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

3. Presentation

Tweaks & Enhancements

1 / 2

- **Addressed extern "C" compatibility issues:**
  - Resolved issues with **extern** "C" declarations in .c files
  - **Solution:** Migrated the entire codebase to C++
  - **Benefit:** Facilitated seamless integration with CUDA
- **Adopted flat array representation:**
  - Improved cache utilization for better performance
  - **Benefit:** Simplified integration with CUDA workflows
- **Optimized malloc usage in functions:**
  - Updated functions:
    - matrix ***add**(matrix *a, matrix *b); → matrix ***add**(matrix *a, matrix *b, matrix *c);
  - **Change:** If c is not NULL, the function bypasses malloc
  - **Benefit:** Allocations moved outside the main loop for enhanced performance

Tweaks & Enhancements

2 / 2

- **Transitioned from the GCC compiler to Clang:**
  - **Problem:** GCC did not function as expected on macOS
  - **Solution:** Clang offers excellent cross-platform support and is particularly optimized for macOS
  - **More differences:**

Feature	GCC	Clang
Performance	Generally considered highly optimized, with mature optimizations	Competitive performance, sometimes faster than GCC in certain benchmarks
Optimization	Extensive optimizations (O1, O2, O3, aggressive optimizations)	Similar set of optimizations, sometimes better diagnostics and debugging support
Use Cases	Widely used in embedded systems, scientific computing, Linux distributions	Preferred in environments like macOS, iOS, Android (via Google’s Android NDK)

Hardware & Benchmarks

Hardware Overview

CPU	Release dates	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 dates	TDP (W)	CUDA cores	Base Clock (MHz)
NVIDIA GeForce RTX 2080	20. September 2018	215	2944	151
NVIDIA GeForce MX350	10. February 2020	20	640	1354

Hardware & Benchmarks

Benchmark Overview

- **Batch size:** 1
- **Epochs:** 128
- **Performance:**
  - In microseconds (μs)
  - Averaged over 10 runs
  - Total time
- **ICPX:**
  - In microseconds (μs)
  - Averaged over 10 runs
  - Intel oneAPI C++ Compiler
- **XL:**
  - In microseconds (μs)
  - Averaged over 10 runs
  - Image dimensions of $(32x30)^2$

CUDA tuning

1 / 2

- Integrated with our framework
- **Implemented the functions:**
  - matrix ***add**(matrix *a, matrix *b);
  - matrix ****biasing**(matrix **a, **int** len, matrix *b);
  - matrix ****conv2d**(matrix *a, matrix **b, **int** len);
  - matrix ****flip_kernels**(matrix **a, **int** len);
  - matrix ****hyperbolic_tangent**(matrix **a, **int** len);
  - matrix ***matmul**(matrix *a, matrix *b);
  - matrix ****maxpool**(matrix **a, **int** len);
  - matrix ****relu**(matrix **a, **int** len);
  - matrix ***transpose**(matrix *a);

CUDA tuning

2 / 2

- **Current challenges:**
  - matrix *flatten(matrix **a, int len) is not feasible
  - Current framework prevents effective implementation
- **Performance bottlenecks:**
  - Each function requires cudaMalloc and cudaMemcpy
  - Significant overhead reduces overall performance
- **Proposed improvements:**
  - Centralized memory allocation
- **Exclusive use of device memory:**
  - Avoid reliance on host memory for computations
- **Eliminate memory transfer overhead:**
  - Optimize by only working with device memory

Benchmark

NVIDIA

Benchmark

NVIDIA XL

OpenMP

Implementation

- **OpenMP overview:**
  - Parallel programming support for C++ with minimal code changes
  - Efficient execution on multi-core processors
  - Compiler directives, library routines, and environment variables
- **Compiler flags:**
  - **OMP:** -Xcompiler -fopenmp -DOMP
  - **OMP_DARWIN:** -Xclang -fopenmp -DOMP
  - **OMP_INTEL:** -Xcompiler -qopenmp -DOMP
- **Implementation:**
  - Integrated OpenMP pragmas for parallel processing
- **Comparison:**
  - Multi-threaded implementation vs. native compiler multi-threading

```
matrix *matmul(matrix *a, matrix *b, matrix *c) {
  // ...
  #ifdef OMP
    #pragma omp parallel for collapse(2)
    for(int i = 0; i < c->x; i++) {
      for(int j = 0; j < c->y; j++) {
        c->m[get_idx(i, j, c->y)] = 0.0;
        #pragma omp simd
        for(int k = 0; k < a->y; k++) {
          c->m[get_idx(i, j, c->y)] += a->m[get_idx(i, k, a->y)] * b->m[get_idx(k, j, b->y)];
        }
      }
    }
  // ...
  #endif
  return c;
}
```

Benchmark

OpenMP

Benchmark

OpenMP XL

Benchmark

OpenMP vs. ICPX OpenMP

Benchmark

OpenMP vs. ICPX OpenMP XL

SIMD

Implementation

- **SIMD (Single Instruction Multiple Data) overview:**
  - Performs operations on multiple data elements in parallel
  - Accelerates machine learning by processing large datasets faster
- **Vector Size:** 128 bits, handling 4 floating-point values concurrently
- **Hardware Support:** Intel SSE3 (-Xcompiler -msse3), Arm Neon
- **Implemented Functions:**
  - void add_simd(mt_arg *mt);
  - void biasing_simd(mt_arg *mt);
  - void conv2d_simd(mt_arg *mt);
  - void matmul_simd(mt_arg *mt);

```
// ...
#if defined(__x86_64__) || defined(_M_X64) || defined(__i386) || defined(_M_IX86)
 #define x86
 #include <immintrin.h>
 // ...
#elif defined(__aarch64__) || defined(_M_ARM64) || defined(__arm__) || defined(_M_ARM)
 #define ARM
 #include <arm_neon.h>
#endif
// ...
#define VECTOR_SIZE (128)
#define CHUNK_SIZE (VECTOR_SIZE / 8 / sizeof(float))
// ...
  void add_simd(mt_arg *mt);
  void biasing_simd(mt_arg *mt);
  void conv2d_simd(mt_arg *mt);
  void matmul_simd(mt_arg *mt);
// ...
```

Benchmark

SIMD

Benchmark

SIMD XL

Outlook

Work in progress

- Framework ✓
- ICPX vs. Clang ✓
- CUDA tuning ✓
  - Eliminate memory transfer overhead
- Multithreading ✓
  - OpenMP GPU offload target
- SIMD ✓
  - Arm SVE
  - AVX-512
- Quantization
- (Apple M3 Pro NPU)