CS683: Advanced Computer Architecture
Programming Assignment 1
Task 2: Embed It

Dhruv Meena, Saptarshi Biswas, Madhava Sriram

September 1, 2025

1 Part A: Software Prefetching

1.1 Code Implementation

Software prefetching did not work as expected. We did not see any improvement in the number of misses and our maximum speedup was 2.0x

Our devices could not support disabling hardware prefetching so we could not perform the software prefetching versus hardware prefetching analysis.

1.2 Results

If not mentioned, assume that blue bar is `__mm_prefetch()`, red is `_builtin_prefetch()` and yellow is naive implementation.

1.2.1 Varying Embedding Table Size

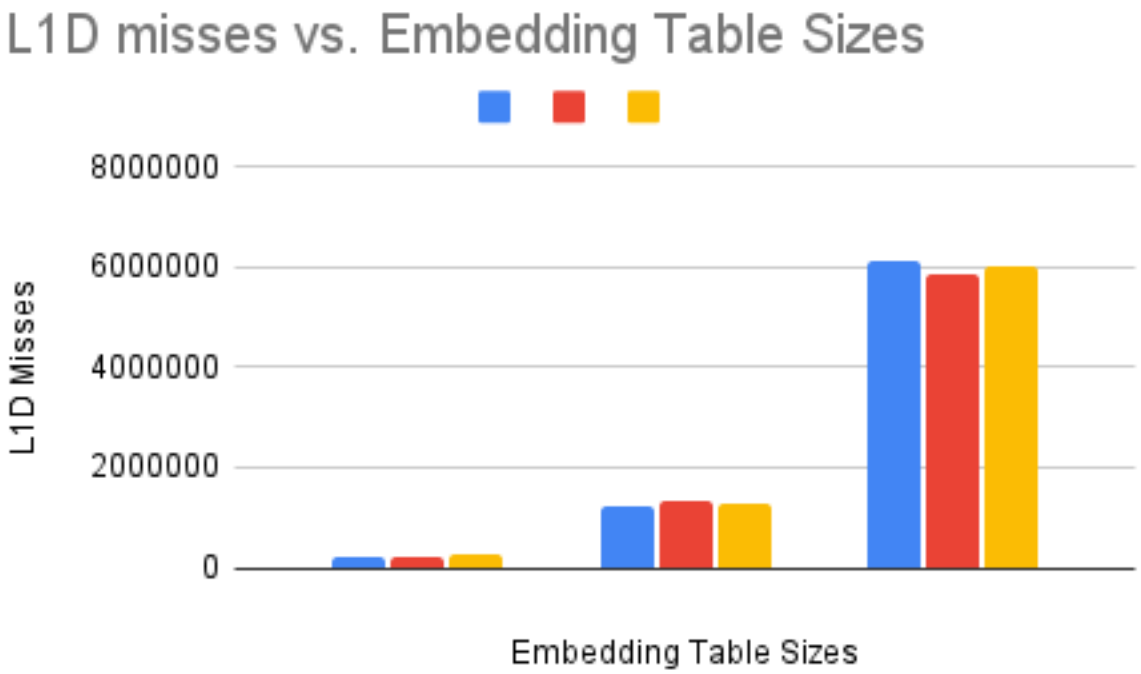

Figure 1: L1D misses vs. Embedding Table Sizes

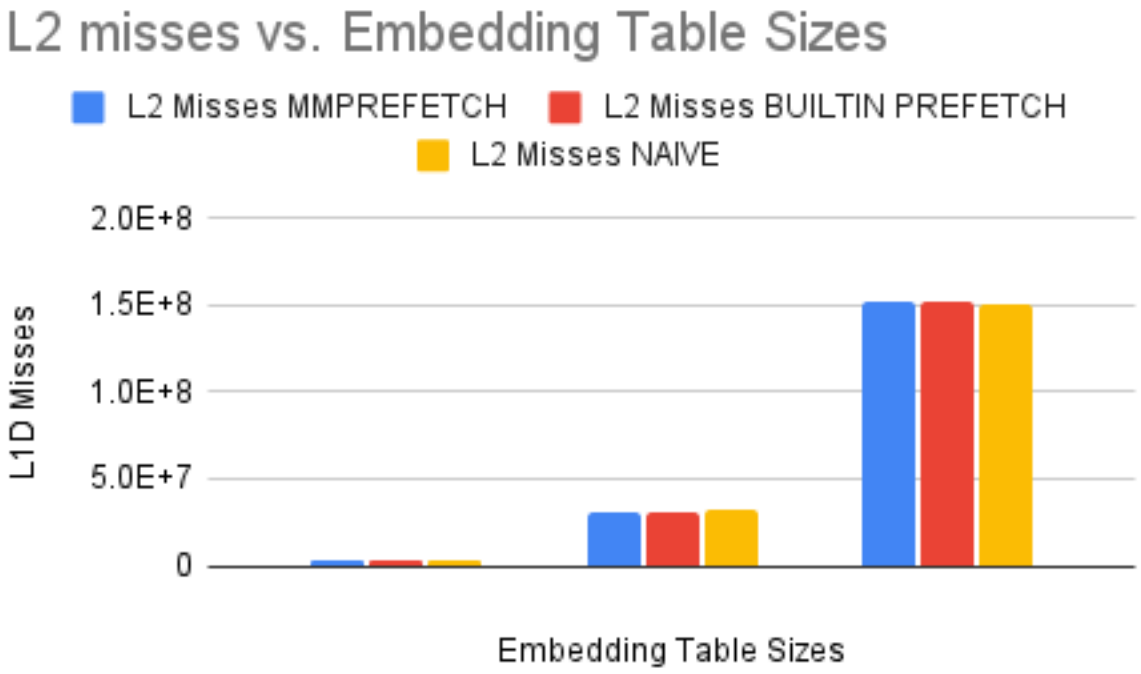

Figure 2: L2 misses vs. Embedding Table Sizes

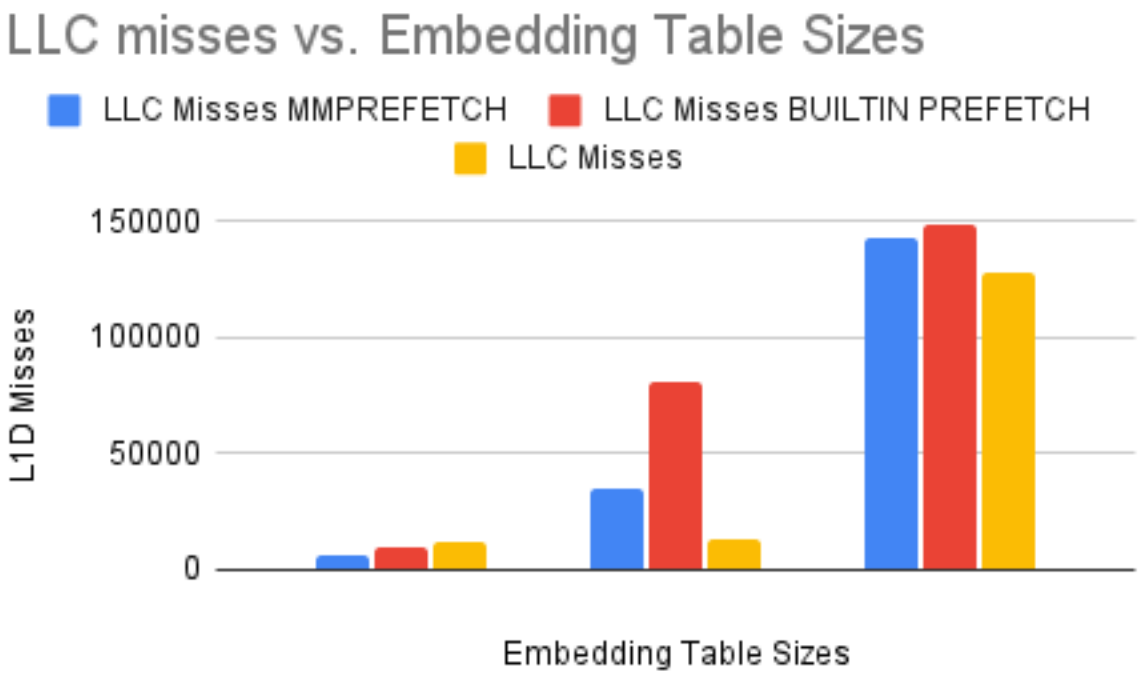

Figure 3: LLC misses vs. Embedding Table Sizes

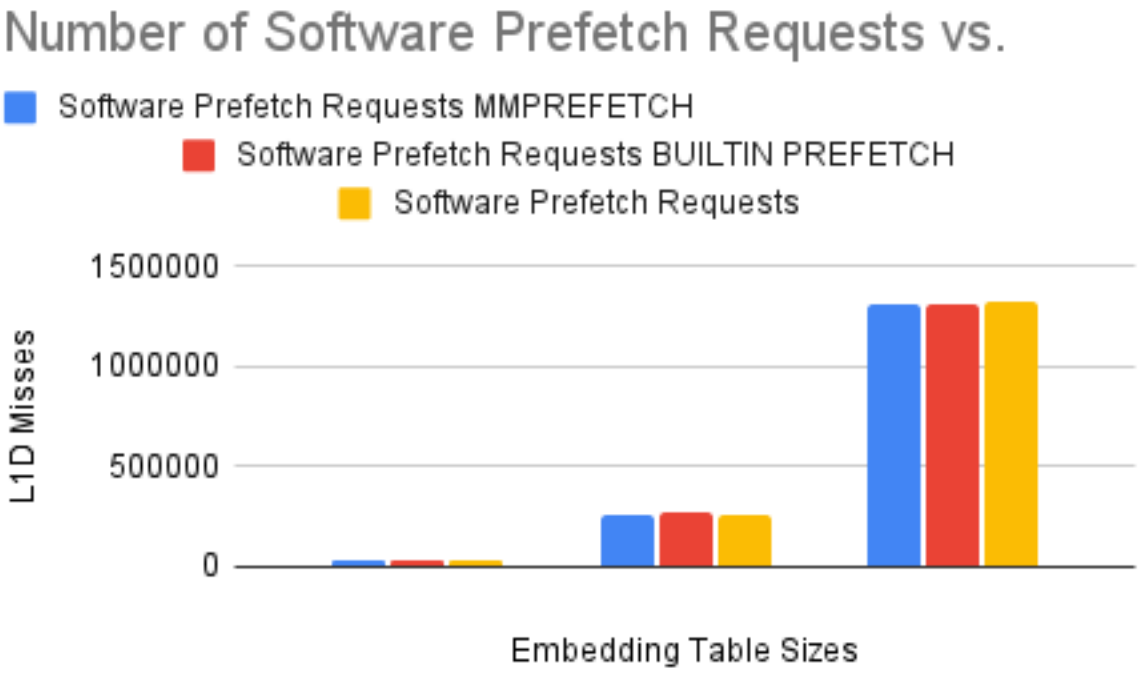

Figure 4: Number of Software Prefetch Requests vs. Embedding Table Sizes

1.2.2 Speedup

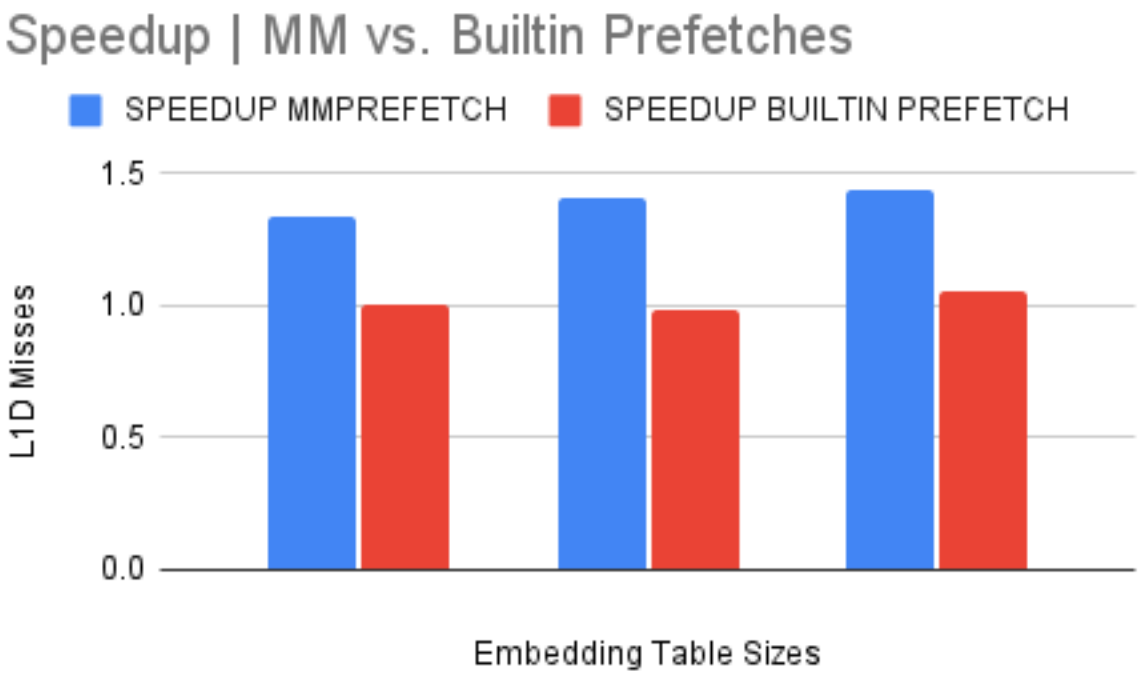

Figure 5: MM vs. Builtin Prefetches

1.3 Varying Prefetch Distance

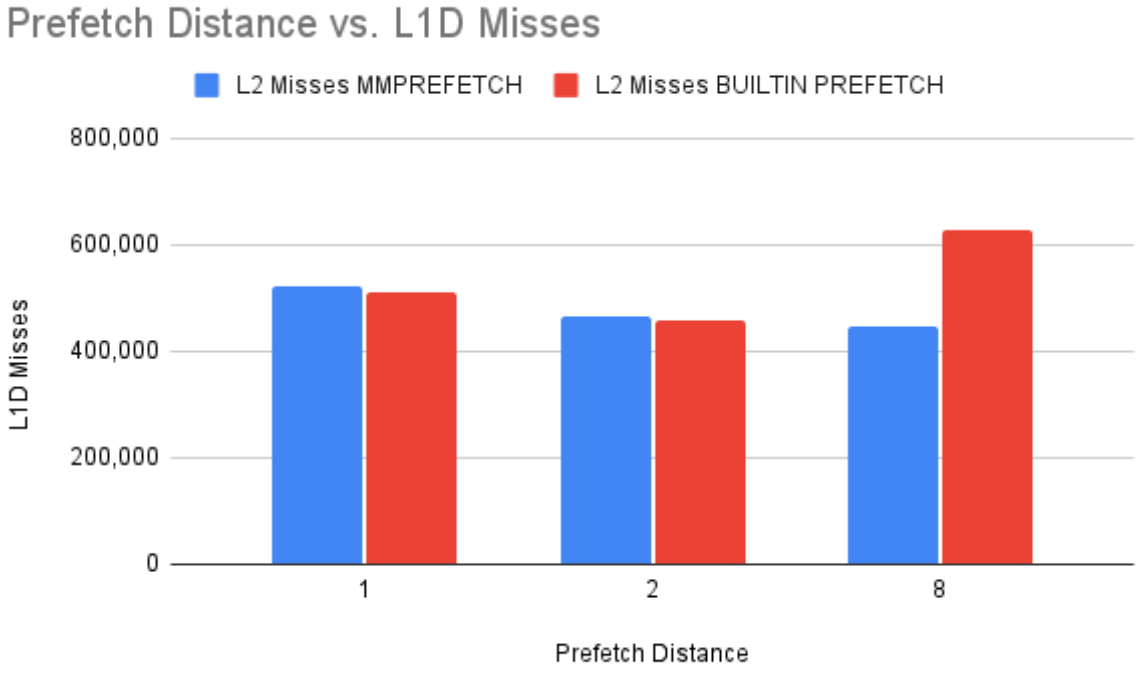

Figure 6: Prefetch Distance vs. L1D Misses

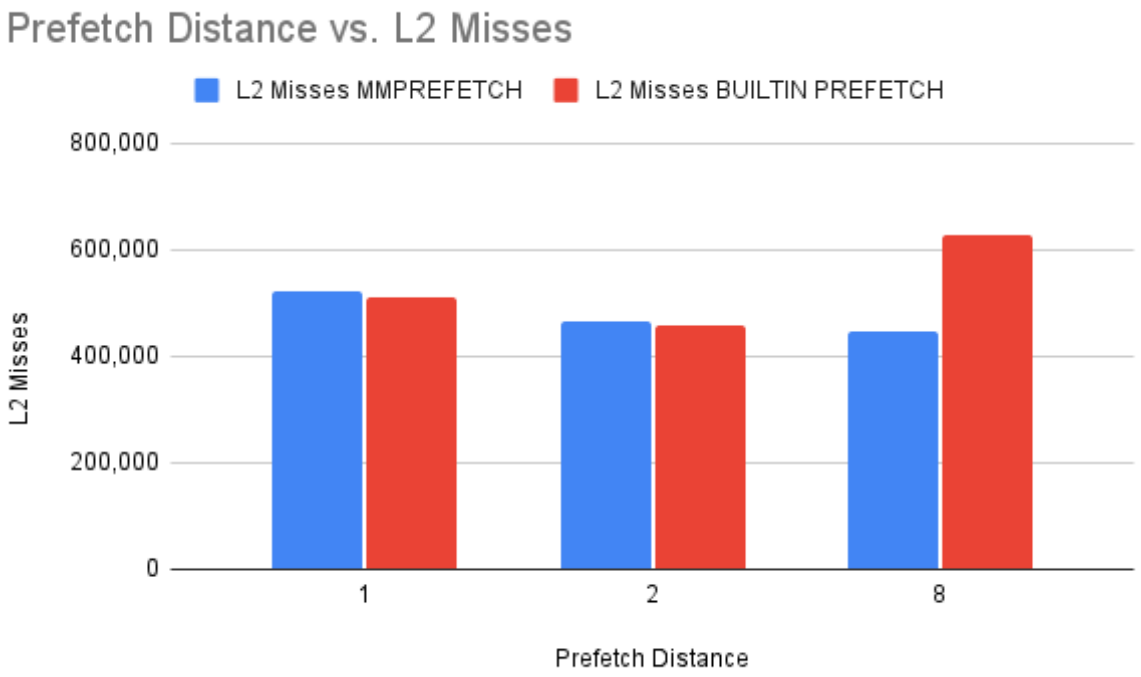

Figure 7: Prefetch Distance vs. L2 Misses

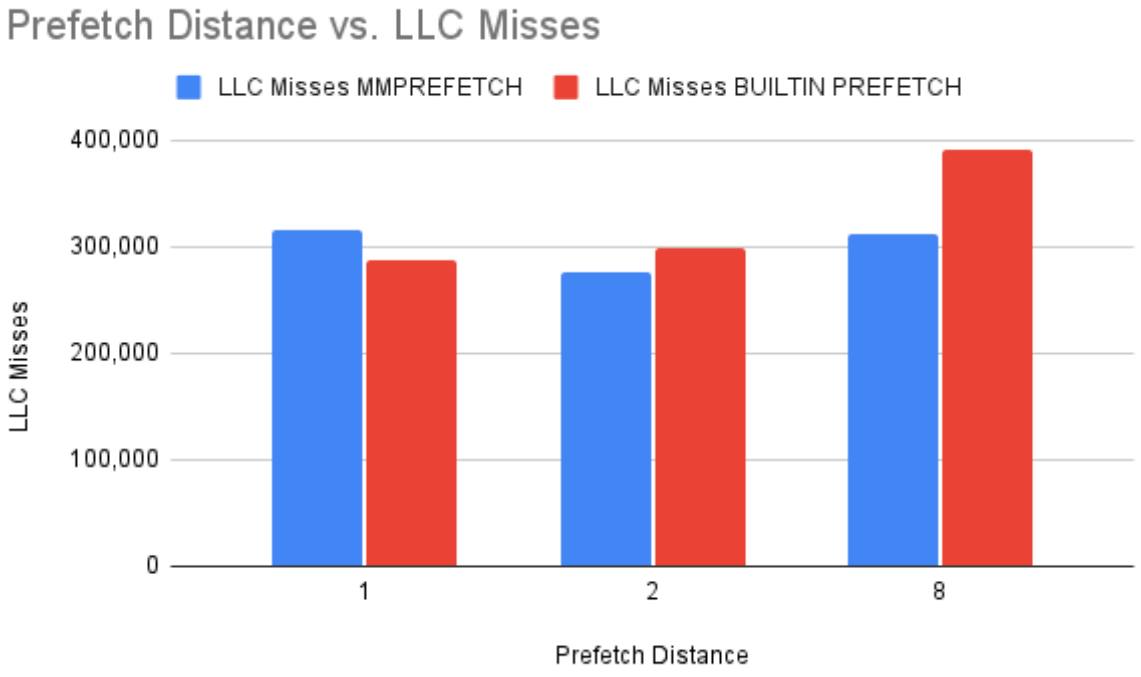

Figure 8: Prefetch Distance vs. LLC Misses

1.4 Discussion

- **What trend do you observe in speedup with different embedding table sizes?**

We observed that there was no significant difference between the speedups across different embedding table sizes for both `__mm_prefetch()` and `_builtin_prefetch()`

This was probably because of the following reasons:

- We did not perform prefetching properly.
- Hardware prefetching was able to hide all of the latency so software prefetching did not provide any improvement.

- **What is the best prefetch distance?**

  Prefetch distance of 2 turned out to be the best for us although the improvement is not significant enough to call it optimal in true sense.

  Overall MM Prefetch worked slightly better than built-in prefetch.

- **At what cache fill level do you achieve the maximum speedup?**

  We achieve maximum speedup for cache fill level 0. The data must come into L1 cache as quick as possible. So HINT T0 for MM Prefetch and Locality 0 for Builting Prefetch worked best.

2 Part B: SIMD

2.1 Code Implementation

We used the AVX SIMD registers for single precision floats rather than double precision floats as mentioned in the assignment document. The reason is that `emb.cpp` declares everything as floats rather than doubles. As a consequence of this we use the instructions `_mm_loadu_ps` and `_mm_add_ps` instead of `_mm256_loadu_pd` or `_mm256_loadu_pd`.

2.1.1 Flags Used

- -mavx2: enables 256bit SIMD registers
- -msse: enables 128bit SIMD registers
- -mavx512f: enables 512bit SIMD registers
- -O0: use no compiler optimization
- -Wall: see all warnings
- -g: easy debugging

The above flags were removed when running the naive implementation as they can cause the compiler to perform auto-vectorization of loops resulting in unneeded optimization.

2.1.2 Perf Stats Used

- `fp_arith_inst_retired.scalar_single`: Counts the number of scalar single-precision float arithmetic operations.
- `fp_arith_inst_retired.128b_packed_single`: Counts 128 bit packed single precision float arithmetic operations.
- `fp_arith_inst_retired.256b_packed_single`: Counts 256 bit packed single precision float arithmetic operations.
- `fp_arith_inst_retired.512b_packed_single`: Counts 512 bit packed single precision float arithmetic operations.

Simply using `instructions` is not useful as it also counts system calls, overhead utility stuff, etc., There were over 44 billion instructions and it was impossible to make out the effect of SIMD.

2.2 Results

2.2.1 Varying Embedding Dimensions

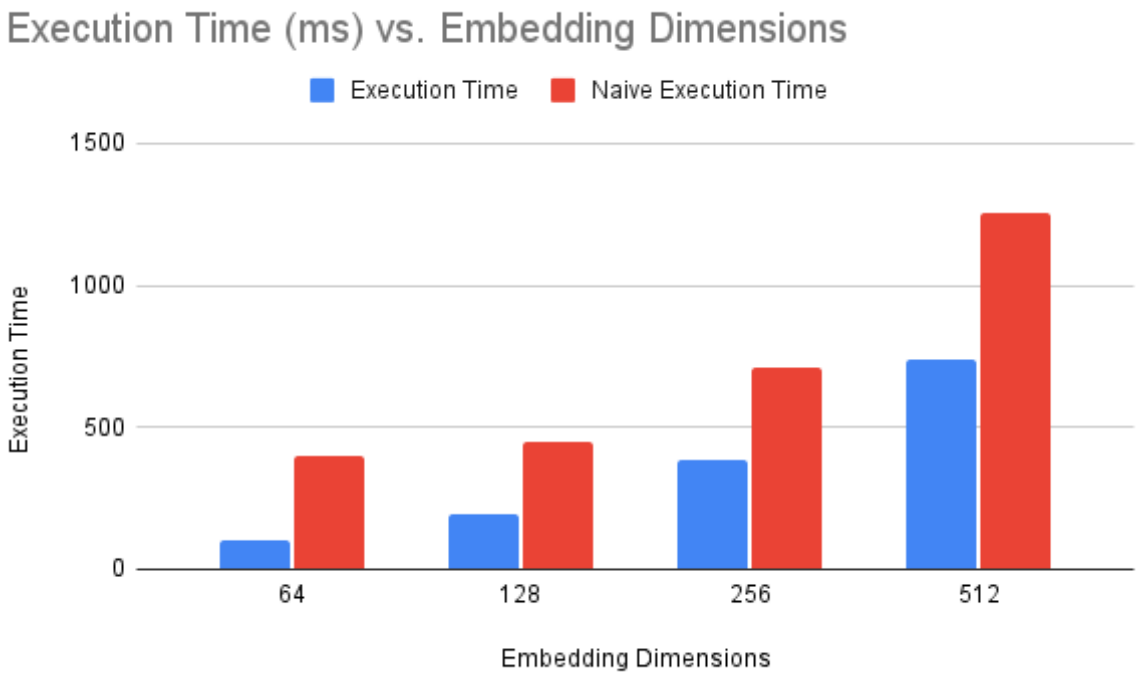

Figure 9: Execution Time vs. Embedding Dimensions

H

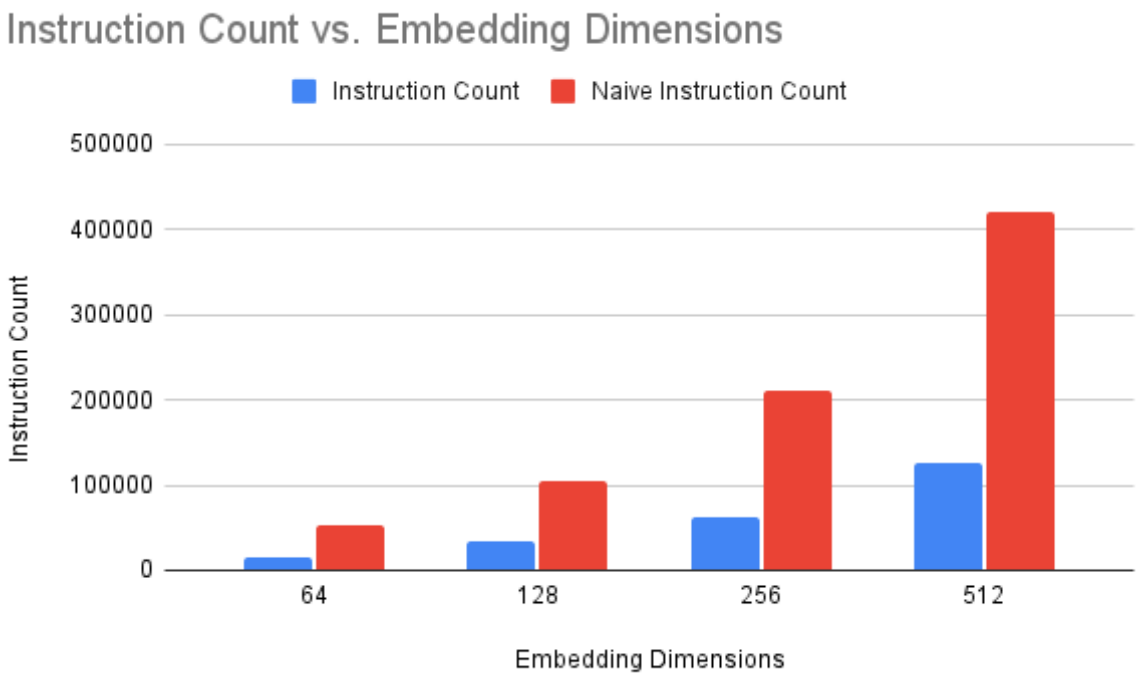

Figure 10: Instruction Count vs. Embedding

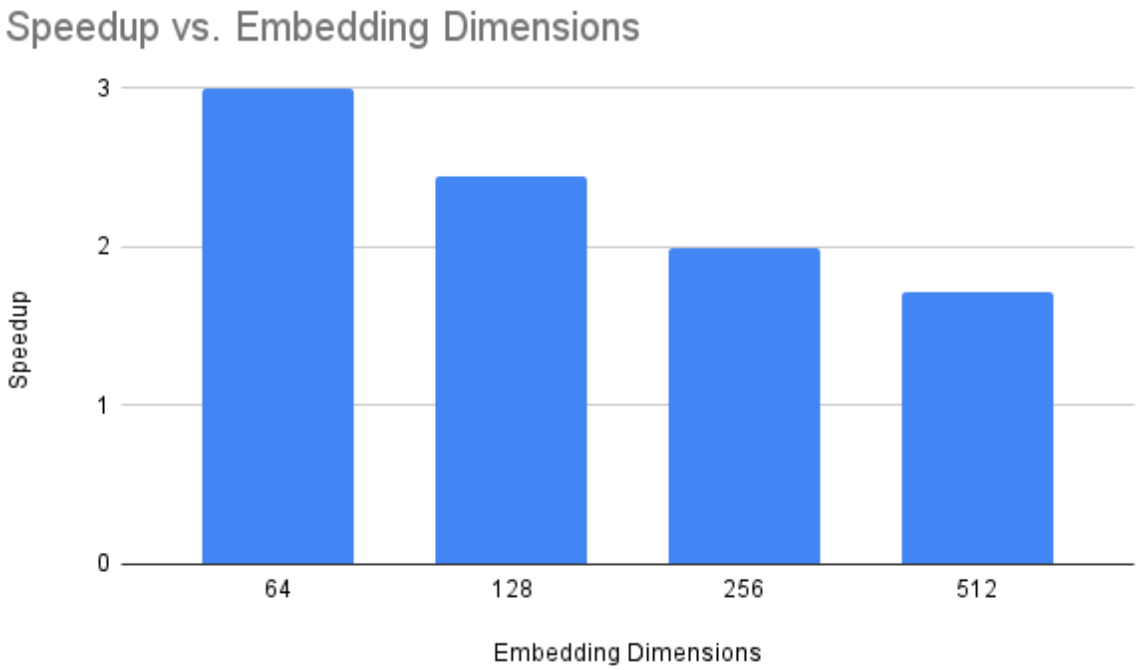

Figure 11: Speedup vs. Embedding Dimensions

2.2.2 Varying SIMD Widths

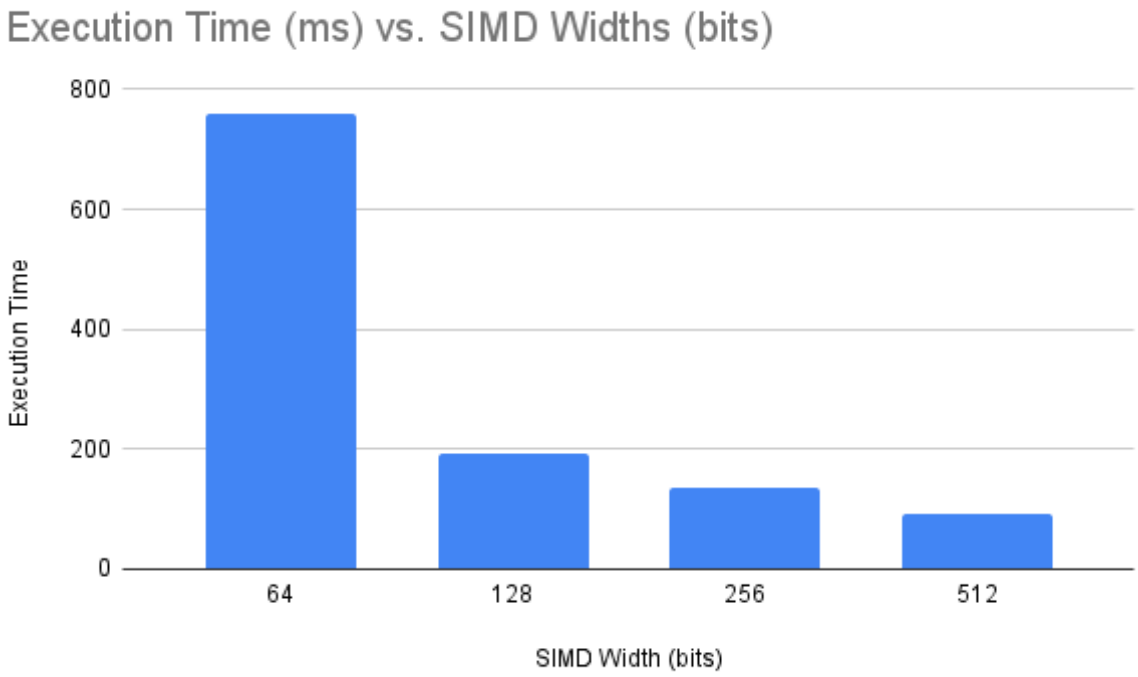

Figure 12: Instruction Count vs. SIMD Widths

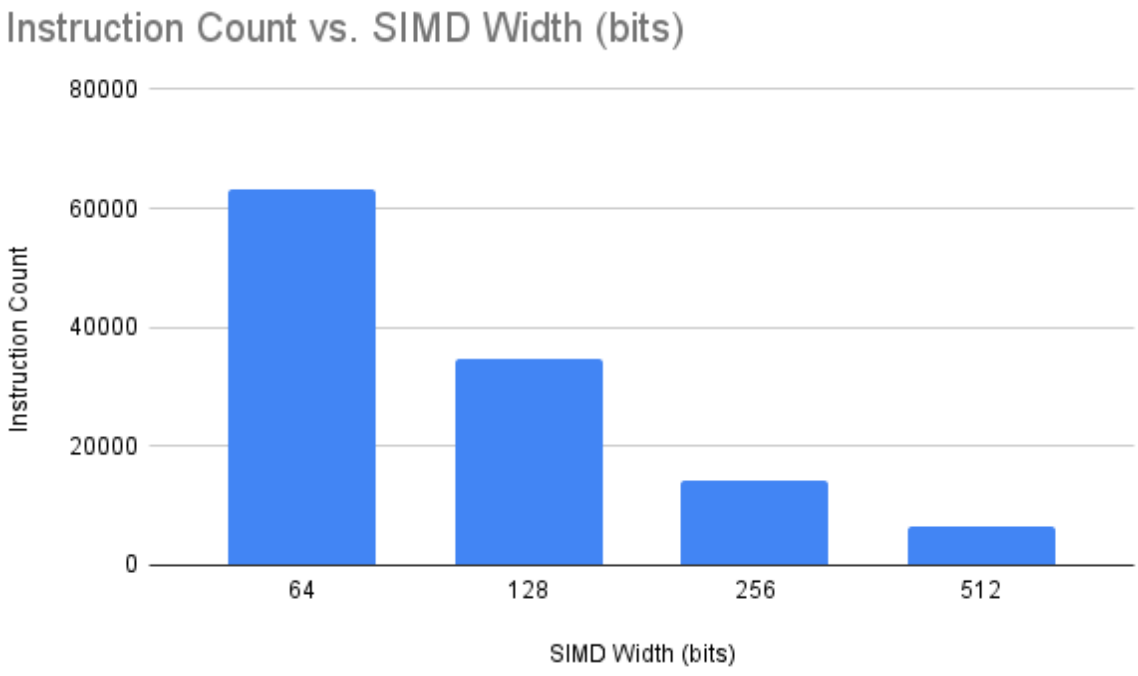

Figure 13: Instruction Count vs. SIMD Widths

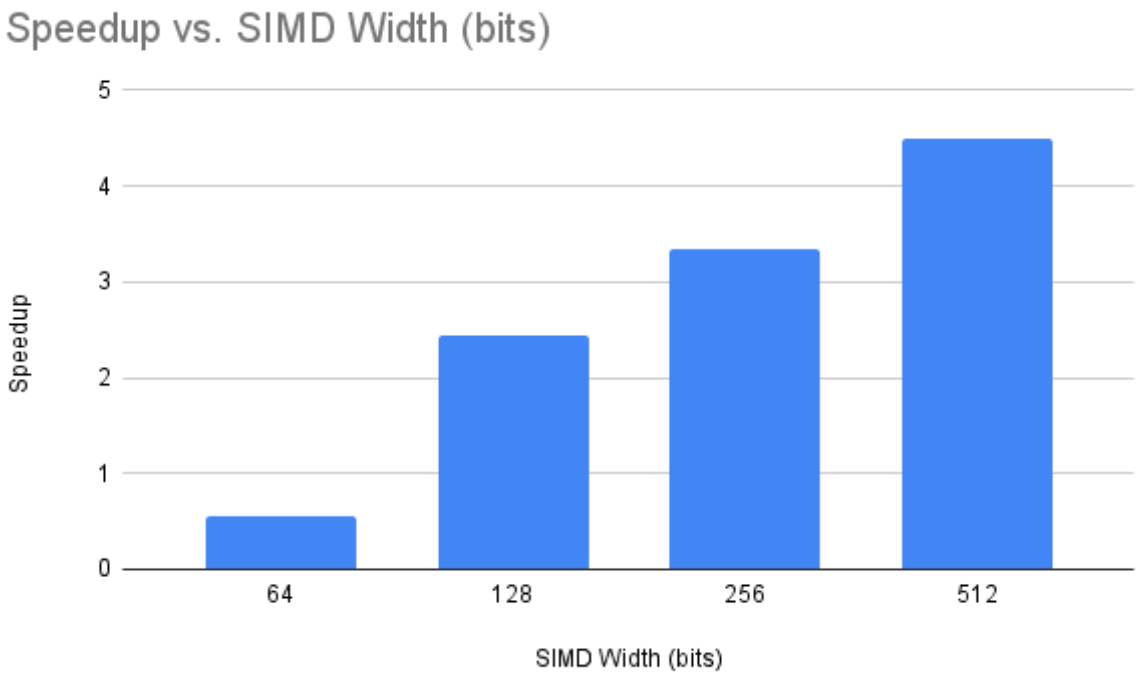

Figure 14: Instruction Count vs. SIMD Widths

2.3 Discussion

- **What trends do you observe in speedup for different combinations of embedding dimensions and SIMD widths?**

  From Figure 6 and Figure 3 we can conclude that as SIMD widths get larger, speedup increases and as Embedding Dimensions gets larger, speedup decreases.

  The worst speedup versus SIMD width was for 64bit. It had to be implemented using 128 bit SIMD registers as there was no 64 bit SIMD register available for use. 128 bit SIMD comes under SSE which is still in use in modern CPUs but 64 bit is outdated and was last used by CPUs with MMX.

- **For which SIMD width do you achieve the maximum speedup?**

  512 bit width SIMD register gives the maximum speedup as its able to compute 16 single precision floats in a single instruction.

- **How does the instruction count change with SIMD width?**

  Larger SIMD widths should decrease the instruction count in theory as they combine multiple instructions into a single SIMD instruction computed in SIMD registers. Our results correctly correspond the theoretical hypothesis. 512 bit width SIMD gives the least instruction count and the number of instructions increases by the SIMD factor. Halving the SIMD width approximately doubles the instruction count.

3 Part C: SIMD and Prefetch+SIMD

4 Conclusion