CMPE-220 Homework

Sravan Kumar Gorati **SJSU ID:017441749**

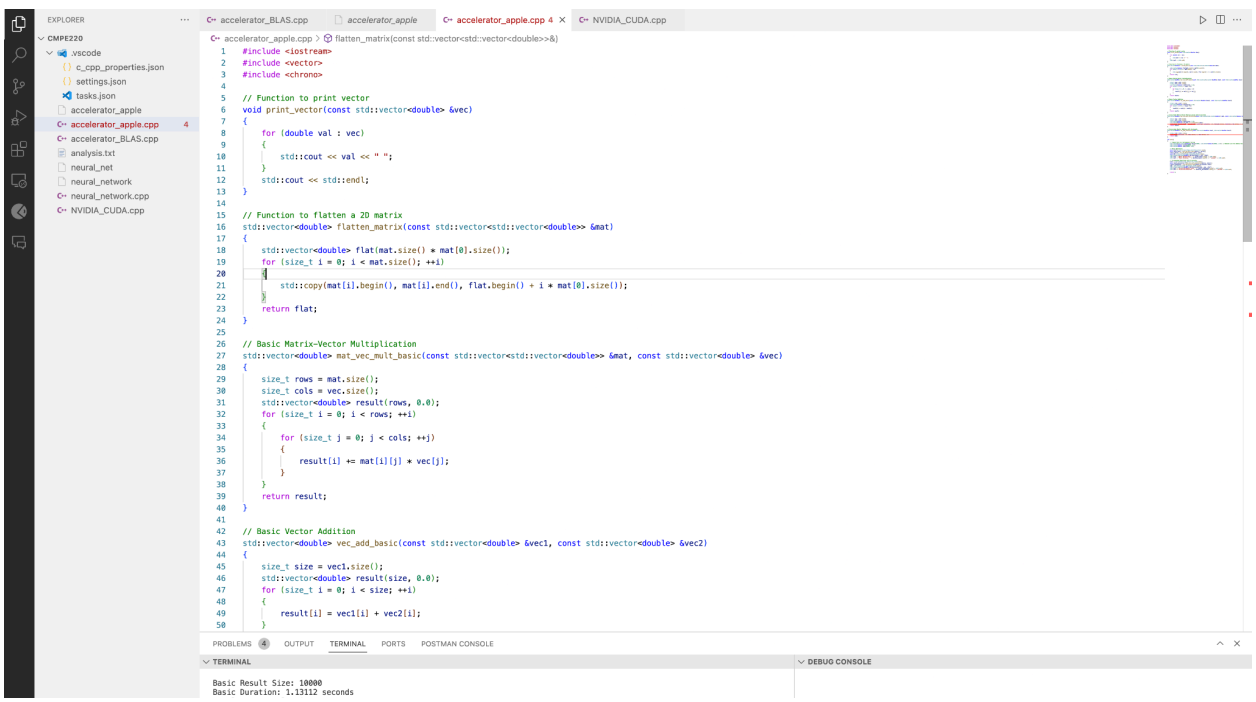
Colab Link: CMPE220-Sravan.ipynb
Github Link: https://github.com/sravangorati2001/CMPE220-Homework

Performance Analysis of

Neural Network Operations in C/C++ and Python

Step1:

Basic Neural Network operation in C/C++:



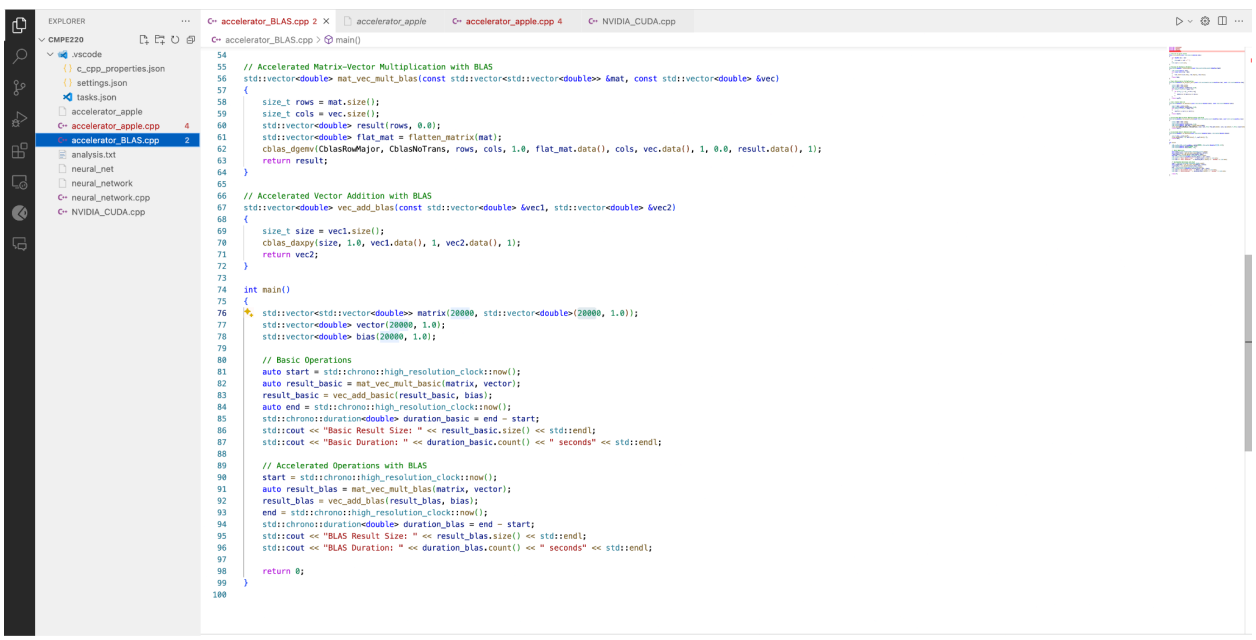
Sample Output for size: 10000

```
Basic Result Size: 10000
Basic Duration: 1.13112 seconds
```

Output for Different Inputs:

Input	Basic
10000	1.0021
20000	1.70555
30000	5.73627
40000	9.89812
45000	12.3204

Acceleration using BLAS (Basic Linear Algebra Subprograms):

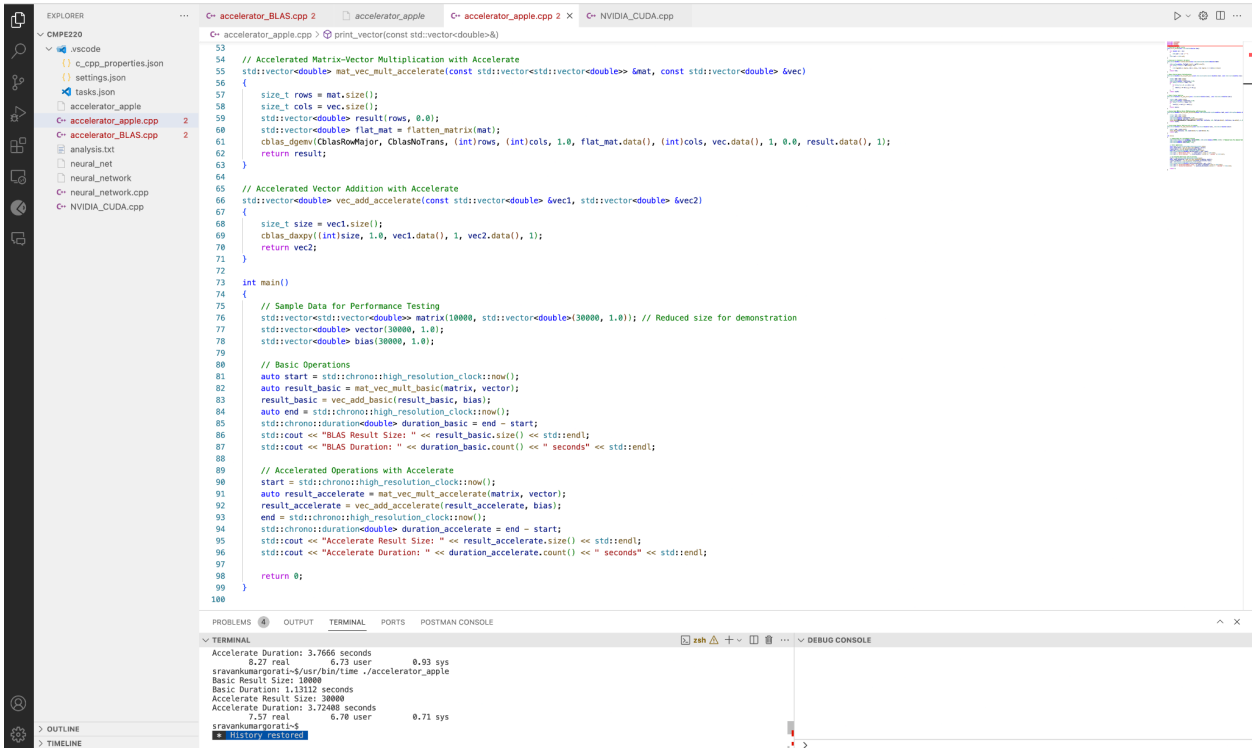



```
// Accelerated Matrix-Vector Multiplication with BLAS
std::vector<double> mat_vec_mult_blas(const std::vector<std::vector<double>> &mat, const std::vector<double> &vec)
{
    size_t rows = mat.size();
    size_t cols = vec.size();
    std::vector<double> result(rows, 0.0);
    std::vector<double> flat_mat = flatten_matrix(mat);
    cblas_dgemv(CblasRowMajor, CblasNoTrans, rows, cols, 1.0, flat_mat.data(), cols, vec.data(), 1, 0.0, result.data(), 1);
    return result;
}

// Accelerated Vector Addition with BLAS
std::vector<double> vec_add_blas(const std::vector<double> &vec1, std::vector<double> &vec2)
{
    size_t size = vec1.size();
    cblas_daxpy(size, 1.0, vec1.data(), 1, vec2.data(), 1);
    return vec2;
}

int main()
{
    std::vector<std::vector<double>> matrix(20000, std::vector<double>(20000, 1.0));
    std::vector<double> vector(20000, 1.0);
    std::vector<double> bias(20000, 1.0);

    // Basic Operations
    auto start = std::chrono::high_resolution_clock::now();
    auto result_basic = mat_vec_mult_basic(matrix, vector);
    result_basic = vec_add_basic(result_basic, bias);
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> duration_basic = end - start;
    std::cout << "Basic Result Size: " << result_basic.size() << std::endl;
    std::cout << "Basic Duration: " << duration_basic.count() << " seconds" << std::endl;

    // Accelerated Operations with BLAS
    start = std::chrono::high_resolution_clock::now();
    auto result_blas = mat_vec_mult_blas(matrix, vector);
    result_blas = vec_add_blas(result_blas, bias);
    end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> duration_blas = end - start;
    std::cout << "BLAS Result Size: " << result_blas.size() << std::endl;
    std::cout << "BLAS Duration: " << duration_blas.count() << " seconds" << std::endl;

    return 0;
}
```

Output for Different Inputs:

Input	BLAS
10000	2.9876
20000	4.7325
30000	14.387
40000	24.0987
45000	33.1647

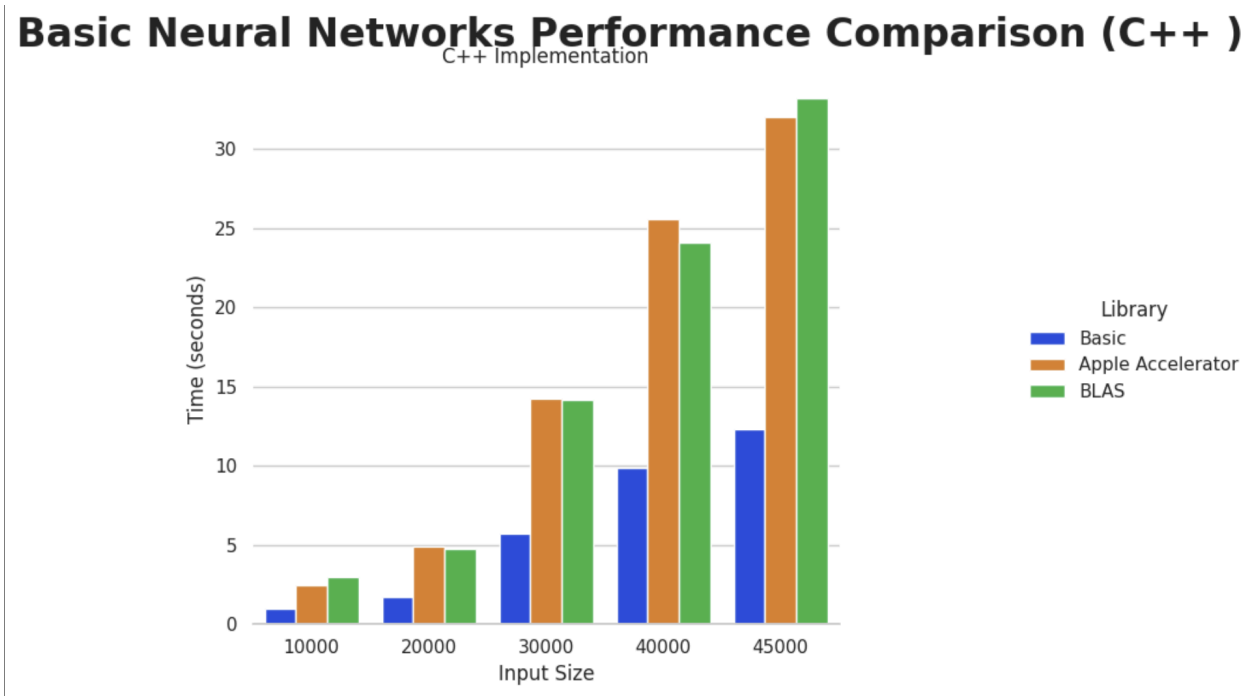
Acceleration using Apple's accelerate:



Output for Different Inputs:

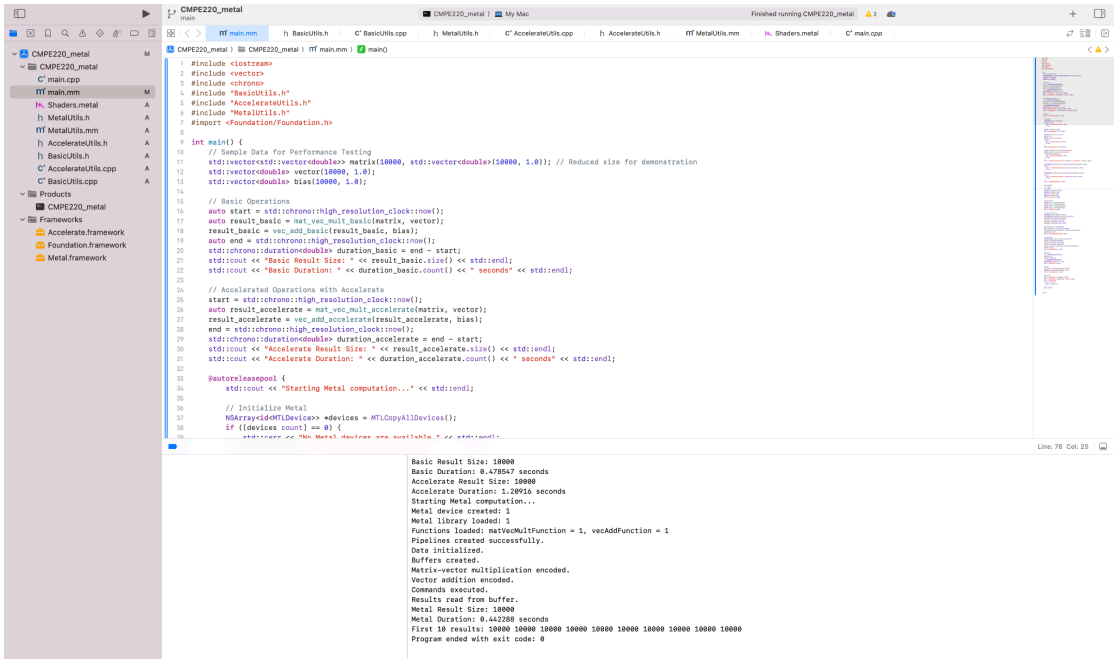
Input	Apple Accelerator
10000	2.4378
20000	4.92506
30000	14.2296
40000	25.5829
45000	32.0233

Performance Analysis of Basic, BLAS, Apple's Accelerator:



Step2:

Neural network operation using Apple's Metal Framework:



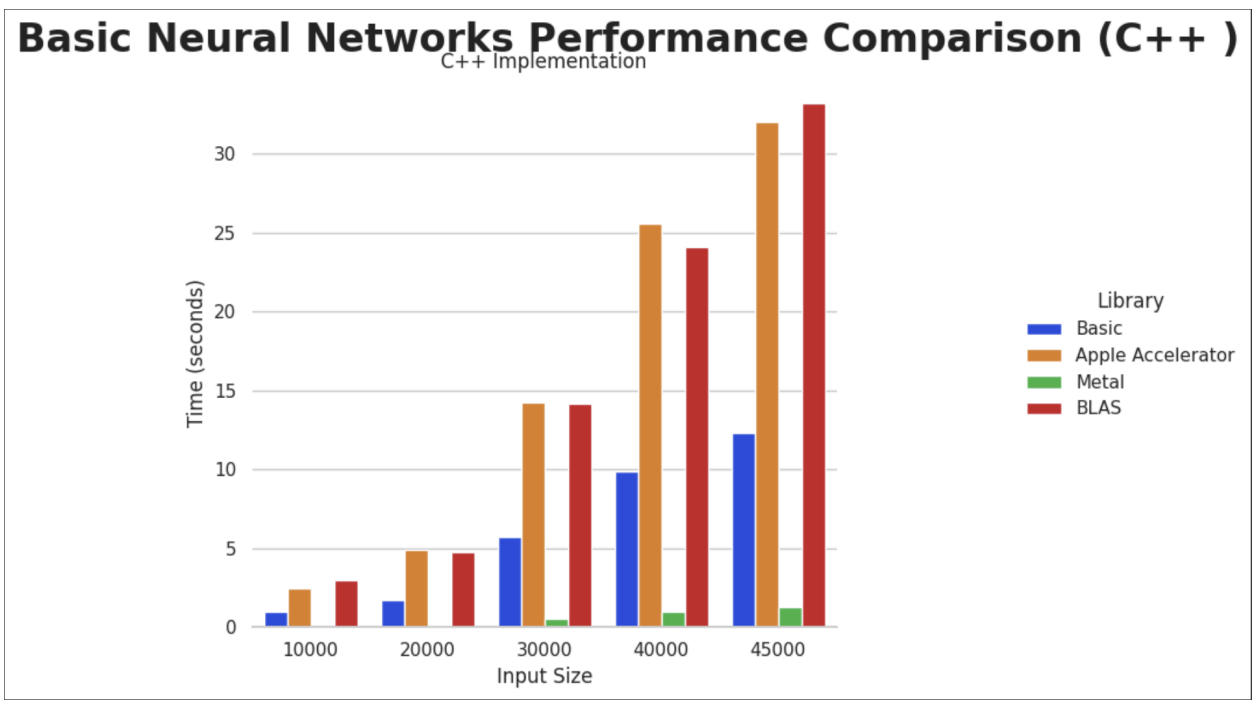
Sample output:

```
Basic Result Size: 10000
Basic Duration: 0.478547 seconds
Accelerate Result Size: 10000
Accelerate Duration: 1.20916 seconds
Starting Metal computation...
Metal device created: 1
Metal library loaded: 1
Functions loaded: matVecMultFunction = 1, vecAddFunction = 1
Pipelines created successfully.
Data initialized.
Buffers created.
Matrix-vector multiplication encoded.
Vector addition encoded.
Commands executed.
Results read from buffer.
Metal Result Size: 10000
Metal Duration: 0.442288 seconds
```

Output for different inputs:

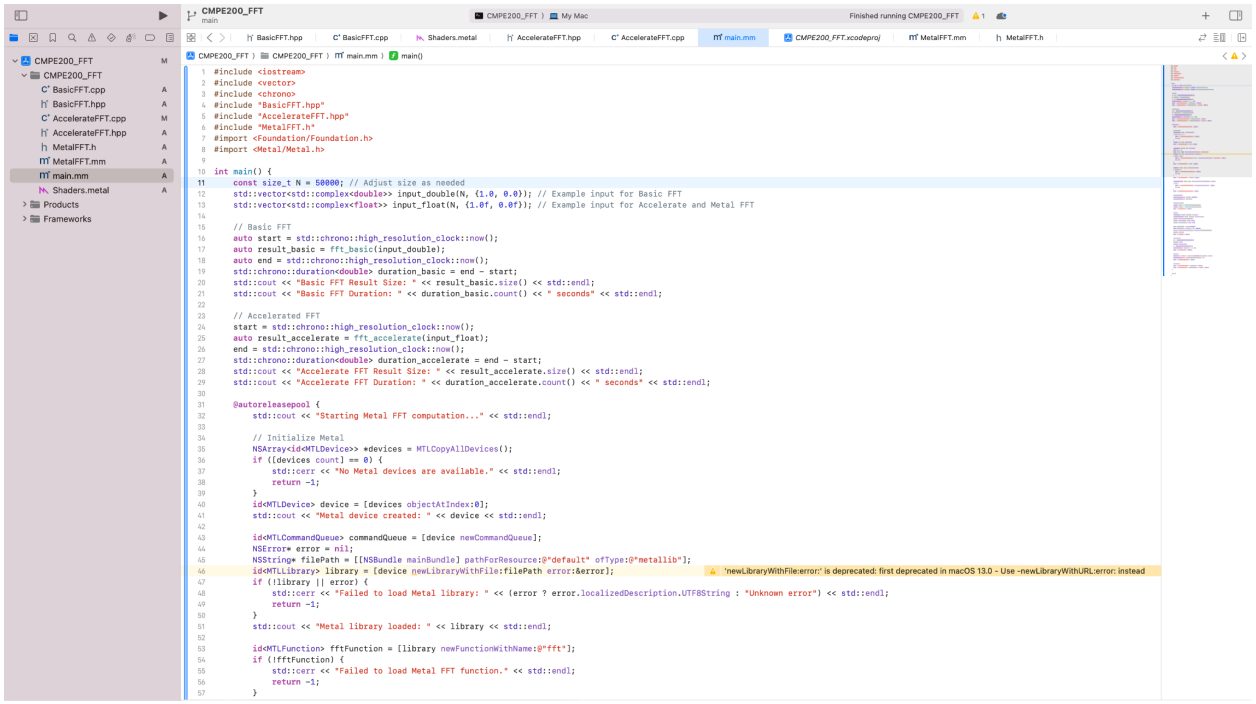
Input	Metal
10000	0.01965
20000	0.108957
30000	0.549007
40000	0.973443
45000	1.26005

Performance Analysis of Basic, BLAS, Apple's Accelerator, Metal:



Step3:

FFT (Fast Fourier Transform) Performance in C++:



```
#include <iostream>
#include <vector>
#include <chrono>
#include "BasicFFT.hpp"
#include "AccelerateFFT.hpp"
#include "MetalFFT.h"
#import <Foundation/Foundation.h>
#import <Metal/Metal.h>

int main() {
    const size_t N = 50000; // Adjust size as needed
    std::vector<std::complex<double>> input_double(N, {1.0, 0.0}); // Example input for Basic FFT
    std::vector<std::complex<float>> input_float(N, {1.0f, 0.0f}); // Example input for Accelerate and Metal FFT

    // Basic FFT
    auto start = std::chrono::high_resolution_clock::now();
    auto result_basic = fft_basic(input_double);
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> duration_basic = end - start;
    std::cout << "Basic FFT Result Size: " << result_basic.size() << std::endl;
    std::cout << "Basic FFT Duration: " << duration_basic.count() << " seconds" << std::endl;

    // Accelerated FFT
    start = std::chrono::high_resolution_clock::now();
    auto result_accelerate = fft_accelerate(input_float);
    end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> duration_accelerate = end - start;
    std::cout << "Accelerate FFT Result Size: " << result_accelerate.size() << std::endl;
    std::cout << "Accelerate FFT Duration: " << duration_accelerate.count() << " seconds" << std::endl;

    @autoreleasepool {
        std::cout << "Starting Metal FFT computation..." << std::endl;

        // Initialize Metal
        NSArray<id<MTLDevice>> *devices = MTLCopyAllDevices();
        if ([devices count] == 0) {
            std::cerr << "No Metal devices are available." << std::endl;
            return -1;
        }
        id<MTLDevice> device = [devices objectAtIndex:0];
        std::cout << "Metal device created: " << device << std::endl;

        id<MTLCommandQueue> commandQueue = [device newCommandQueue];
        NSError* error = nil;
        NSString* filePath = [[NSBundle mainBundle] pathForResource:@"default" ofType:@"metallib"];
        id<MTLLibrary> library = [device newLibraryWithFile:filePath error:&error];
        if (!library || error) {
            std::cerr << "Failed to load Metal library: " << (error ? error.localizedDescription.UTF8String : "Unknown error") << std::endl;
            return -1;
        }
        std::cout << "Metal library loaded: " << library << std::endl;

        id<MTLFunction> fftFunction = [library newFunctionWithName:@"fft"];
        if (!fftFunction) {
            std::cerr << "Failed to load Metal FFT function." << std::endl;
            return -1;
        }
```

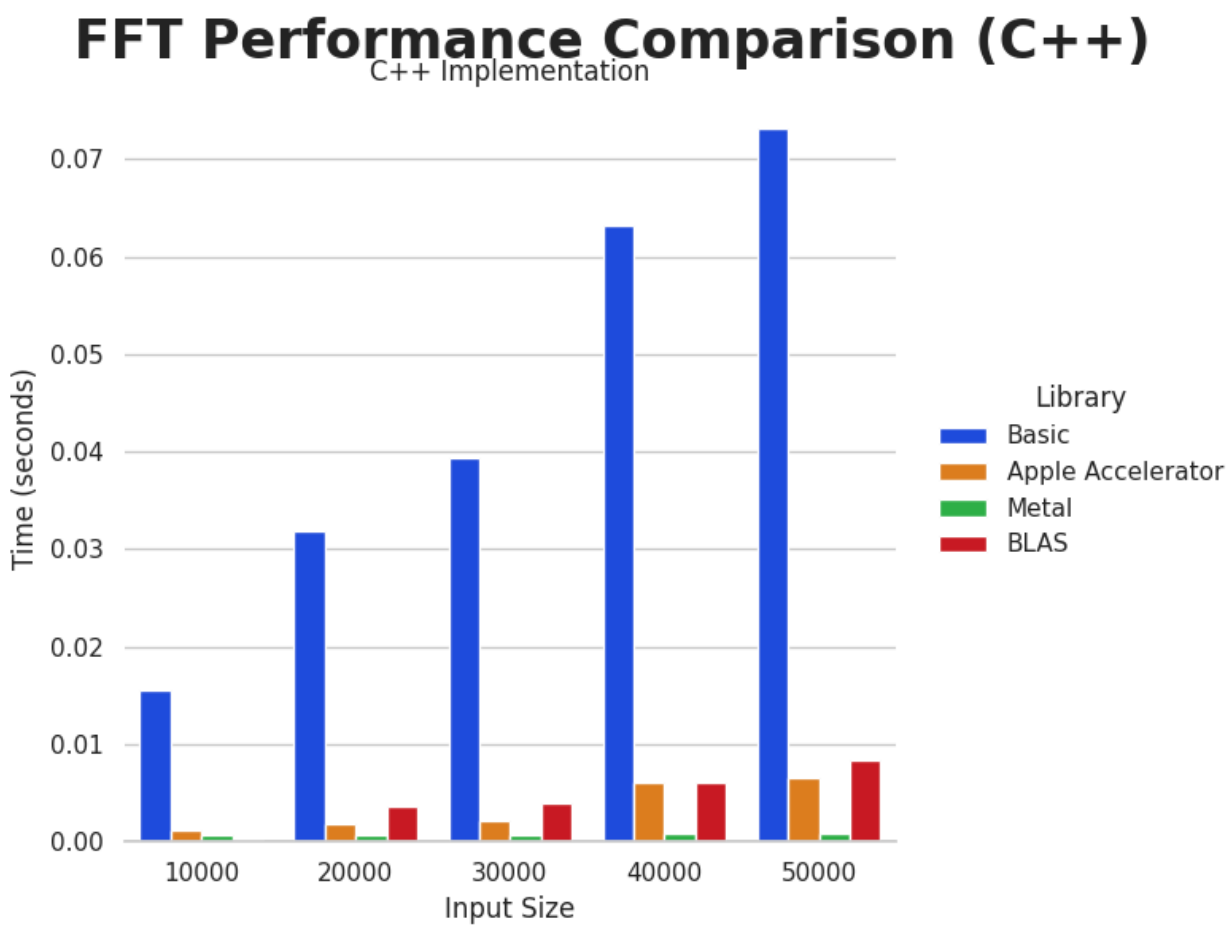
Sample output for input 10000:

```
Basic FFT Result Size: 10000
Basic FFT Duration: 0.0160458 seconds
Accelerate FFT Result Size: 10000
Accelerate FFT Duration: 0.00114825 seconds
Starting Metal FFT computation...
Metal device created: 1
Metal library loaded: 1
FFT Function loaded: 1
FFT Pipeline created successfully.
Buffers created.
FFT encoded.
Commands executed.
Results read from buffer.
Metal FFT Result Size: 10000
Metal FFT Duration: 0.000606125 seconds
```

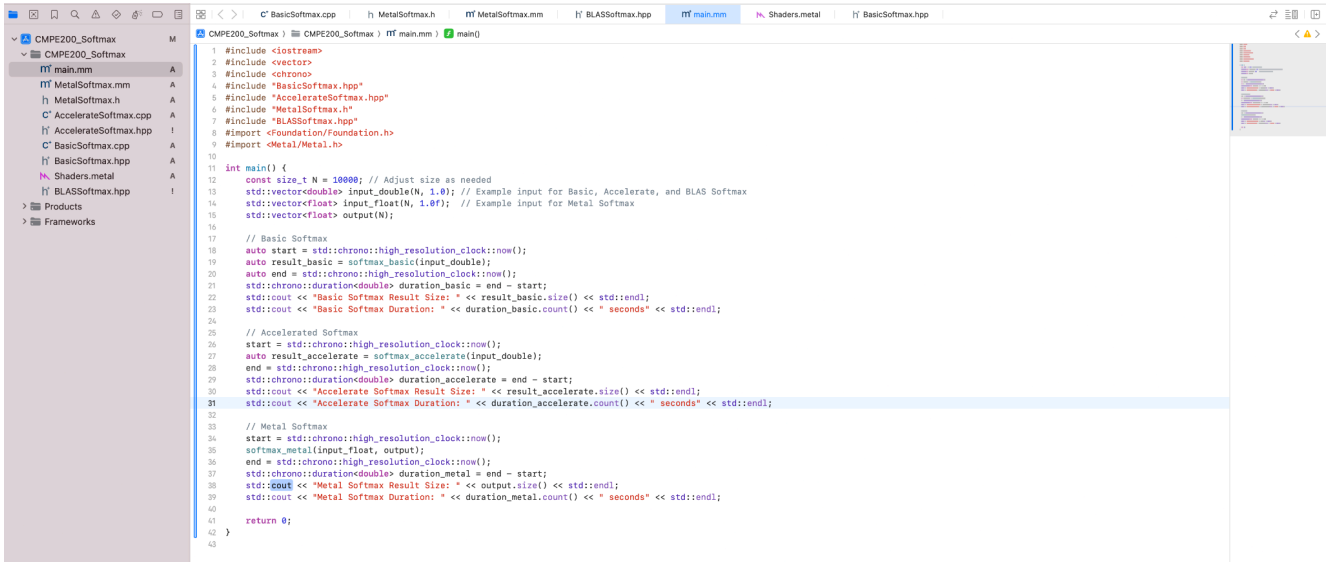
Output for Different Inputs:

FFT				
Input	**Metal**	**Apple Accelerator**	**BLAS**	**Basic**
10000	0.000716042	0.00112067	0.0015234	0.0156052
20000	0.0007319929	0.00191117	0.0359372	0.0319372
30000	0.000757792	0.00219592	0.03956	0.039306
40000	0.000790084	0.006104	0.06145	0.063168
50000	0.000813459	0.00659729	0.08354	0.0730943

Performance Analysis of FFT in C++:



Softmax Implementation in C++:



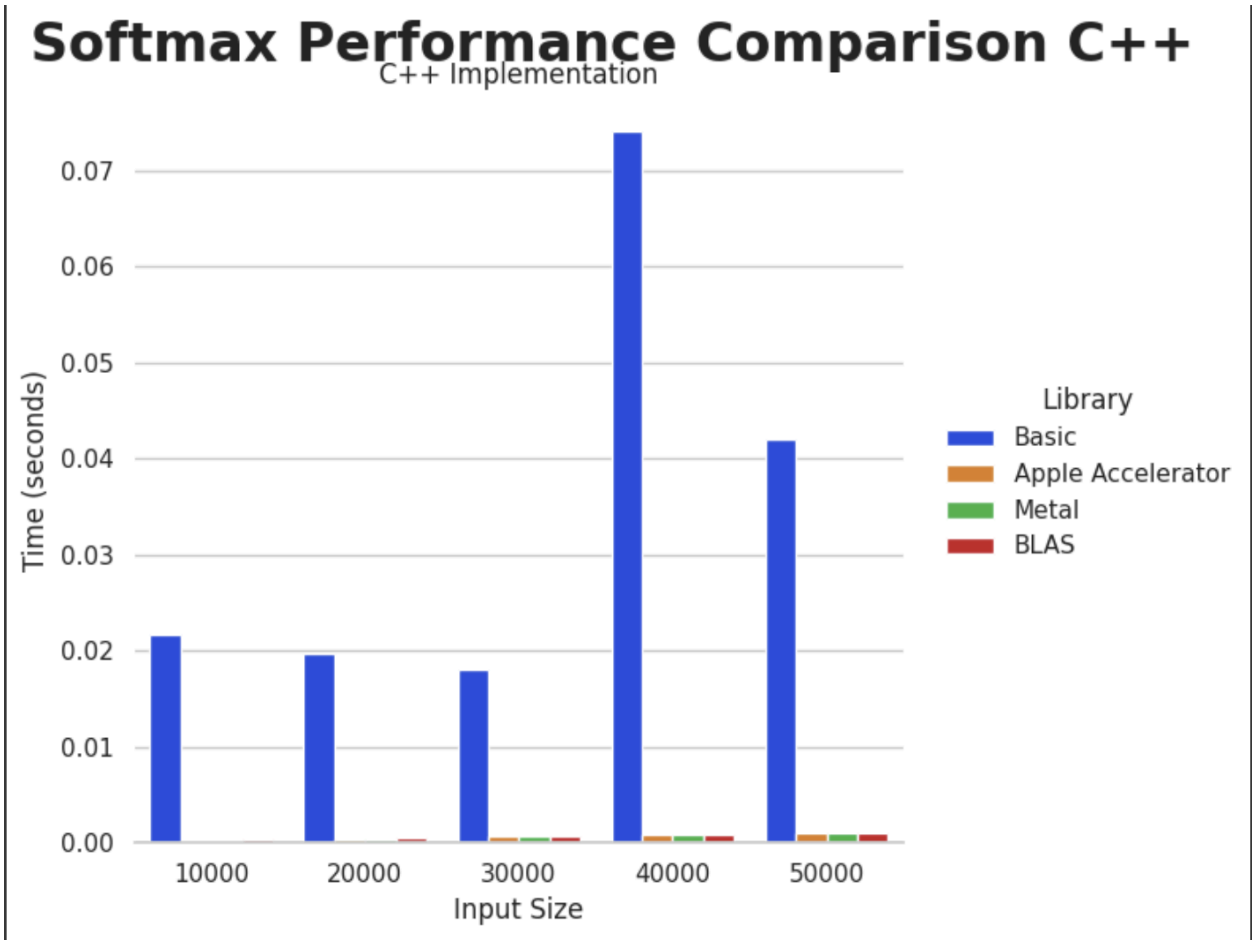
Sample output for input 10000:

```
Basic Softmax Result Size: 10000
Basic Softmax Duration: 0.0002065 seconds
Accelerate Softmax Result Size: 10000
Accelerate Softmax Duration: 0.000217084 seconds
Metal Softmax Result Size: 10000
Metal Softmax Duration: 0.0182341 seconds
Program ended with exit code: 0
```

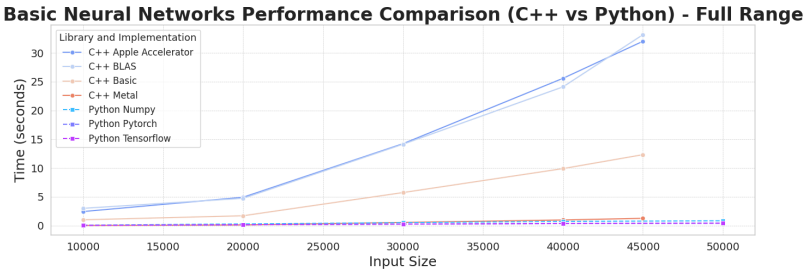
Output for Different Inputs:

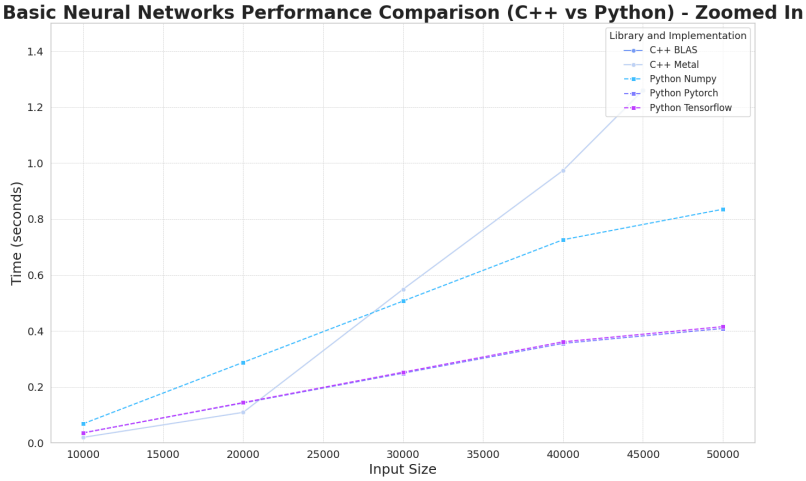
Softmax				
Input	**Metal**	**Apple Accelerator**	**BLAS**	**Basic**
10000	0.000201667	0.000254	0.000314	0.0217576
2000	0.000414084	0.000392458	0.000352458	0.019743
3000	0.000624542	0.000621583	0.00071583	0.0180205
4000	0.000825709	0.000878542	0.000867542	0.074053
5000	0.00103075	0.00109221	0.00199521	0.0421108

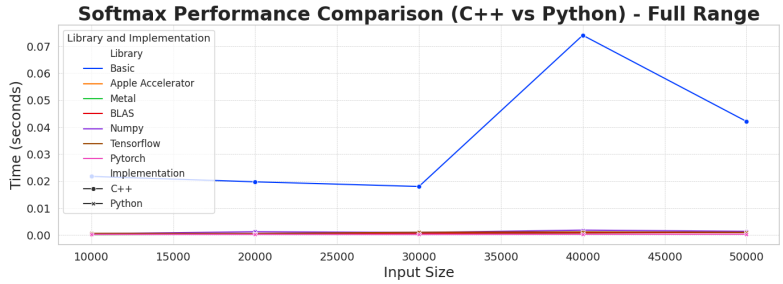
Performance Analysis of Softmax in C++:

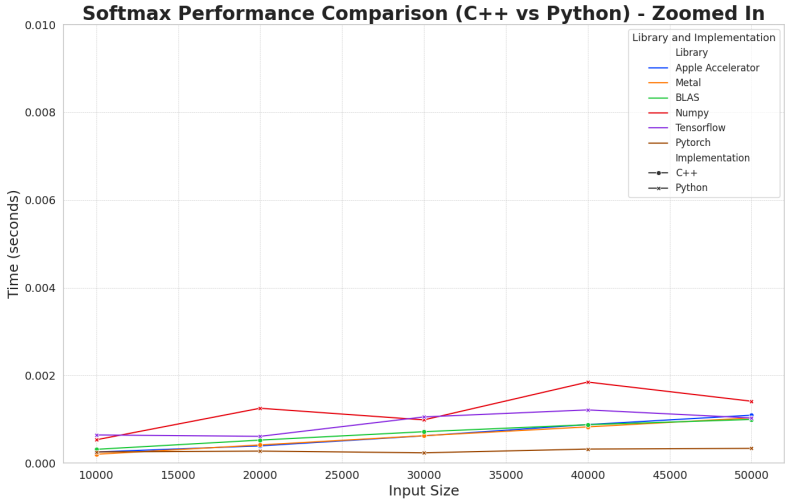


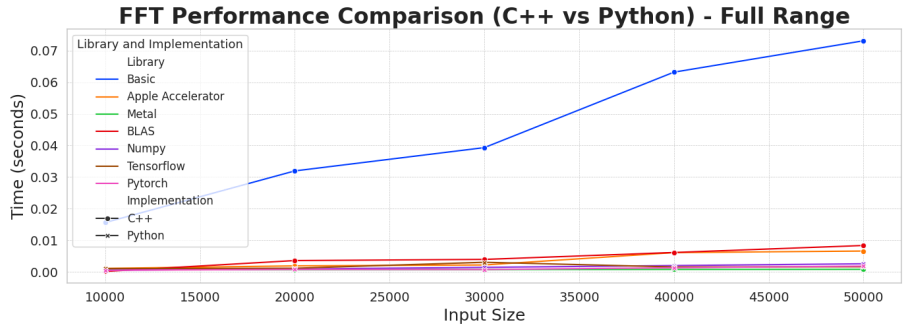
C++ vs Python Performance Analysis

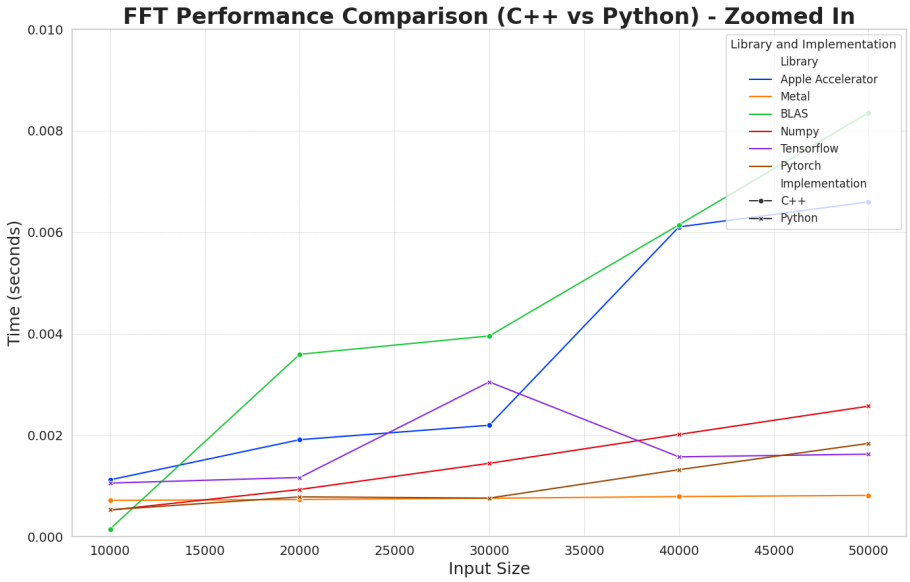









FFT Performance Comparison (C++ vs Python) - Full Range
Library and Implementation
Library
Basic
Apple Accelerator
Metal
BLAS
Numpy
Tensorflow
Pytorch
Implementation
C++
Python
Time (seconds)
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
10000
15000
20000
25000
30000
35000
40000
45000
50000
Input Size


FFT Performance Comparison (C++ vs Python) - Zoomed In
Library and Implementation
Library
Apple Accelerator
Metal
BLAS
Numpy
Tensorflow
Pytorch
Implementation
C++
Python
Time (seconds)
0.000
0.002
0.004
0.006
0.008
0.010
10000
15000
20000
25000
30000
35000
40000
45000
50000
Input Size