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 well as we expected. We were unable to get a speedup of more than 2x.

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

1.2 Results

1.2.1 Varying Embedding Table Size

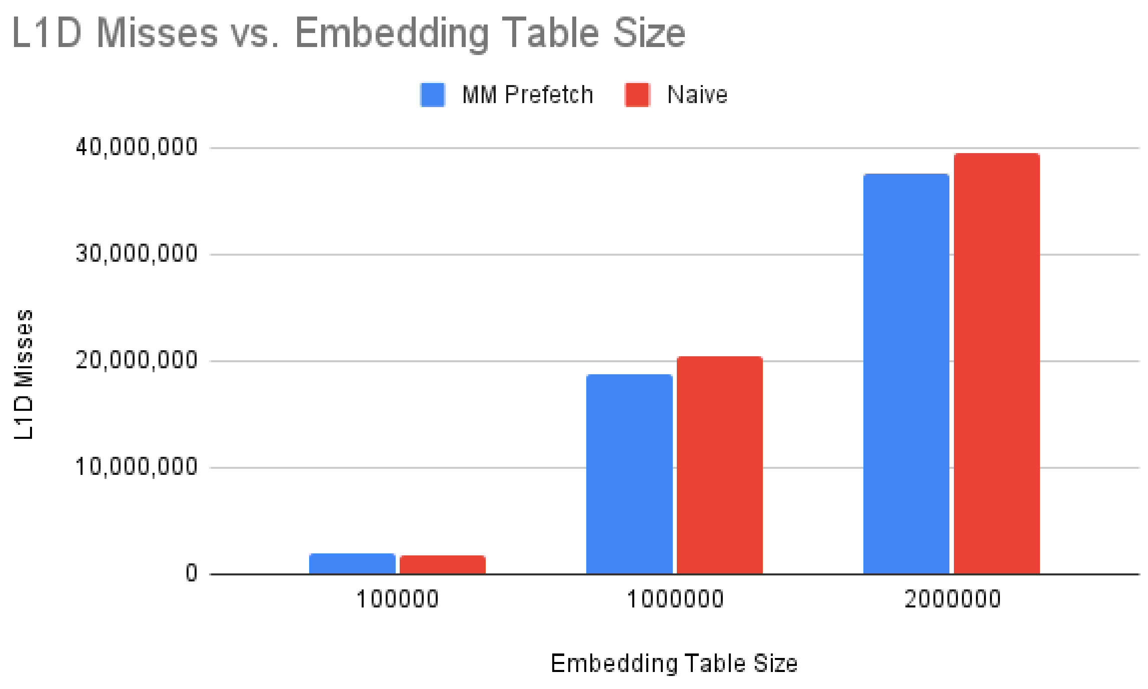

Figure 1: L1D misses vs. Embedding Table Sizes

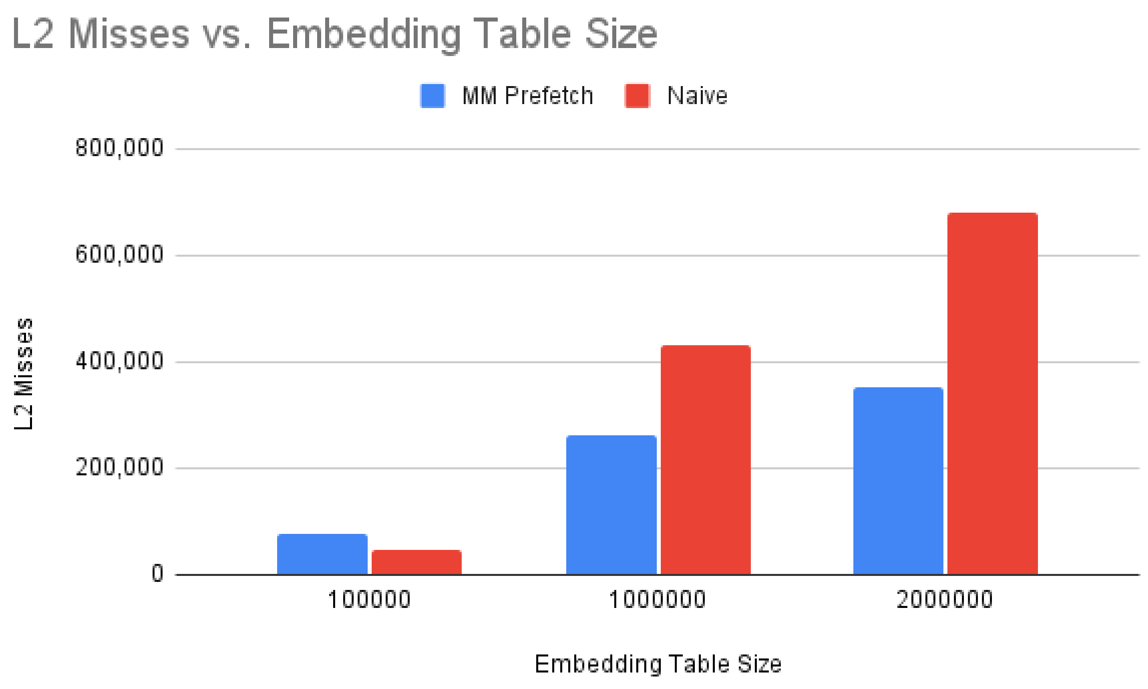

Figure 2: L2 misses vs. Embedding Table Sizes

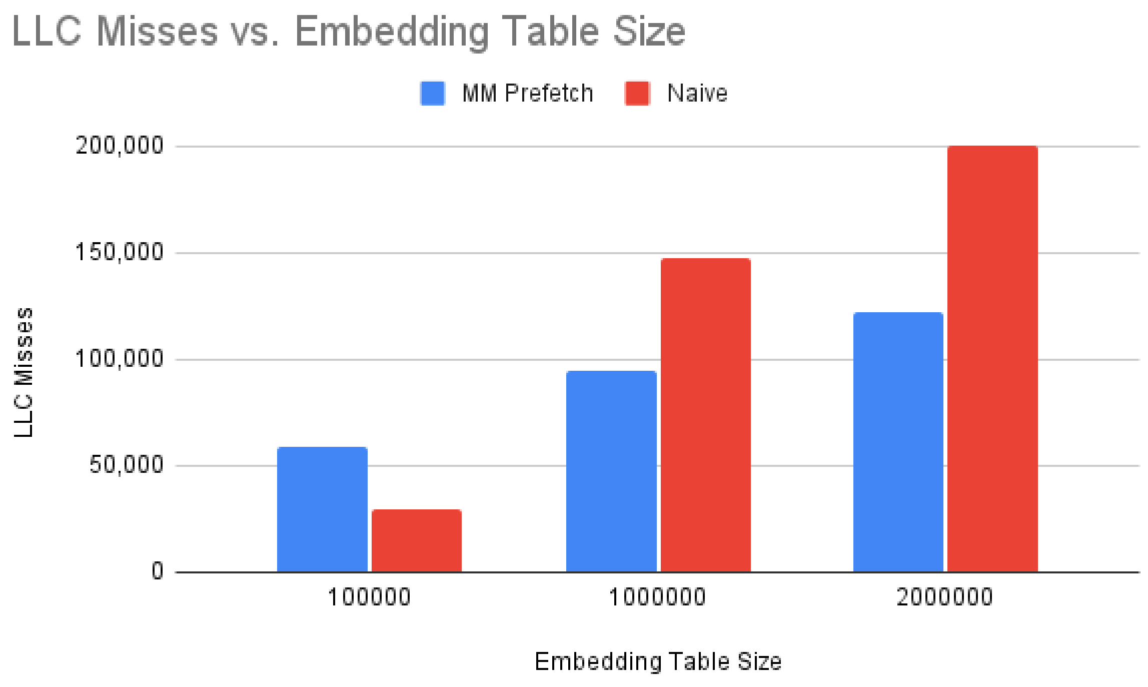

Figure 3: LLC misses vs. Embedding Table Sizes

The above analysis was for prefetch distance of 6. The results are as expected. Prefetching reduces the number of misses across all cache levels. One thing we noticed is that the naive version had fewer misses when the embedding table was small. We believe this is because for a small embedding table, the hardware is able to hide the latency well enough that prefetching does not really help. $100000 \times 128 \times 4B$ 51MB means that this table can fit entirely in LLC so hardware prefetching works really well. While in larger tables, the data will come from main memory and software prefetching helps as the data is random.

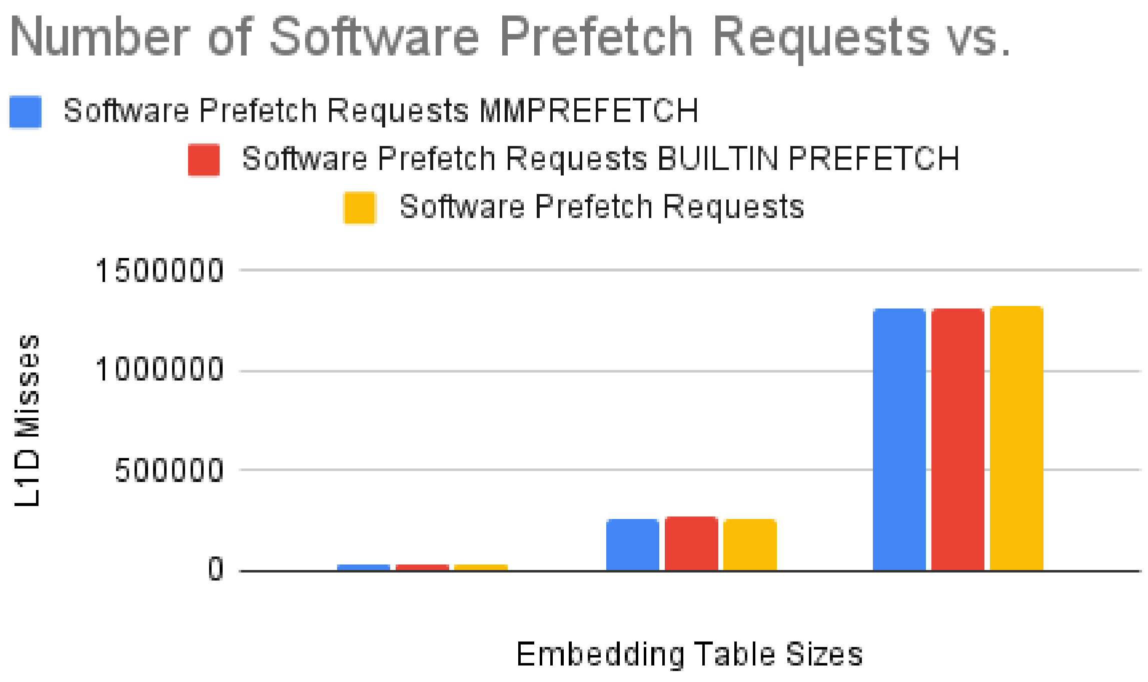

Figure 4: Number of Software Prefetch Requests vs. Embedding Table Sizes

Surprisingly we did not find a very big difference in the number of software prefetch requests. A big reason for that is how perf counts this stat. It includes prefetch requests for system calls, main function, etc. The number of requests issued by our prefetch function is less than 5% of the total.

1.2.2 Speedup

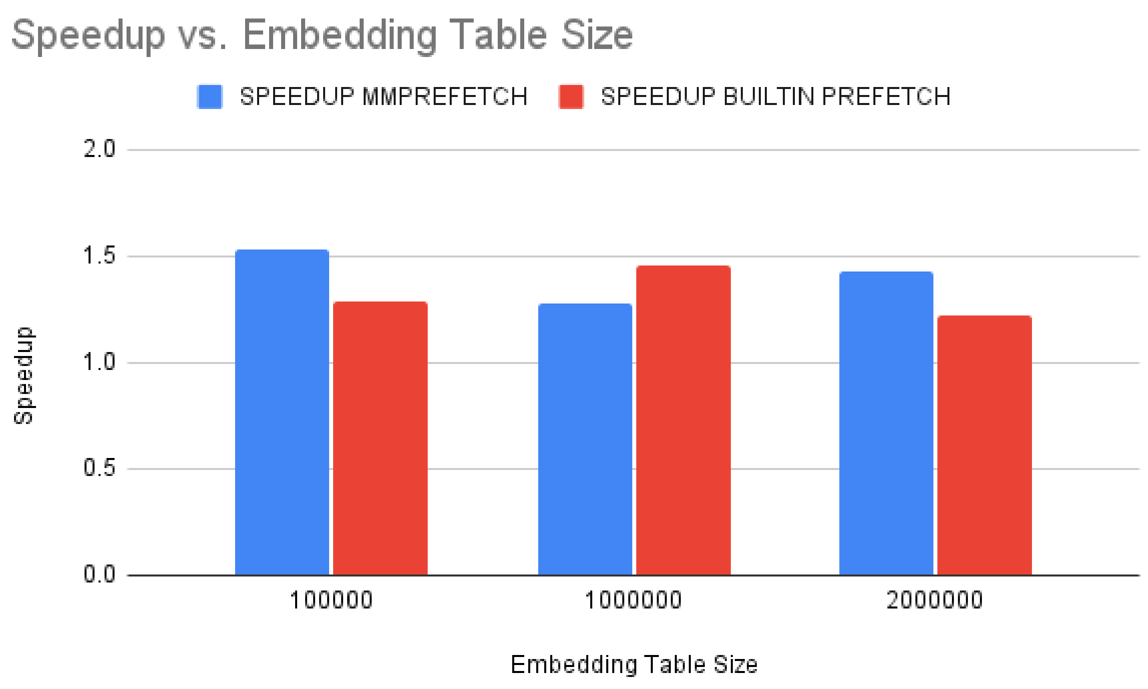

Figure 5: MM vs. Builtin Prefetch

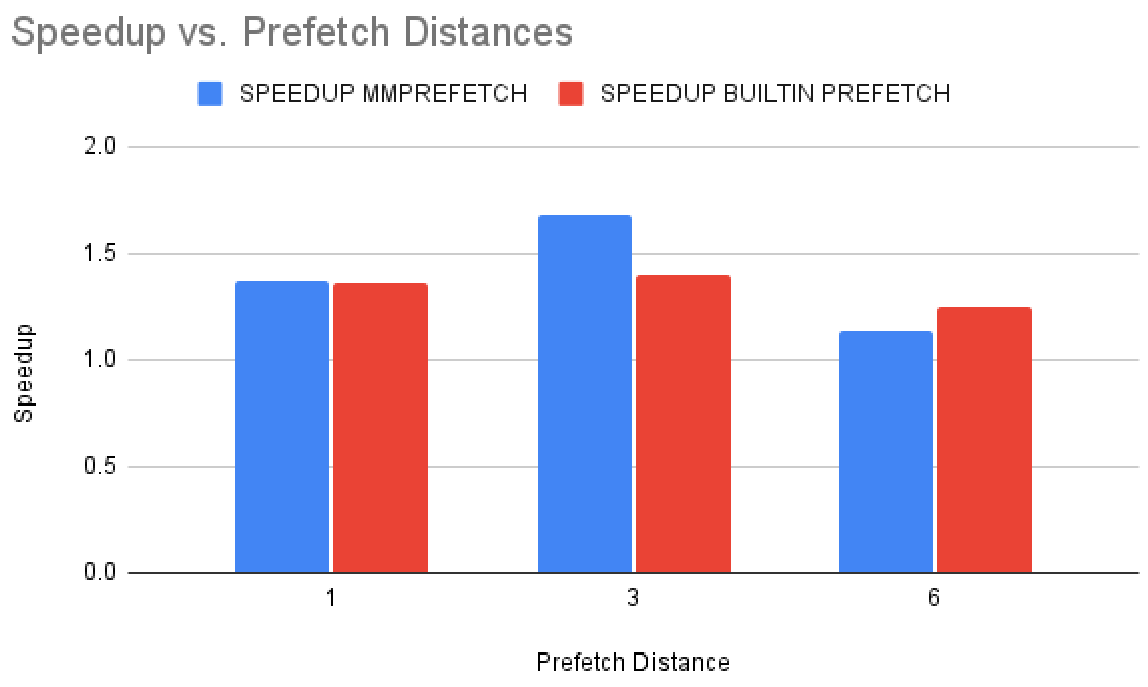

Figure 6: MM vs. Builtin Prefetch

We got a consistent speedup between 1.22x - 1.53x. There was no clear trend here against table size or prefetch distance. Prefetch distance of 3 gave us maximum speedup across both prefetchers.

1.3 Varying Prefetch Distance

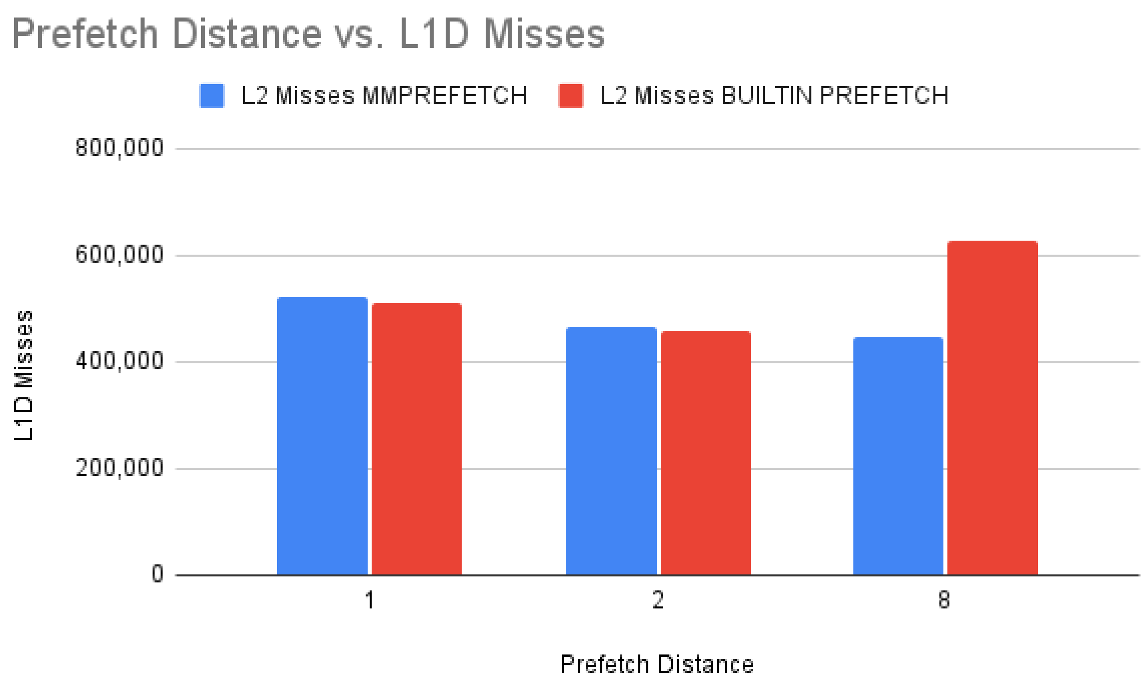

Figure 7: Prefetch Distance vs. L1D Misses

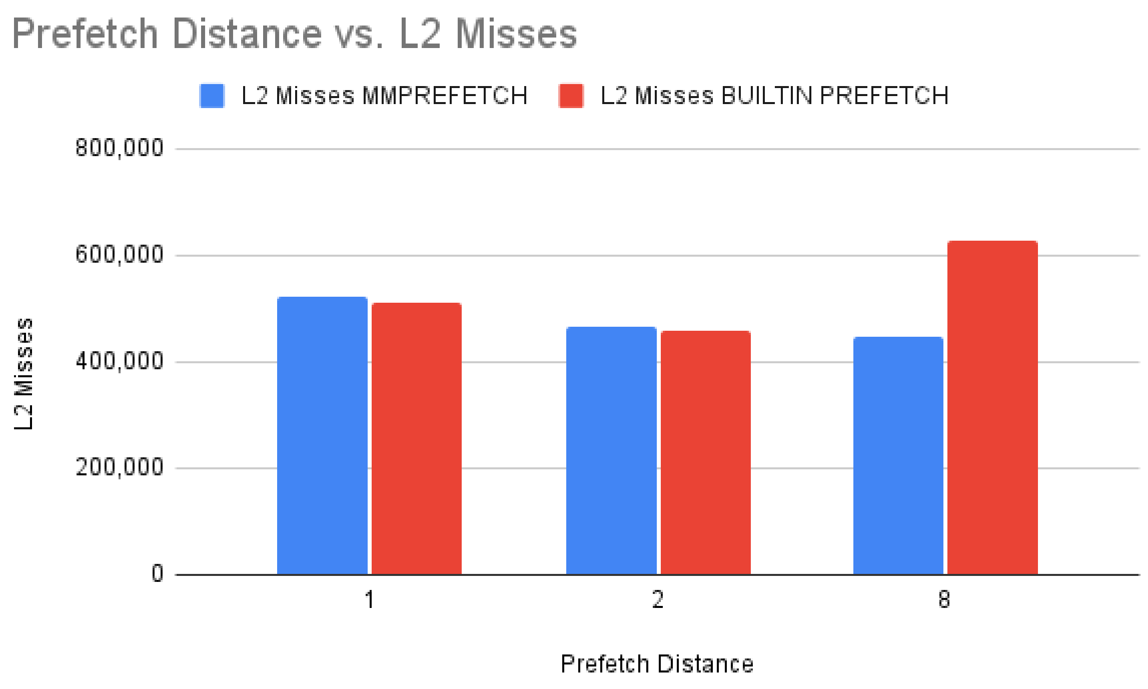

Figure 8: Prefetch Distance vs. L2 Misses

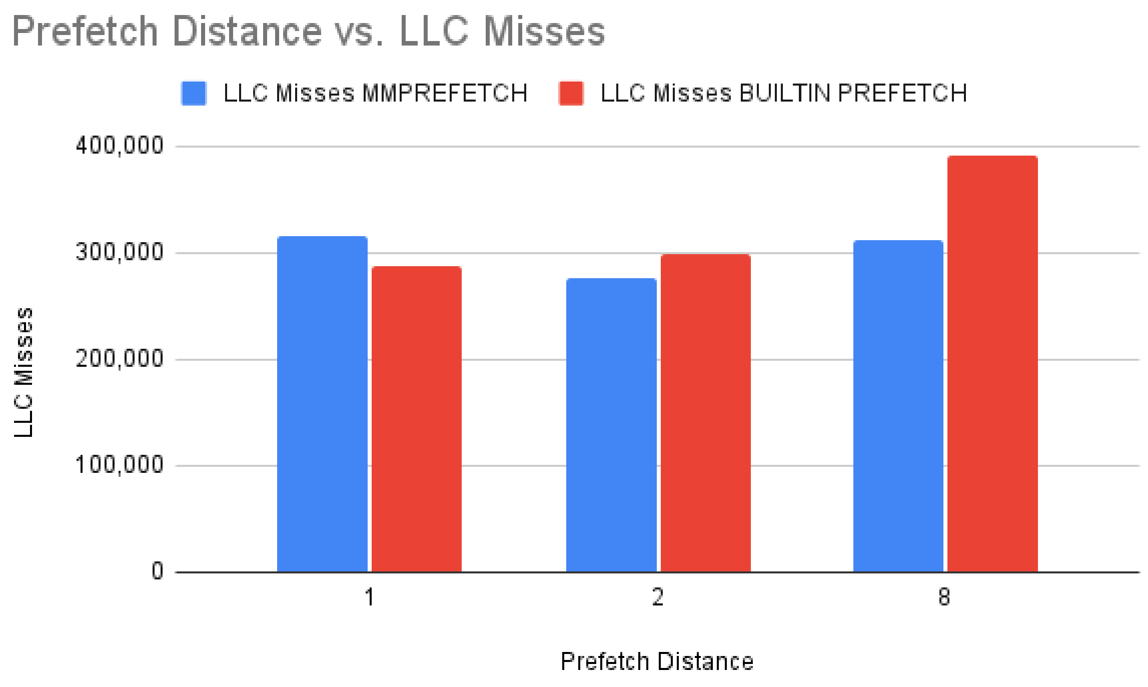

Figure 9: Prefetch Distance vs. LLC Misses

This was the most ambiguous part of the experiment. Every run gave a different answer and we did not see a clear trend here. Varying the prefetch distance did alter the speedups but the change in number of misses is either too small to notice in perf stats or is missing completely.

1.3.1 Varying Cache Fill Level

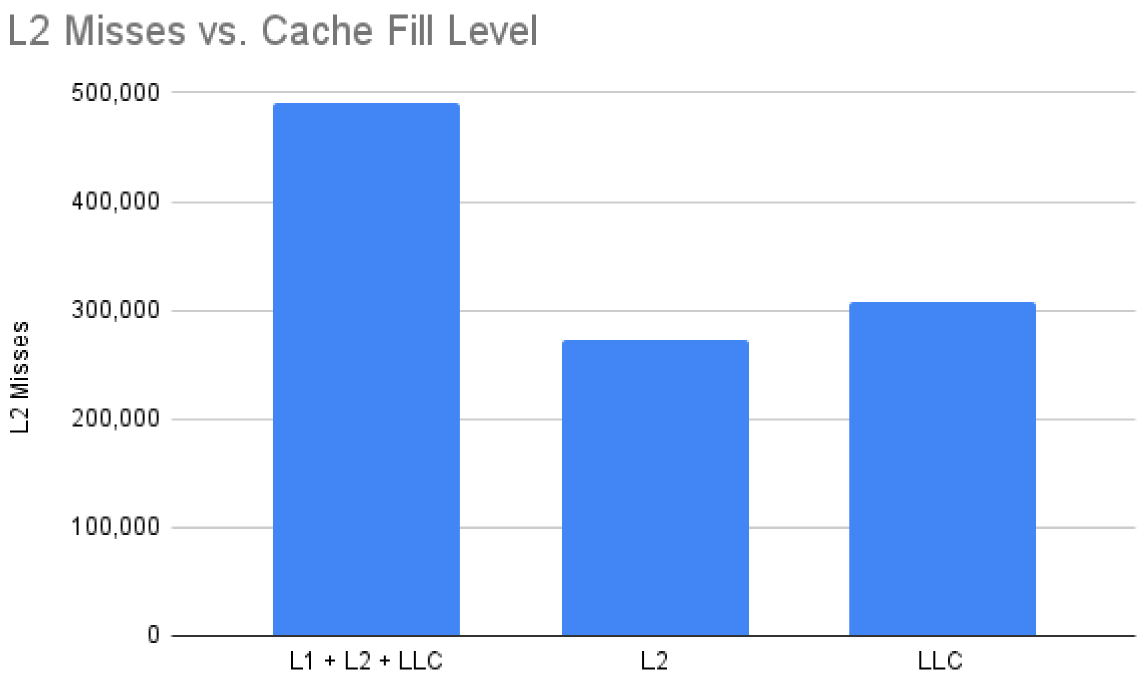

Figure 10: L2 Misses vs. Cache Fill Level

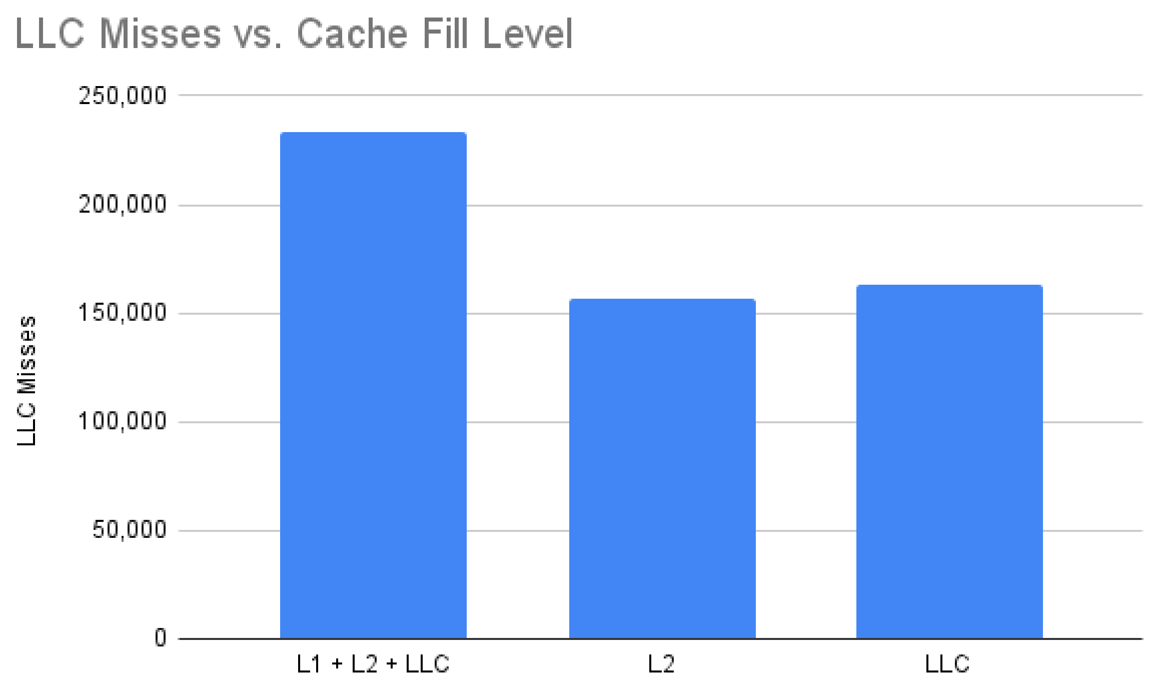

Figure 11: LLC Misses vs. Cache Fill Level

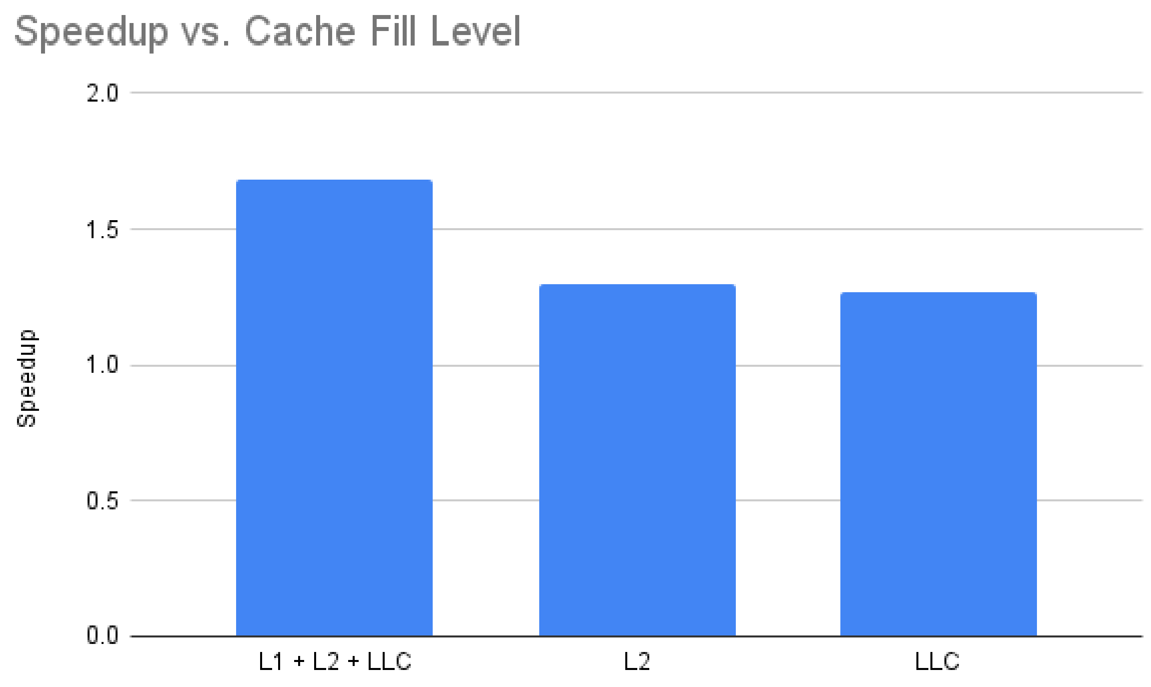

Figure 12: Speedup vs. Cache Fill Level

1.4 Discussion

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

Prefetch distance of 3 turned out to be the best for us.

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

  When the data is in all cache levels, we get maximum speedup.

  Prefetch Hints:

  - T0: All cache levels. This includes L1 so it is used when you want data to be as close as possible to the processor.
  - T1: All cache levels above L2.
  - T2: All cache levels above LLC.
  - NTA: No temporal data. Used for streaming.

  Here **L1+L2+LLC** implies using the T0 hint, which is the maximum speedup.

  We can observe that when cache level is L2 or higher, the number of L2 misses becomes significantly lower (50% decrease). When level is LLC or higher, both L2 and LLC misses fall by 50% atleast.

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

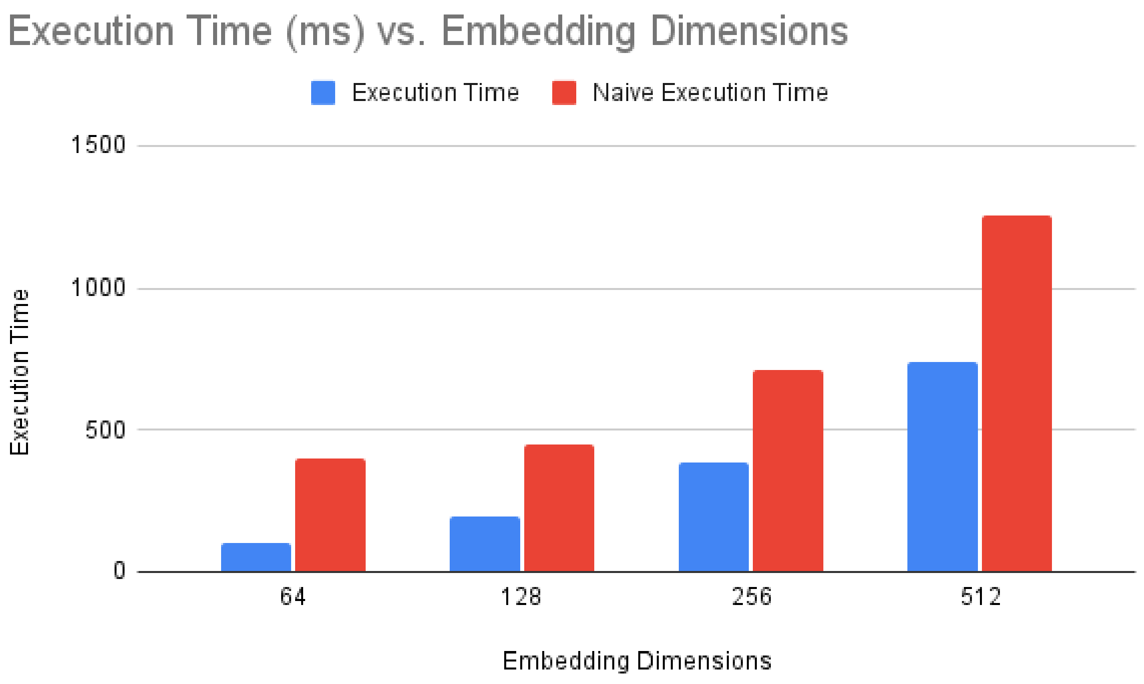

Figure 13: Execution Time vs. Embedding Dimensions

H

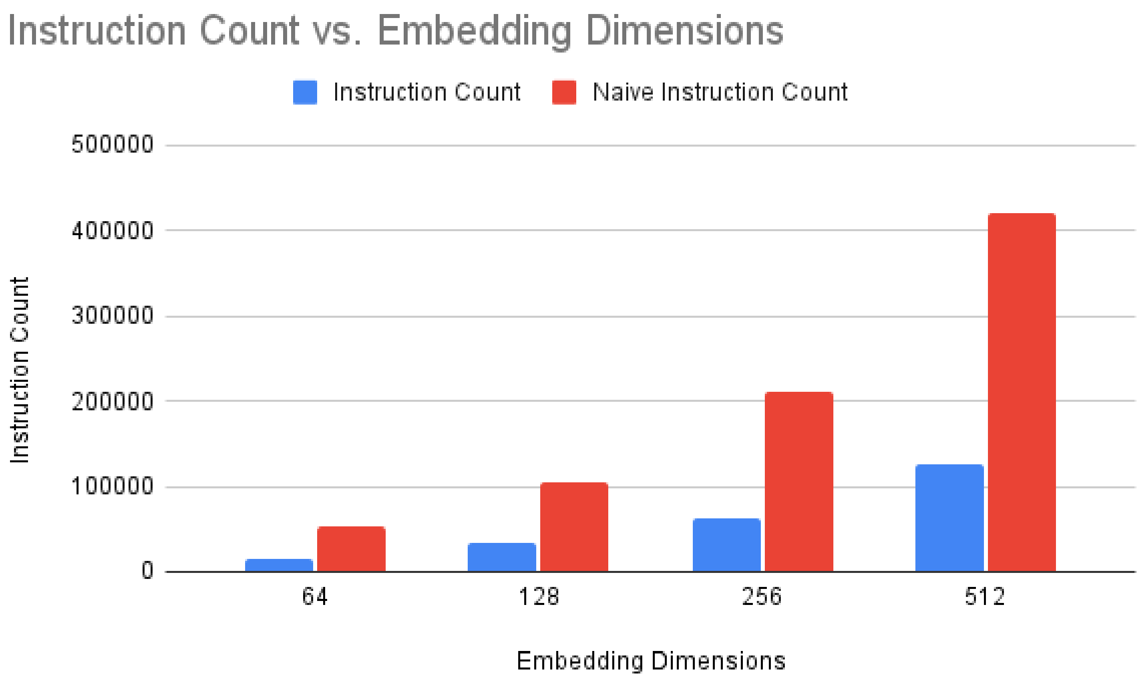

Figure 14: Instruction Count vs. Embedding

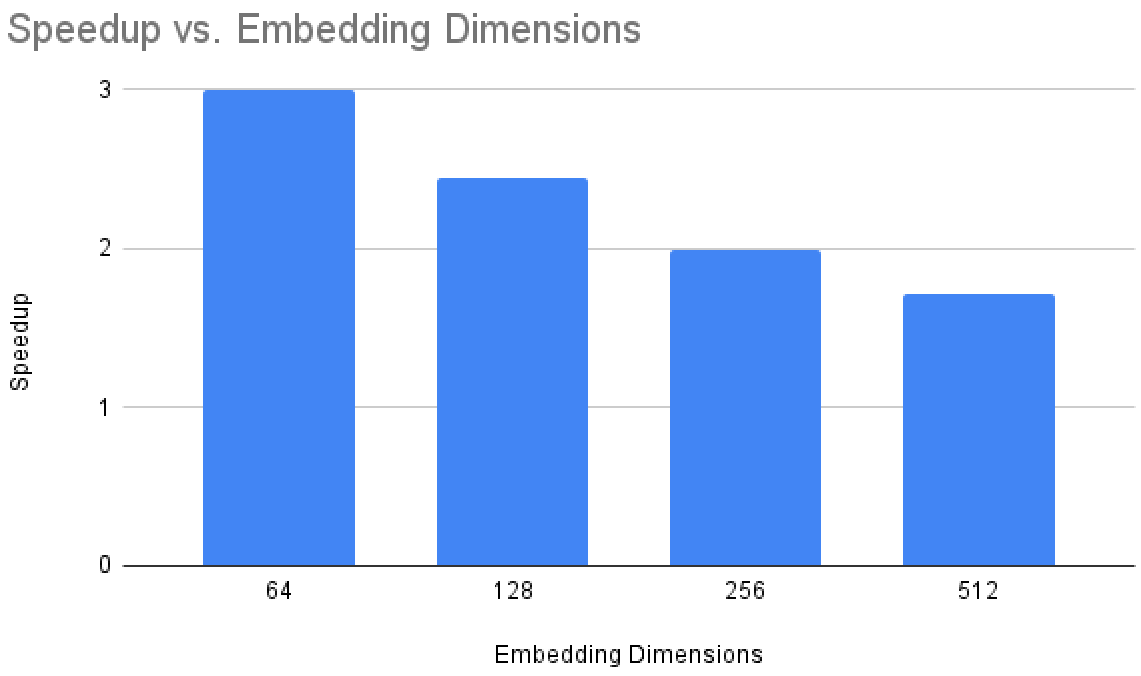

Figure 15: Speedup vs. Embedding Dimensions

2.2.2 Varying SIMD Widths

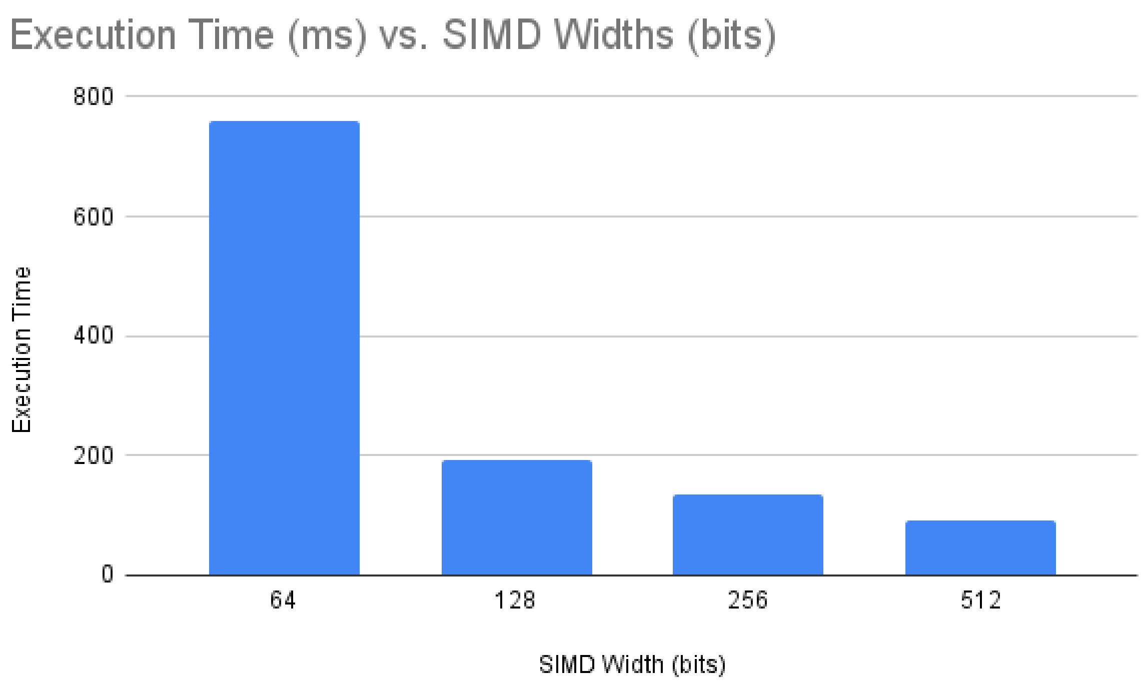

Figure 16: Instruction Count vs. SIMD Widths

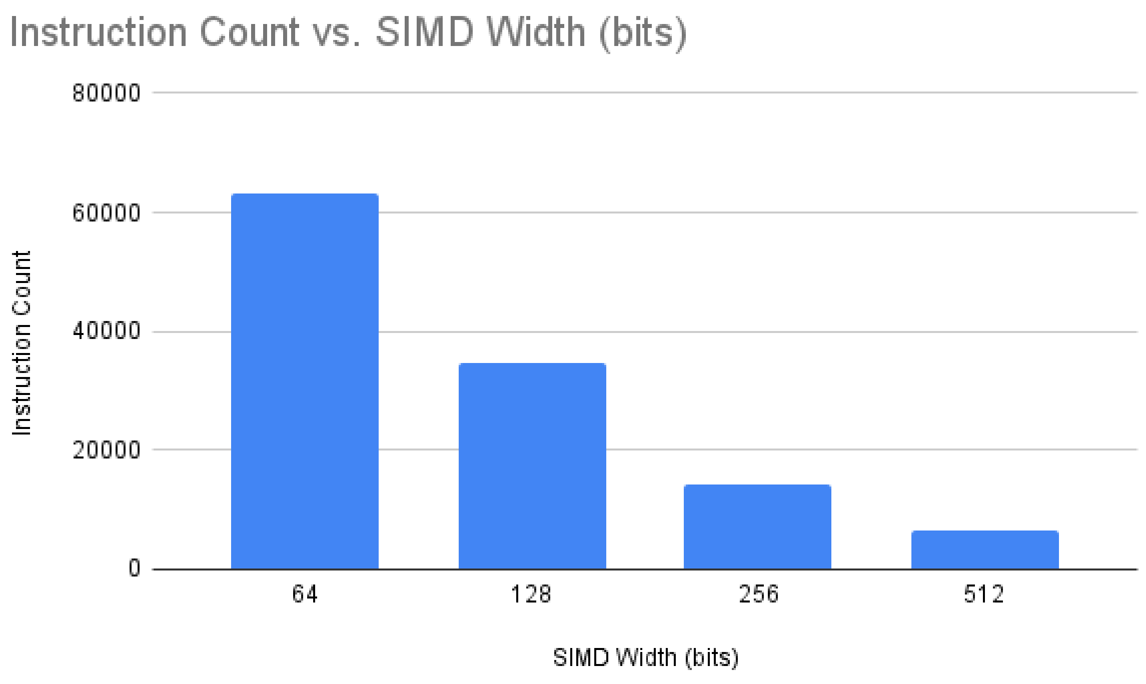

Figure 17: Instruction Count vs. SIMD Widths

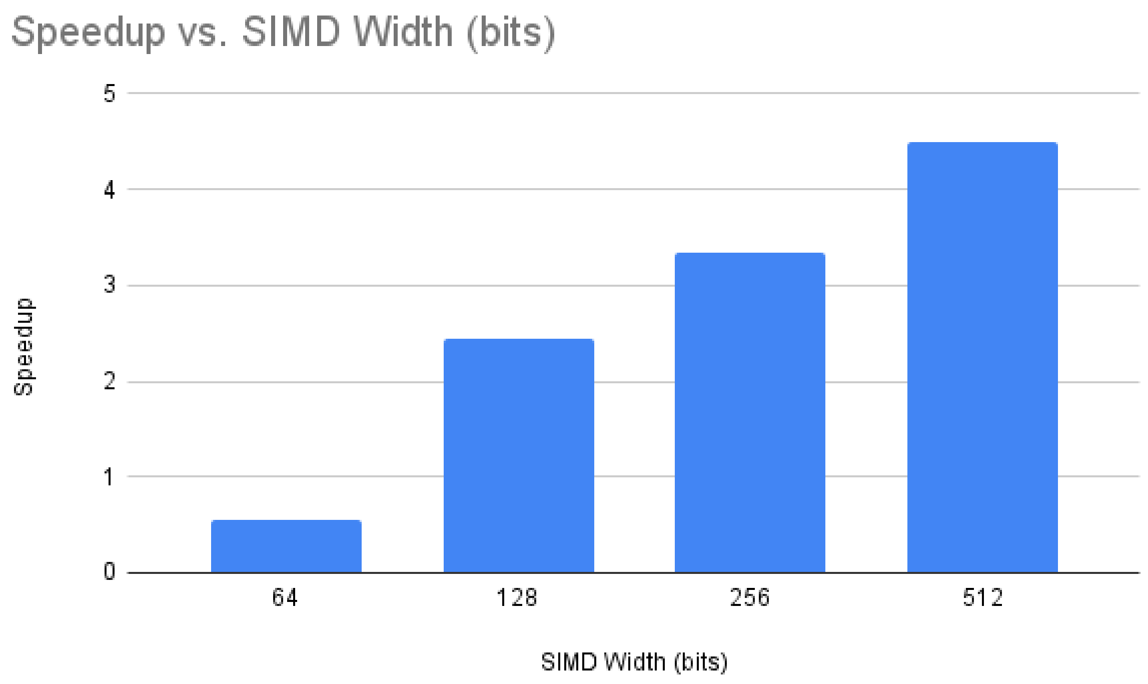

Figure 18: Instruction Count vs. SIMD Widths

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: Prefetch+SIMD

3.1 Code Implementation

We used the best values for the tunable parameters we found in 2A and 2B. For software prefetching, we took T0 as the cache fill level.

3.2 Results

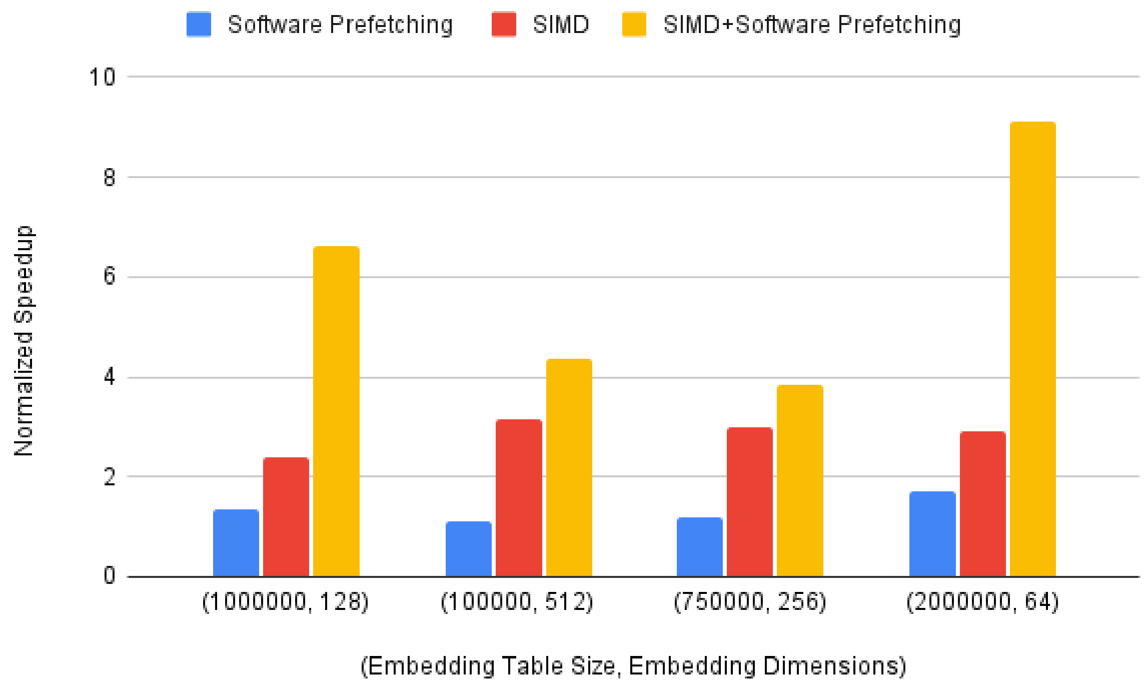

Figure 19: Combined Speedup