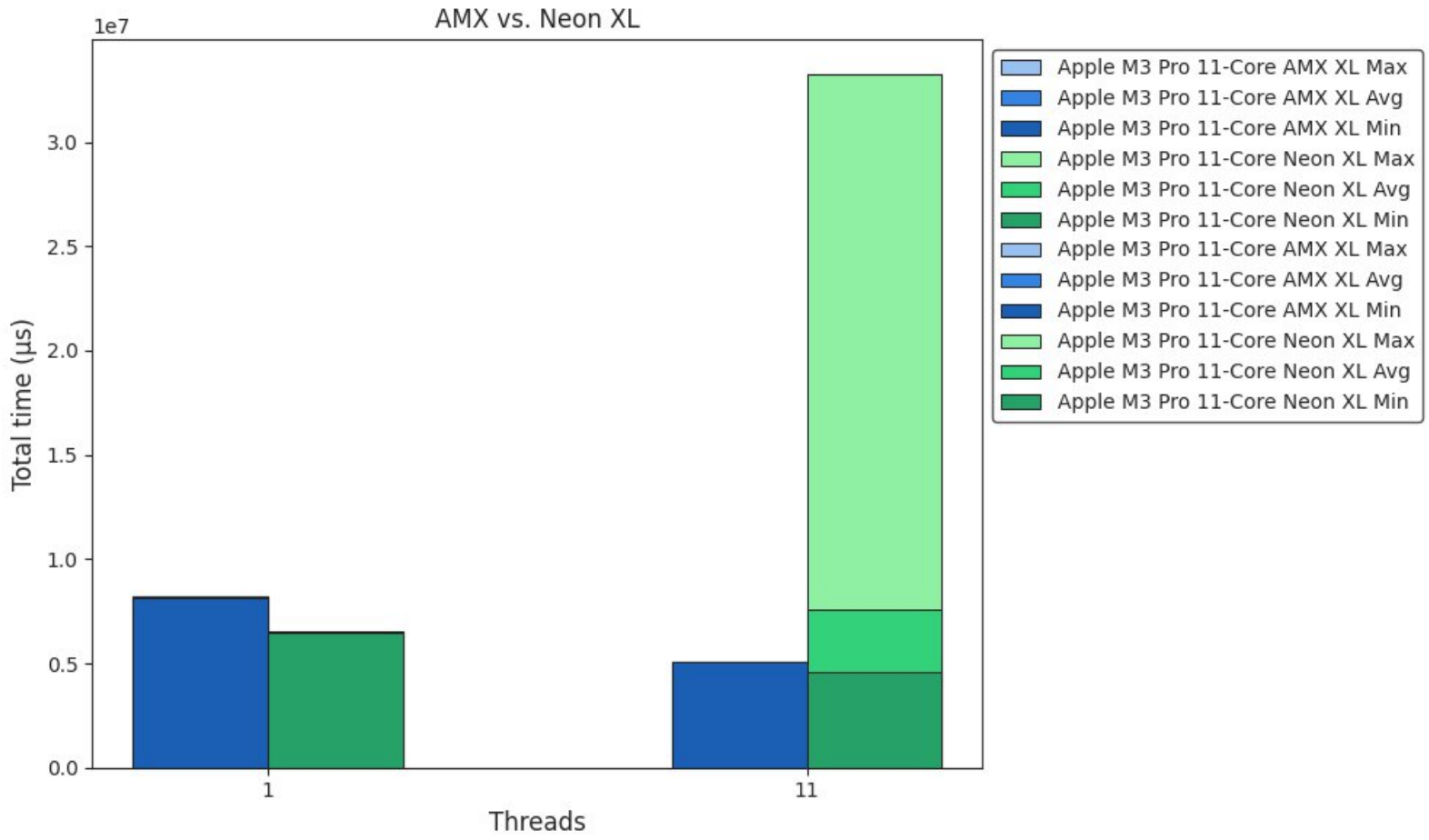
Benchmark

AMX vs. Neon



Benchmark

AMX vs. Neon XL



Quantization

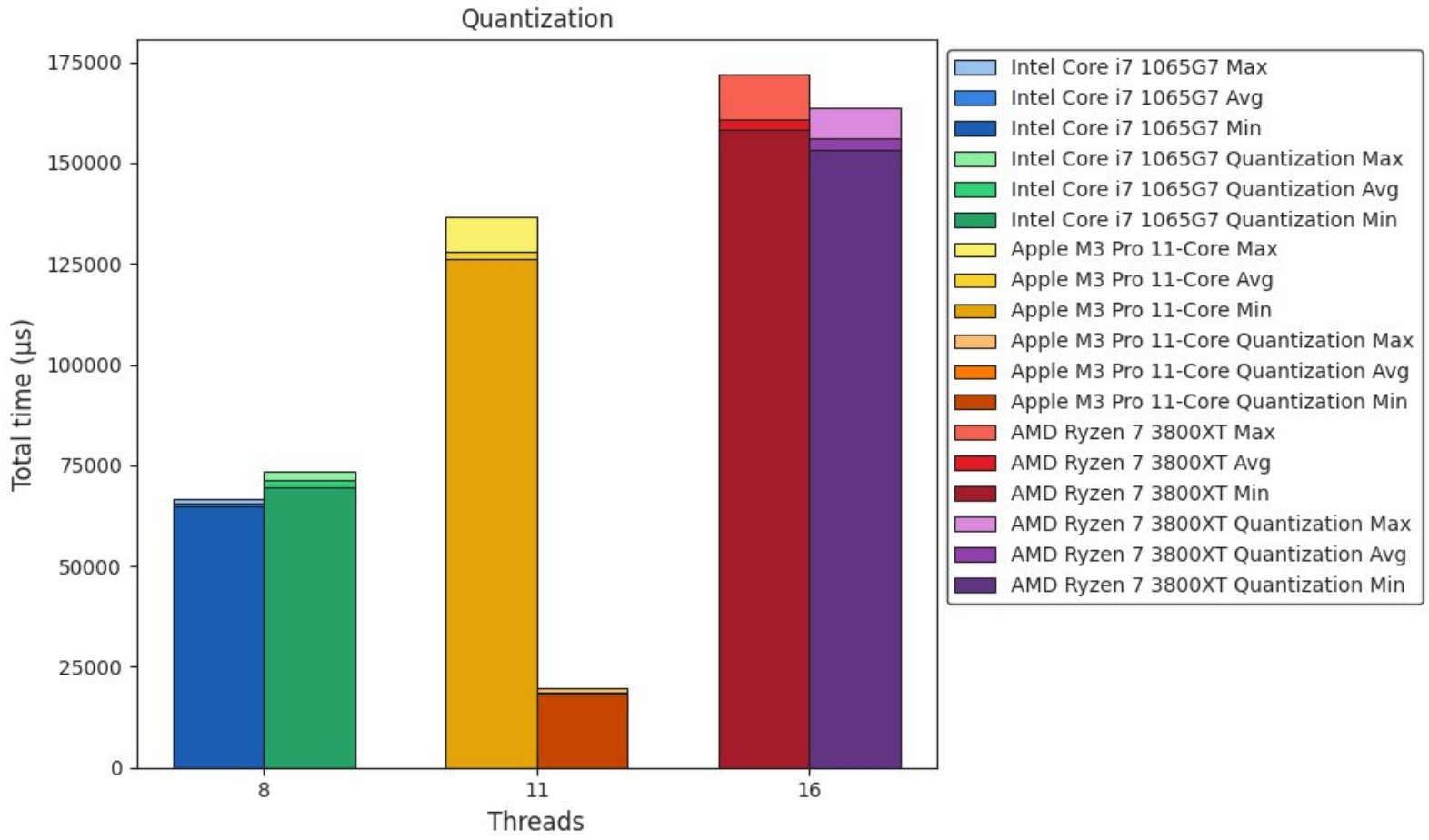
Overview

- **Data Type Optimization**
  - **Change:** float32 → int8
  - **Benefit:** Standard for AI/ML quantization, enabling greater efficiency with minimal accuracy loss
- **SIMD Integration**
  - **Change:** Added support for int8 in AVX-512, AVX2, and Arm Neon
  - **Benefit:** Process even more elements in parallel, significantly boosting computation speed
- **Memory Reduction**
  - **Change:** Model size reduced by ~10% (282.8MB → 253.7MB)
  - **Benefit:** Frees up memory on resource-limited devices

```
__attribute__((always_inline)) inline void matmul_simd(mt_arg *mt) {
  #ifdef x86
    #ifdef __AVX512F__
      #ifdef INT
        long CHUNK_SIZE = sizeof(__m256i) / sizeof(DATA_TYPE);
        __m256i a, b;
        __mmask32 m;
        for(int k = 0; k < (*mt->a)->y; k += CHUNK_SIZE) {
          m = (__mmask32)((1 << (((k + CHUNK_SIZE) <= (*mt->a)->y) ?
CHUNK_SIZE : (*mt->a)->y - k)) - 1);
          a = _mm256_maskz_loadu_epi8(m, &amp;(*mt->a)->m[get_idx(mt->i,
k, (*mt->a)->y)]);
          b = _mm256_maskz_loadu_epi8(m, &amp;(*mt->b)->m[get_idx(mt->j,
k, (*mt->b)->y)]);
          (*mt->c)->m[get_idx(mt->i, mt->j, (*mt->c)->y)] +=
_mm256_reduce_add_epi16(_mm256_mullo_epi16(_mm256_cvtepi8_epi16(_mm
256_extractf128_si256(a, 0)),
_mm256_cvtepi8_epi16(_mm256_extractf128_si256(b, 0))));
          (*mt->c)->m[get_idx(mt->i, mt->j, (*mt->c)->y)] +=
_mm256_reduce_add_epi16(_mm256_mullo_epi16(_mm256_cvtepi8_epi16(_mm
256_extractf128_si256(a, 1)),
_mm256_cvtepi8_epi16(_mm256_extractf128_si256(b, 1))));
        }
        ...
}
```

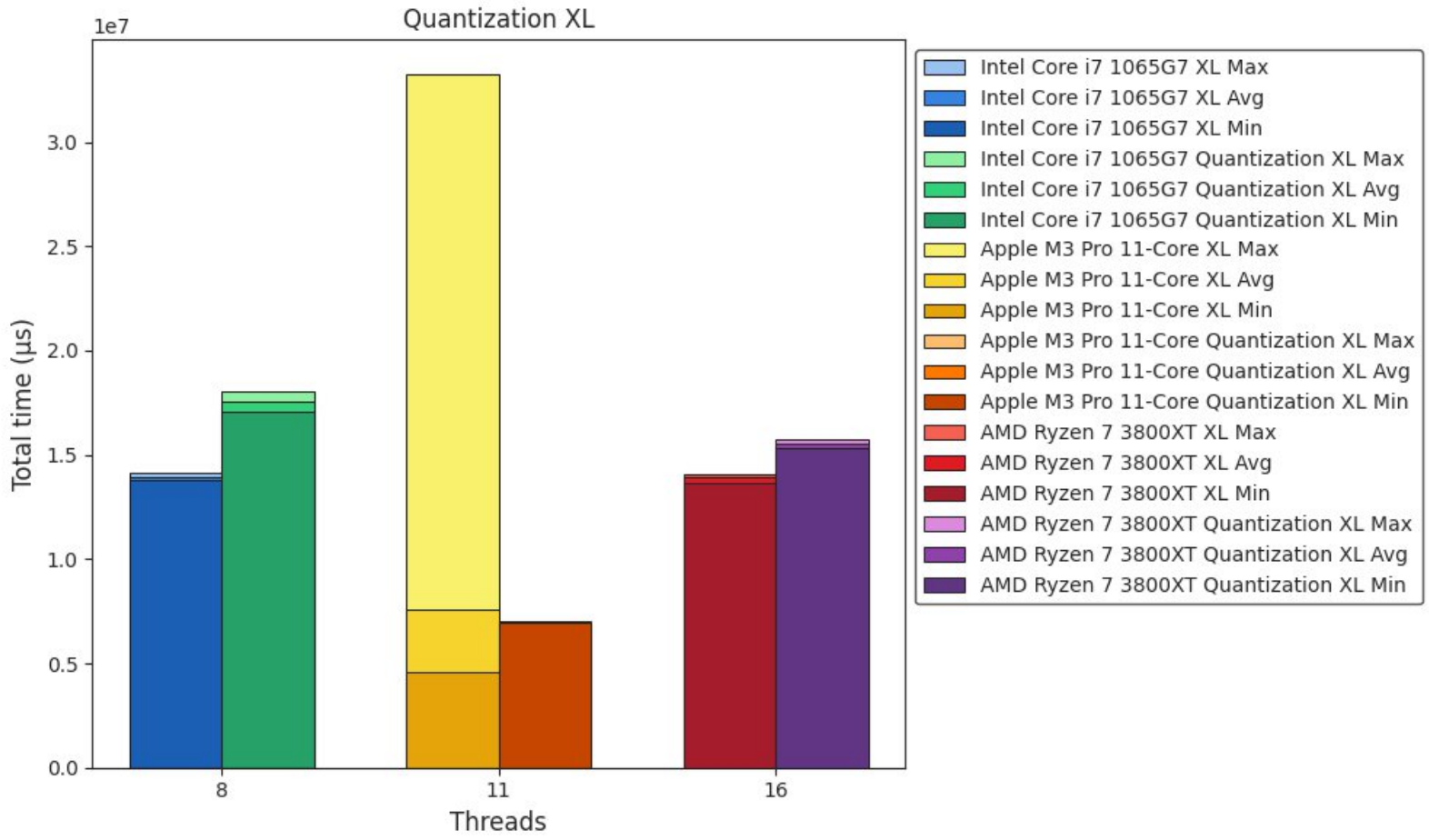
Benchmark

Quantization



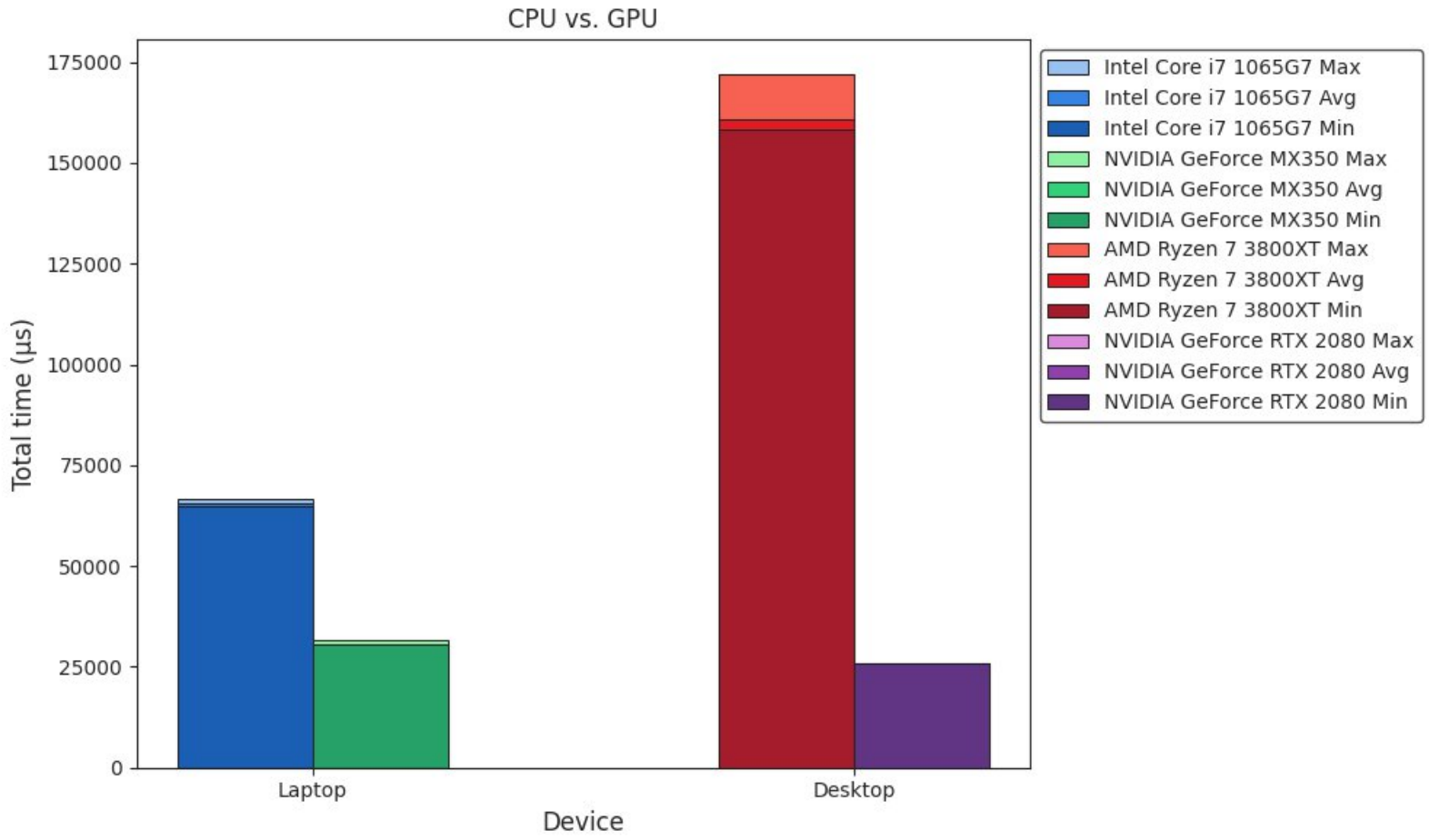
Benchmark

Quantization XL



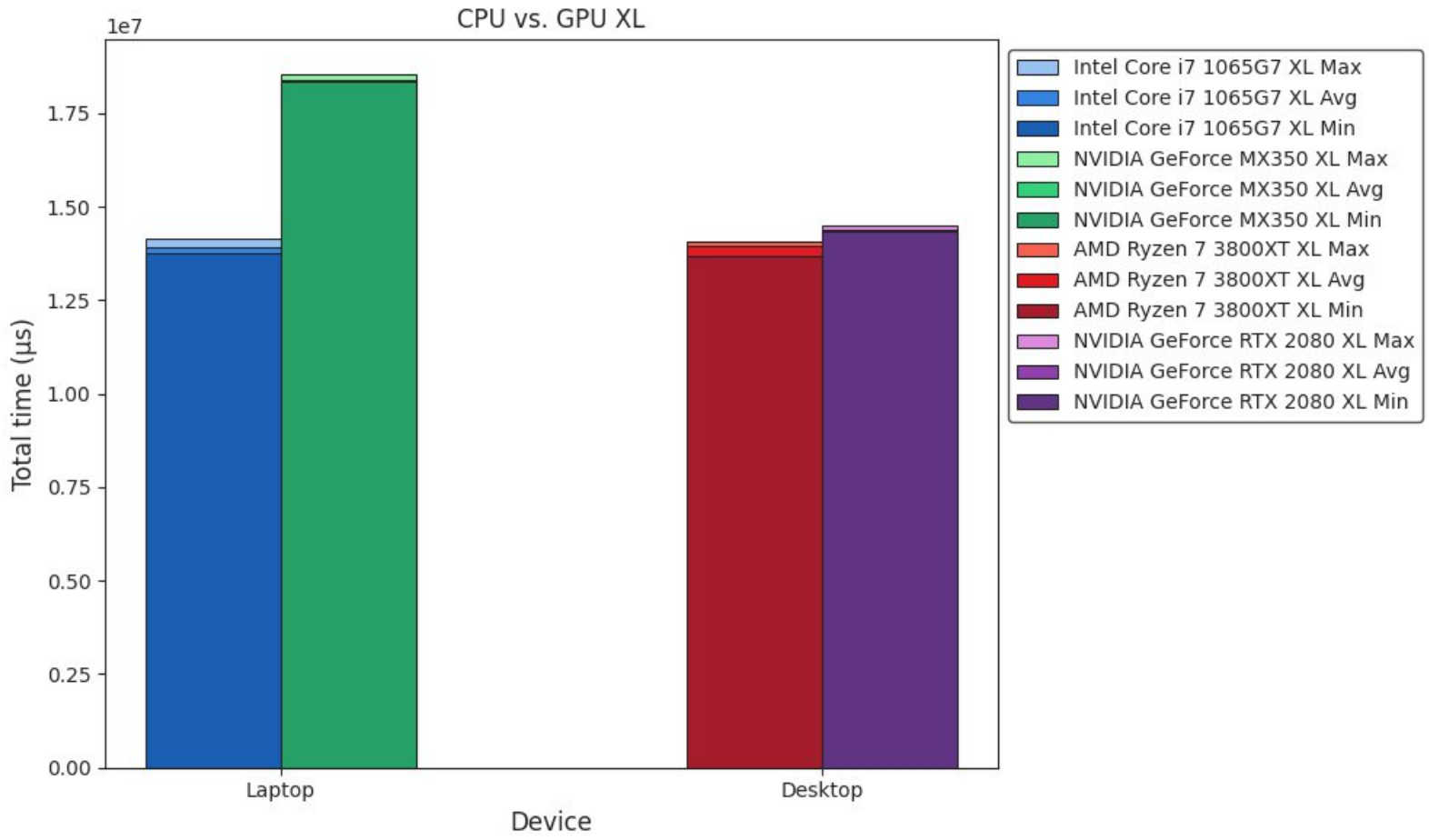
Benchmark

CPU vs. GPU



Benchmark

CPU vs. GPU XL



Outlook

Work In Progress

- Framework ✔
- ICPX vs. Clang ✔
- CUDA Tuning ✔
  - Improve matmul performance
- Multithreading ✔
  - Multithread images
  - OpenMP GPU offload target
- SIMD ✔
- Quantization 🚧
  - Apple AMX
  - Quantization layer