Assignment 11 - Exercise 1 Report

1. Introduction

The aim of this exercise is to evaluate the effect of auto-vectorization on a simple multiply-add operation applied to three float vectors. Performance is measured in terms of execution time and floating-point instruction counts using `perf` on the LCC3 cluster.

2. Program Description

The program performs the operation `a[i] += b[i] * c[i]` in a loop repeated 1e6 times, for vectors of varying sizes. All vectors are initialized with constant values. The result is verified by checking the final sum of vector `a`. Compilation was done using `gcc 12.2.0`.

3. Compilation and Execution

Baseline compilation:

```
gcc -O1 -o vec_baseline vec_baseline.c
```

Auto-vectorized compilation:

```
gcc -O1 -ftree-vectorize -o auto_vectorized vec_baseline.c
```

4. Results

Vector size: 2048

Repetitions: 1e6

Baseline:

- Elapsed time: 2.68 s
- sum(a): 12287900672.0
- perf (SSE Single Precision): 4,096,172,587 (r4010:u)

Assignment 11 - Exercise 1 Report

Auto-vectorized:

Elapsed time: 0.51 s

sum(a): 12287900672.0

perf (SSE Single Precision): 1,024,013,419 (r4010:u)

5. Analysis and Observations

Auto-vectorization achieved a speedup of approximately 5.2x over the baseline. The result remained numerically stable, confirming correctness. `perf` analysis shows significantly fewer SSE single-precision instructions executed, indicating efficient use of SIMD (packed operations).

Vector size influences performance: small sizes incur overhead, while larger sizes better exploit vectorization. The flag `-ftree-vectorize` enables vectorization without introducing other optimizations. Additional flags like `-fopt-info-vec` or `-march=native` may help fine-tune future experiments.

6. Conclusion

Compiler auto-vectorization using `-ftree-vectorize` leads to substantial performance gains for data-parallel operations. This experiment validates both the correctness and efficiency of such optimizations. Future work can explore manual SIMD usage or advanced vector instruction sets like AVX.