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

4. Presentation

01 Tweaks & Enhancements

02 Hardware & Benchmark

03 CUDA Tuning

04 SIMD

05 Outlook

Tweaks & Enhancements

1 / 2

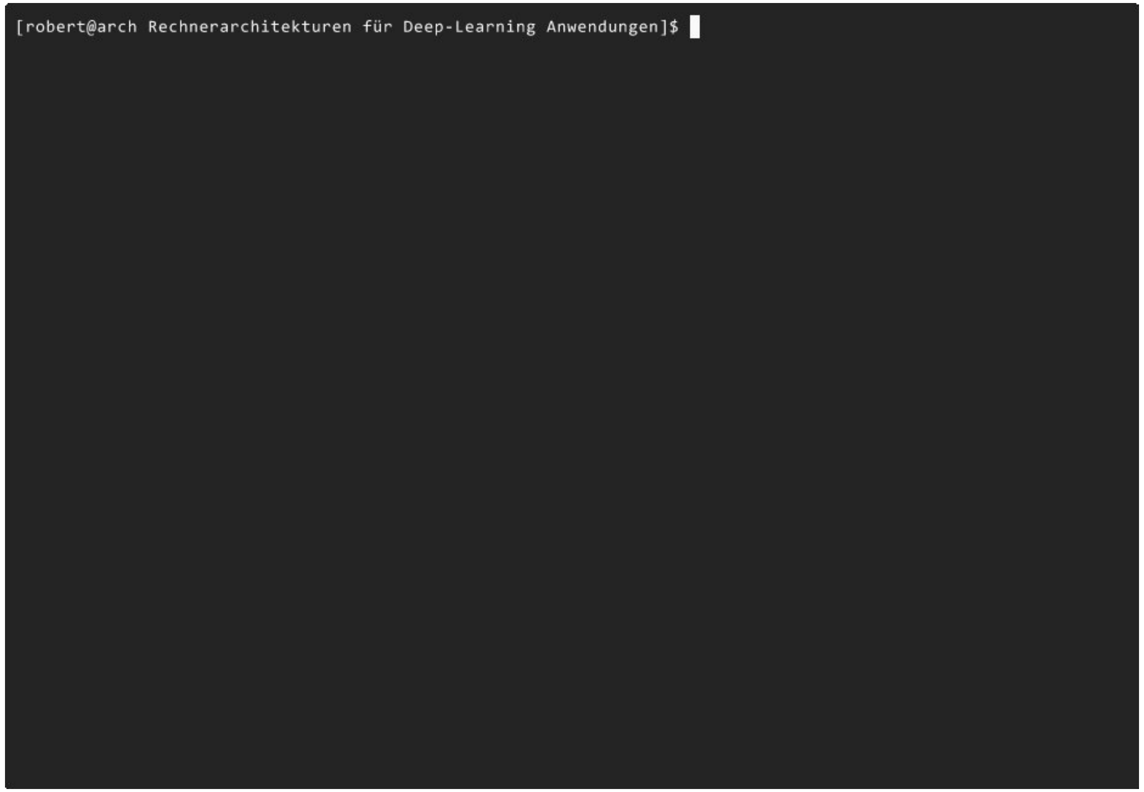
- **Inlining**
  - **Goal:** Minimize overhead caused by function calls
  - **Solution:** Inlined all utility functions
- **Data Type Flexibility**
  - **Goal:** Enable the use of integers when working with quantized weights
  - **Solution:** Began implementing support for diverse data types
- **Smart Multithreading Implementation**
  - **Goal:** Mitigate unnecessary synchronization overhead
  - **Solution:** Applied multithreading only under significant load

```
__attribute__((always_inline)) inline int get_idx(int i, int j, int y);

#ifndef DATA_TYPE
   #ifdef INT
      typedef int DATA_TYPE;
   #else
      typedef float DATA_TYPE;
   #endif
 #endif
typedef struct matrix {
   int x;
   int y;
   DATA_TYPE *m;
 } matrix;

if(THREADS > c->y) {
   single_core = 1;
}
```

Tweaks & Enhancements

2 / 2

– Implemented make config for a user-friendly option selection



Hardware & Benchmark

Hardware Overview

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 Overview

- **Batch size:** 1
- **Epochs:** 128
- **ICPX:**
  - In microseconds (µs)
  - Averaged over 10 runs
  - Intel oneAPI C++ Compiler
- **Old:**
  - In microseconds (µs)
  - Averaged over 10 runs
  - Total time (last presentation)
- **XL:**
  - In microseconds (µs)
  - Averaged over 10 runs
  - Image dimensions of $(\mathbf{32}\text{x}30)^2$

CUDA Tuning

1 / 2

- **Finalized CUDA Implementation**
  - **Change:** Adjusted maxpool output dimensions to ensure compatibility with CUDA kernels
  - **Benefit:** Implemented the last missing function
    - `matrix *flatten(matrix *a, int len, matrix *c);`
- **Increased CUDA Thread Count**
  - Optimized utilization of the massive parallelism offered by GPU cores
- **GPU Memory Management**
  - Enabled main function to allocate matrices directly in GPU memory
- **IO Matrices Copied To GPU In Main Function**
  - Moved allocations outside the main loop to enhance performance and centralization

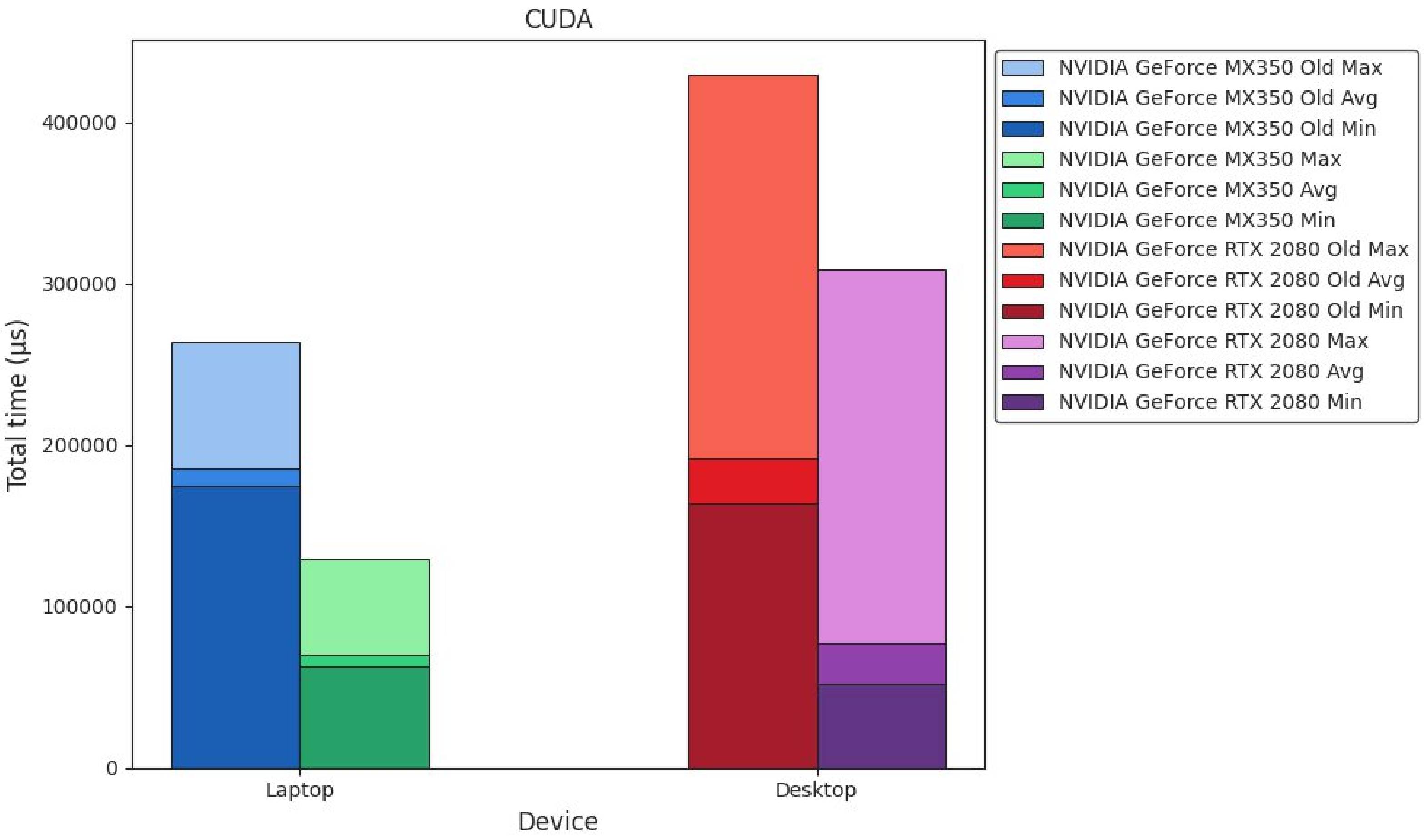
CUDA Tuning

2 / 2

- **Performance Bottlenecks**
  - Frequent use of cudaMalloc and cudaMemcpy for calculation matrices
  - Significant overhead impacts overall performance
- **Proposed Improvements**
  - Further centralize memory allocation
  - Eliminate memory transfer overhead during calculations

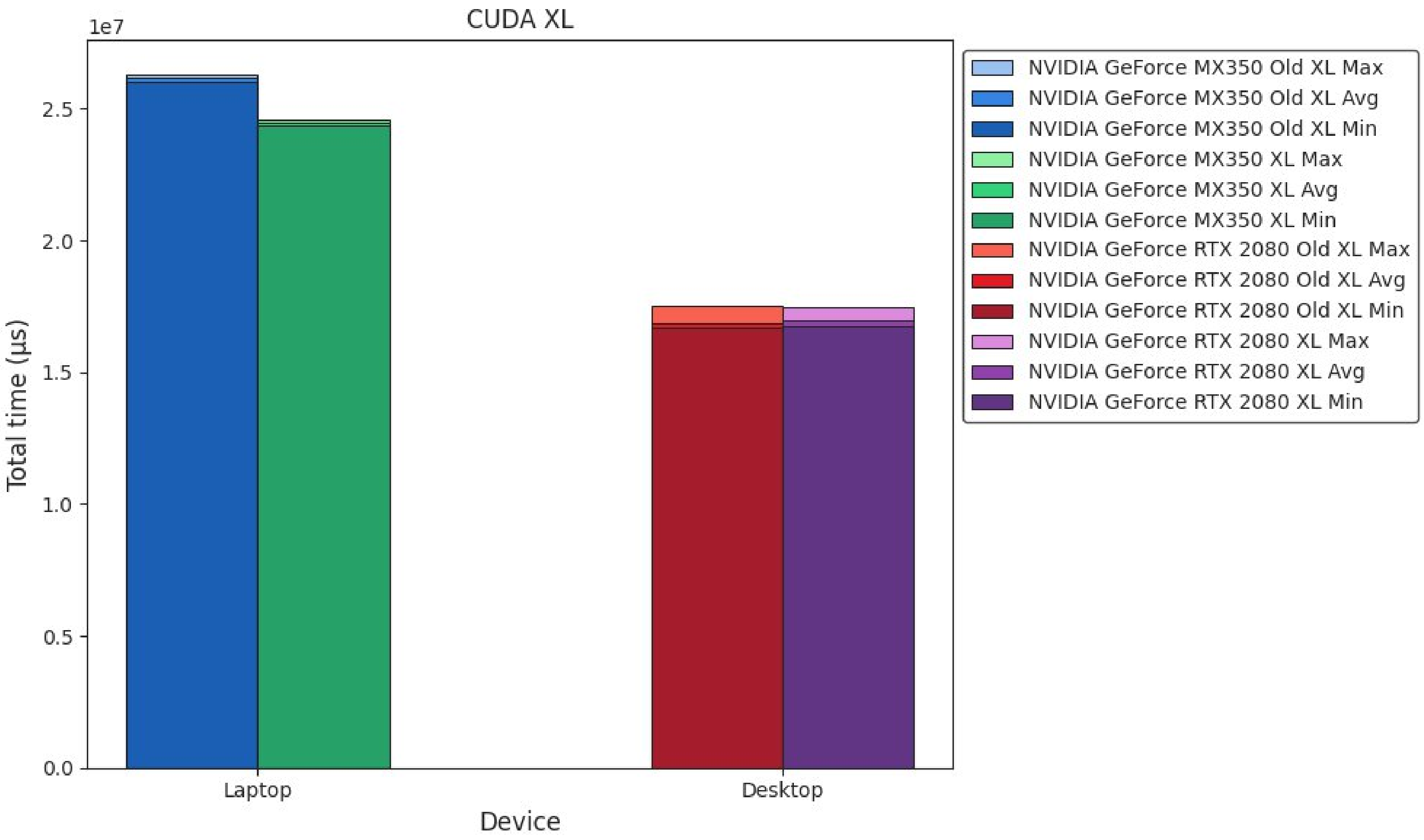
Benchmark

CUDA



Benchmark

CUDA XL



SIMD

ARM

- **Progress**
  - Added support for AMX
  - Implemented **void matmul_simd**(mt_arg *mt); using AMX

- **Apple AMX Instruction Set**
  - Integrated into Apple Silicon
  - Specialized matrix multiplication engine
  - Optimized for AI and ML tasks

- **Benefits**
  - Accelerates matrix-heavy operations
  - Improves performance
  - Type flexibility (floats and integers)

- **Challenges**
  - **Reverse Engineering:**
    - Relying on a personal GitHub repository
  - **Lack Of Documentation:**
    - Difficult implementation
    - Difficult troubleshooting

SIMD

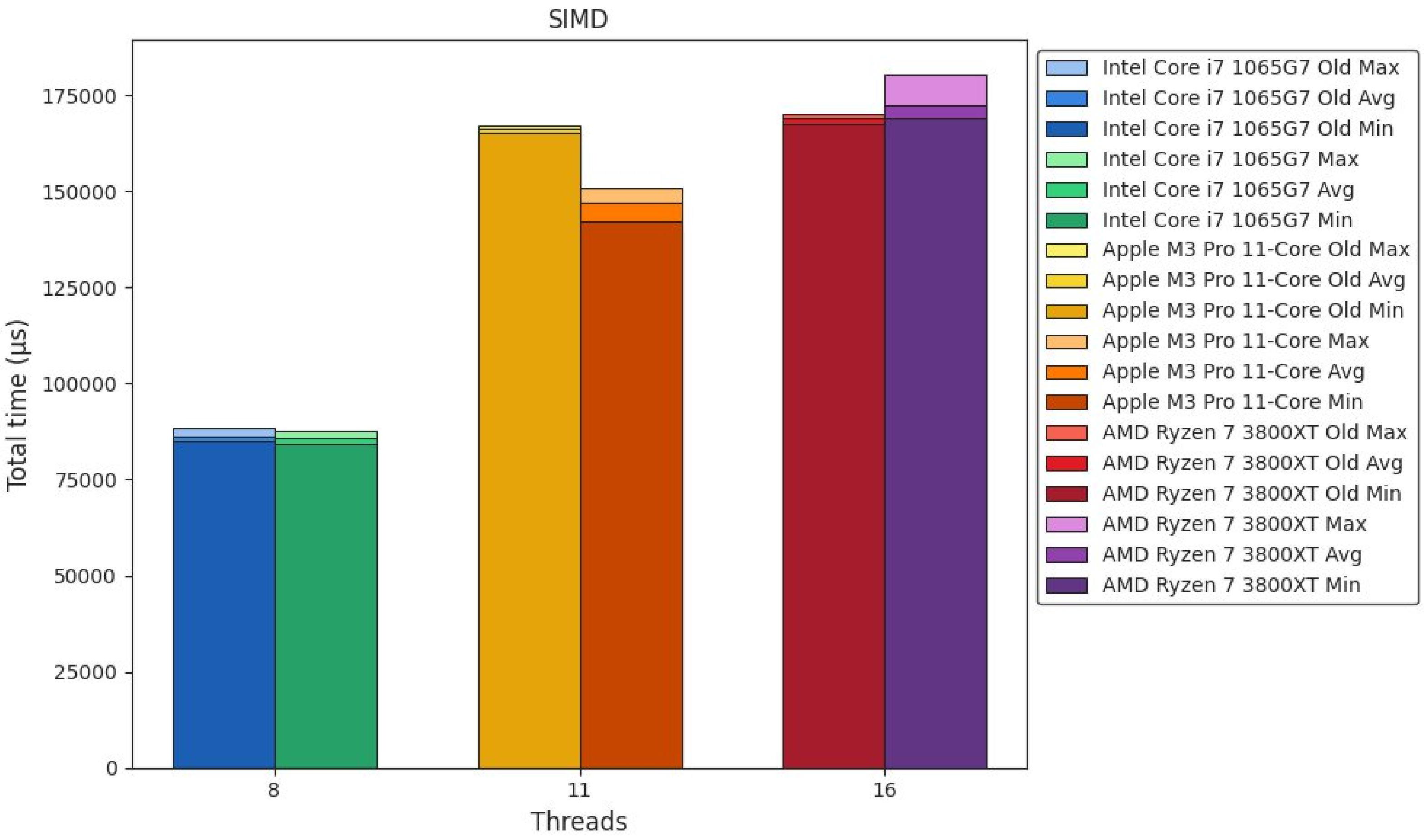
x86

- **Inlined SIMD**
  - **Change:** Inlined all SIMD functions
  - **Benefit:** Minimize function calls and reduce overhead
- **Transposed Inputs**
  - Reordered data for improved memory access efficiency
- **AVX-512 Implementation**
  - Added support to process 16 values at a time (instead of 4)
  - Added padding to ensure parallel processing, even when insufficient data is available
- **AVX2 Fallback**
  - Added AVX2 fallback for systems lacking AVX-512 support
  - Checks AVX-512 support at runtime for compatibility

```
__attribute__((always_inline)) inline void matmul_simd(mt_arg *mt) {
  #ifdef x86
    if(is_avx512_supported()) {
      long CHUNK_SIZE = sizeof(__m512) / sizeof(DATA_TYPE);
      __m512 a, b;
      __mmask16 m;
      for(int k = 0; k < mt->a->y; k += CHUNK_SIZE) {
        m = (__mmask16)((1 << (((k + CHUNK_SIZE) <= mt->a->y) ? CHUNK_SIZE : mt->a->y - k)) - 1);
        a = _mm512_maskz_loadu_ps(m, &amp;mt->a->m[get_idx(mt->i, k, mt->a->y)]);
        b = _mm512_maskz_loadu_ps(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)] += _mm512_reduce_add_ps(_mm512_mul_ps(a, b));
      }
    }
  #endif
}
```

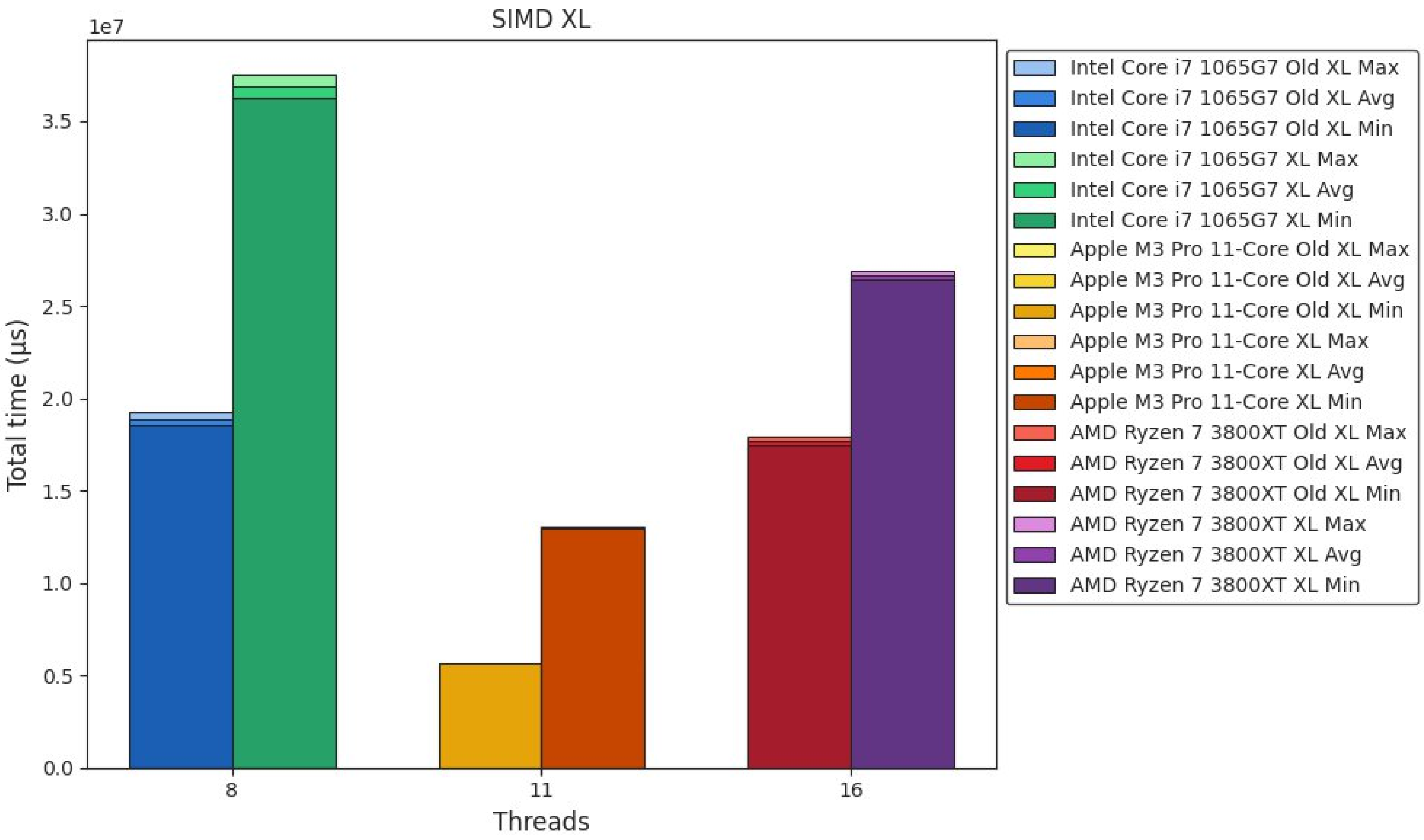
Benchmark

SIMD



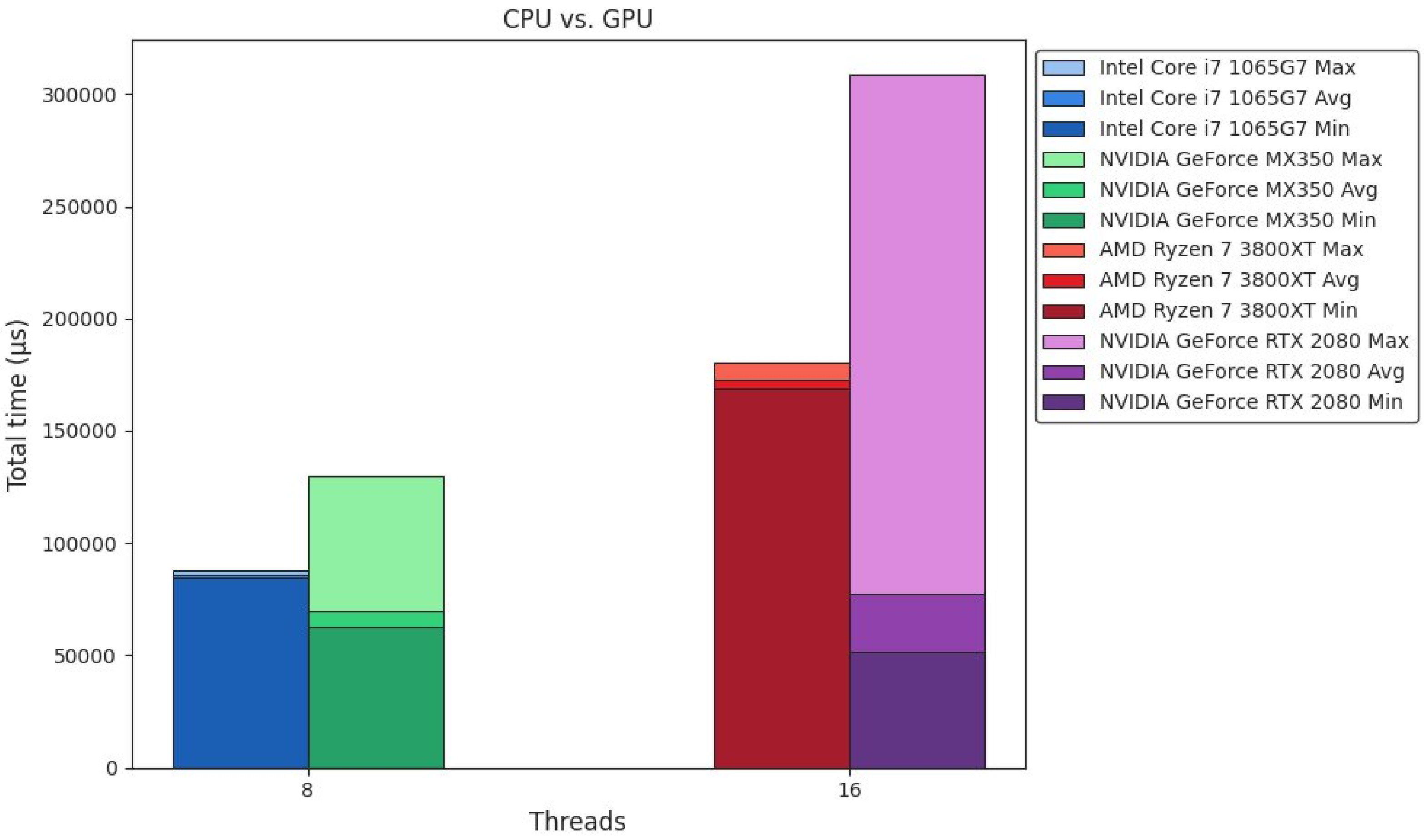
Benchmark

SIMD XL



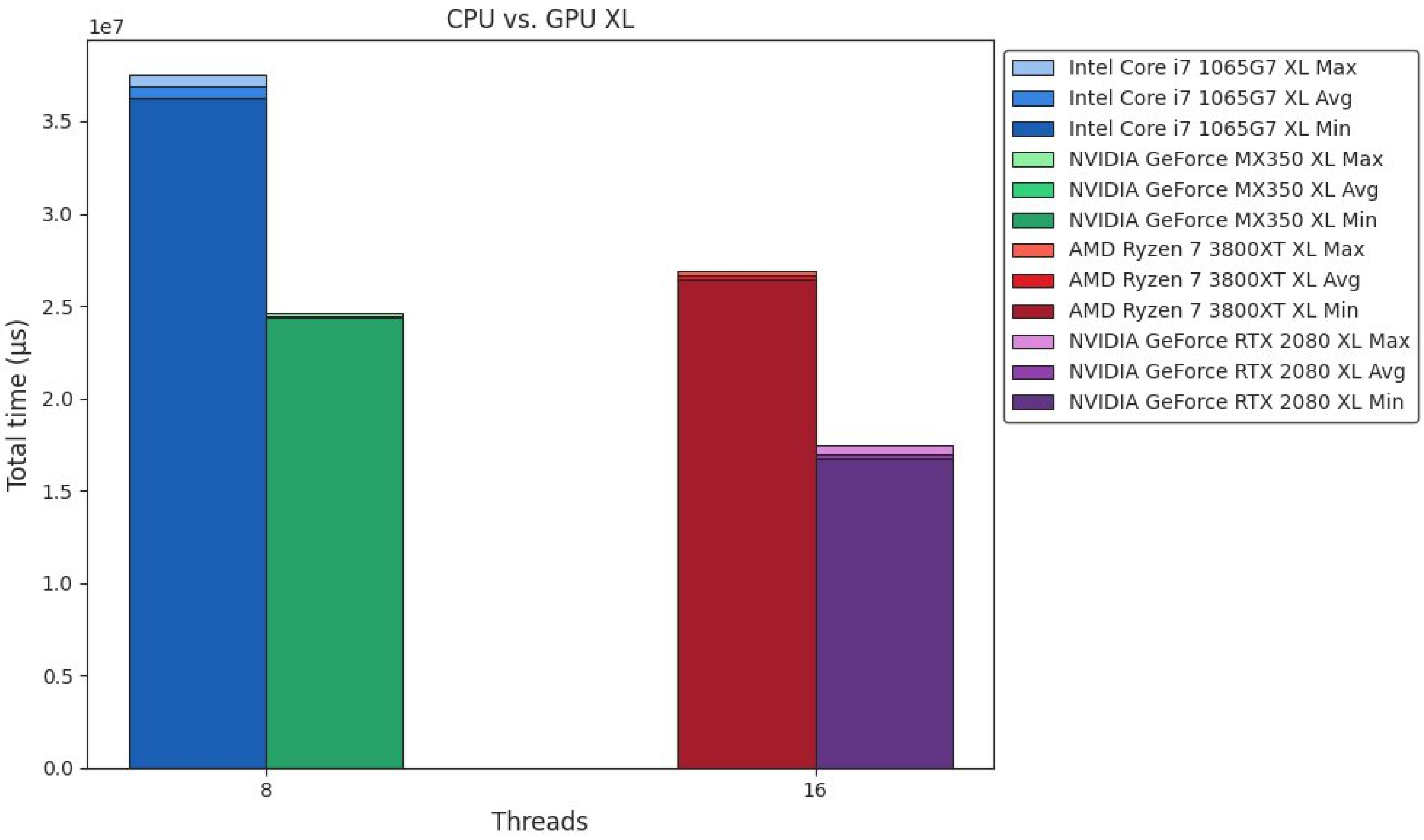
Benchmark

CPU vs. GPU



Benchmark

CPU vs. GPU XL



Outlook

Work In Progress

- Framework ✓
- ICPX vs. Clang ✓
- CUDA Tuning ✓
  - Eliminate memory transfer overhead (WIP)
- Multithreading ✓
  - (OpenMP GPU offload target)
- SIMD ✓
  - Apple AMX Instruction Set (WIP)
- Quantization 🚧
  - Prepare SIMD for integers (WIP)
- (Apple M3 Pro NPU)