Programming Assignment 1
THREE GENIUSES

Task 1E: Confusion hi confusion hai !!

Analysis Report (techniques and results)

We tried different ways to speed up dense matrix multiplication:

• Naive simple loop (ijk) — This is the basic version. It’s easy to understand but not fast, because it keeps reading data from memory over and over, which slows things down.

• Loop reordering (ikj) — We changed the order of the loops so that values from matrix A are reused more before they are thrown out of cache. This made a big difference, especially for bigger matrices, cutting the running time roughly in half for sizes 1024×1024 and larger.

• Loop unrolling — We manually expanded the inner loop to reduce the small overhead of looping. In practice, this didn’t help much, and sometimes it even slowed things down for large matrices. It didn’t fix the main problem (too much memory reading) and also put more pressure on CPU registers.

• Tiling (blocking) — We split the matrices into smaller pieces (tiles) that fit in cache. This kept the needed parts of A and B in fast memory while working on them. It worked as well as or slightly better than loop reordering, especially for large matrices.

• SIMD vectorization (AVX2) — We made the CPU do several multiplications and additions at once in a single instruction. This helped for small matrices (where memory isn’t the main problem) but didn’t help much for large matrices, because the CPU was still waiting for data from memory.

Summary: The best improvements came from loop reordering and tiling. Both reduce the time spent moving data in and out of memory, which is the real slowdown in large matrix multiplication. SIMD can give some extra boost when combined with these smarter memory techniques, but on its own it’s not enough. Loop unrolling which provides ILP by itself is rarely useful without other changes.

Matrix Multiplication Optimization Performance Analysis

Performance Summary

Basic
719.7 ms
1.00x

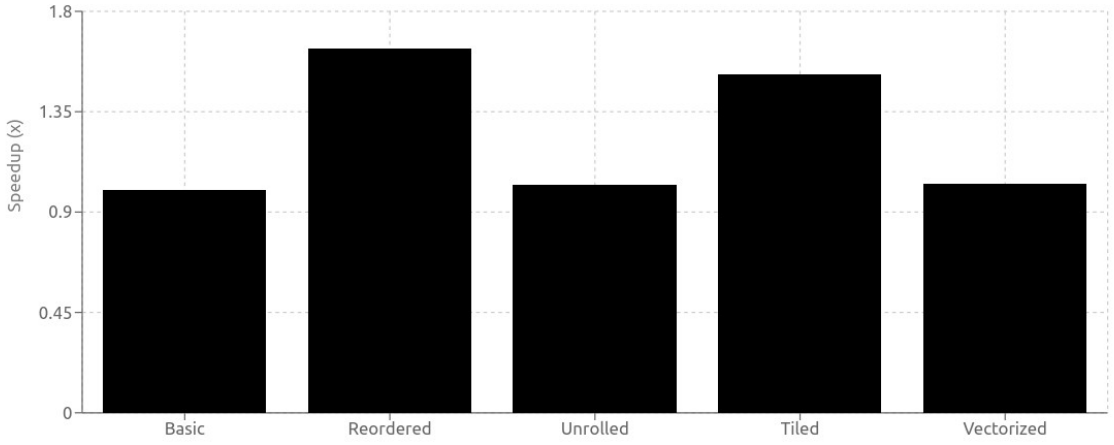
Reordered
440.1 ms
1.64x

Unrolled
705.1 ms
1.02x

Tiled
474.4 ms
1.52x

Vectorized
701.4 ms
1.03x

Speedup Comparison



Comparative Analysis

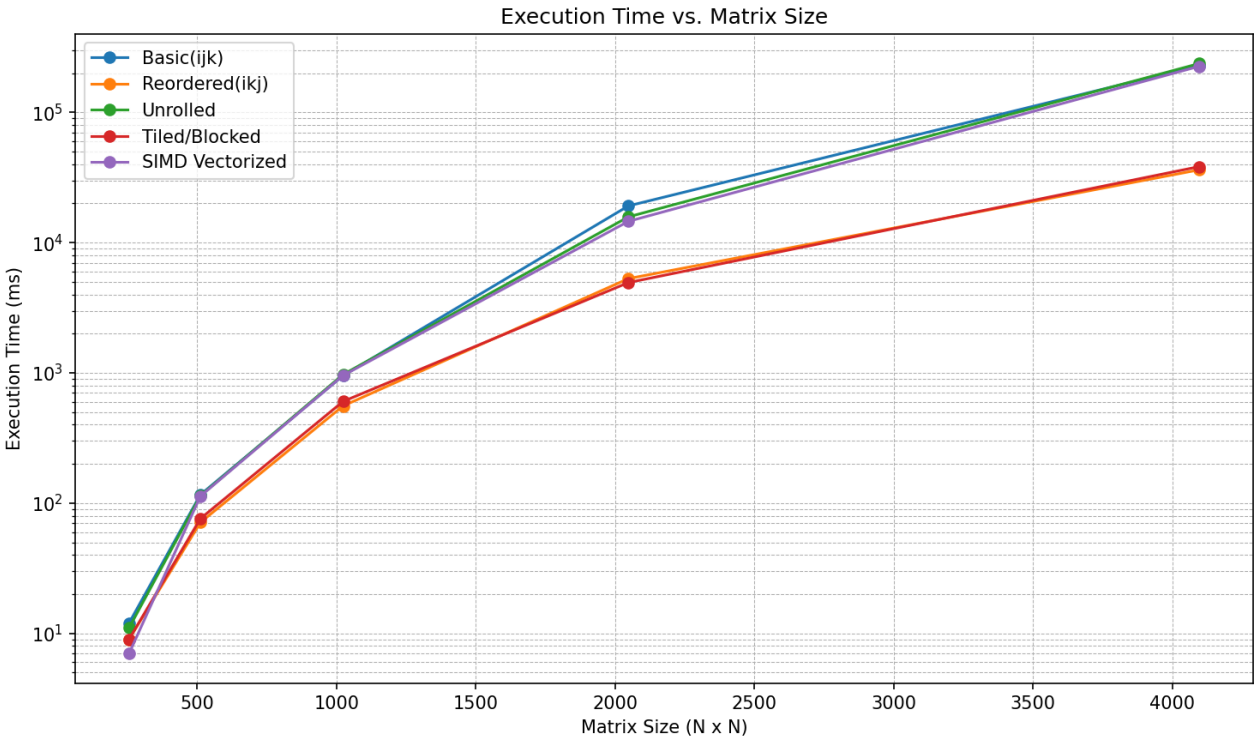
Absolute performance operations per second)

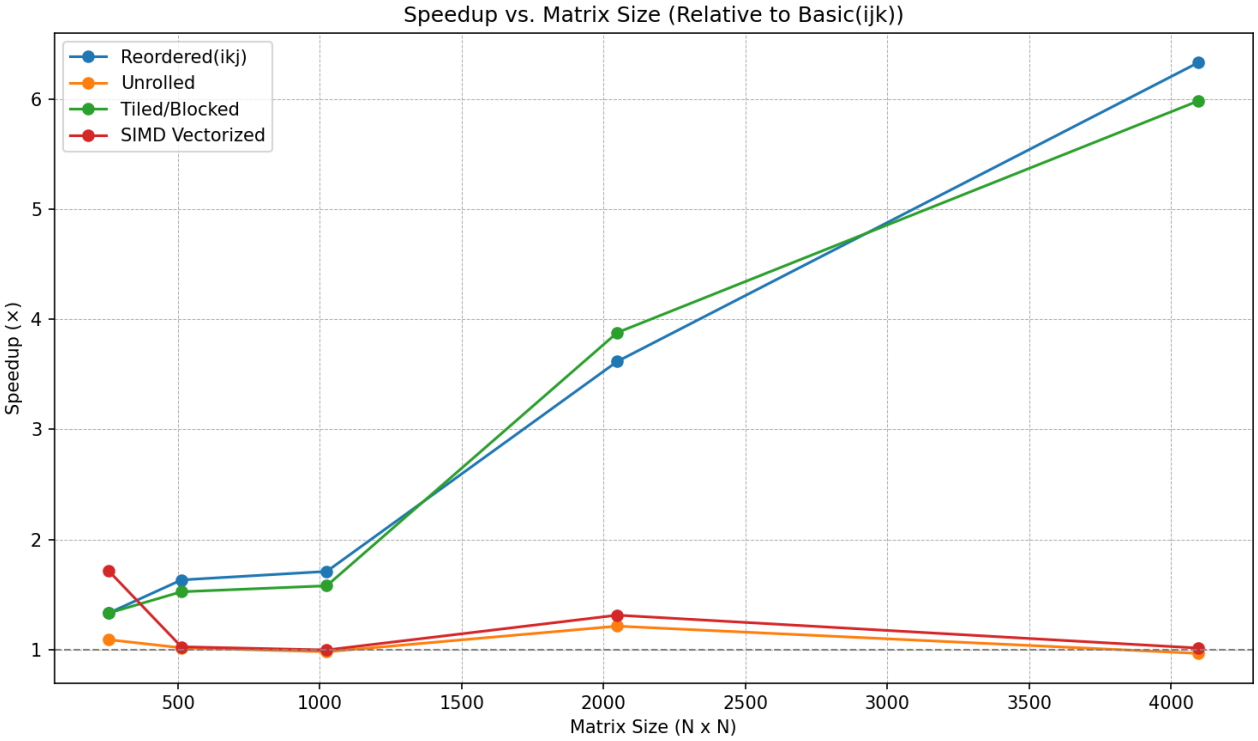
The basic (ijk) version is always the slowest.
For small matrices (like 256×256), the SIMD version wins, the CPU can do several multiplications and additions at once, and all the needed data still fits in cache.
For medium matrices (512 and 1024), the reordered (ikj) and tiled versions pull ahead. They keep the needed data in fast memory longer, which saves time.
For very large matrices (2048 and 4096), only reordered and tiled keep scaling well. SIMD and loop unrolling don’t help here, because they don’t fix the real problem: waiting for data to arrive from memory.

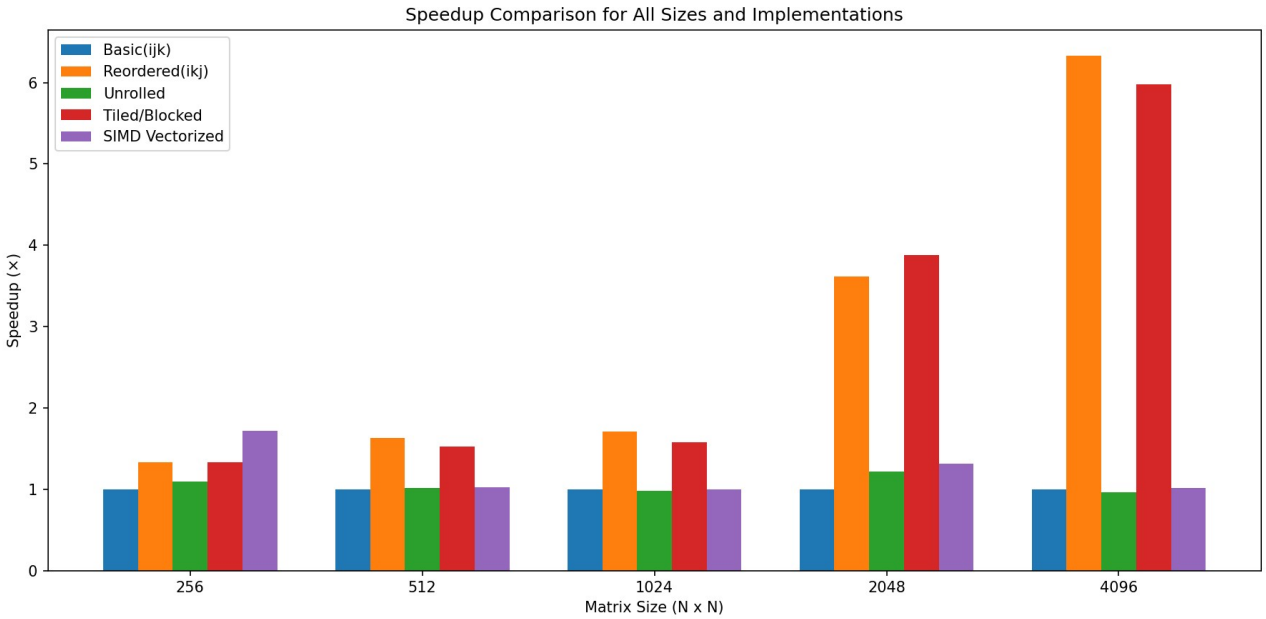
Speed Up
At size 256: SIMD is about 1.7× faster, reordered and tiled about 1.3× faster.
At sizes 512 and 1024: reordered and tiled are roughly twice as fast, SIMD barely improves things, and unrolling sometimes makes things worse.
At sizes 2048 and 4096: reordered is about 3–5× faster, tiled is in the same range or a little better. SIMD and unrolled versions drop back close to the slow basic version.

Memory access patterns
The basic version keeps reading the same parts of matrix B again and again, which wastes time and thrashes the cache.
The reordered (ikj) version fixes this by reusing data from A while computing multiple results before throwing it away.
The tiled version goes further, it works on small blocks of A and B that fit into cache, so it reuses data even more efficiently.
SIMD works on multiple numbers at a time but doesn’t change the way data is fetched, so it still reloads the same data often.
Unrolling only changes how the CPU runs the loops, not how it accesses memory, so it doesn’t help data reuse.

Cache efficiency
Reordered and tiled versions are built to use the cache well. They make sure each piece of data is used several times before being replaced. That’s why they keep performing well even as matrices grow.
SIMD only helps when data is already in the cache — it doesn’t help bring data there faster.
Unrolling just saves a little on loop overhead but doesn’t solve the real issue of moving data in and out of memory, so it doesn’t help much at large sizes.


Execution Time vs. Matrix Size
Basic(ijk)
Reordered(ikj)
Unrolled
Tiled/Blocked
SIMD Vectorized
Execution Time (ms)
10^5
10^4
10^3
10^2
10^1
500
1000
1500
2000
2500
3000
3500
4000
Matrix Size (N x N)


Speedup vs. Matrix Size (Relative to Basic(ijk))
Reordered(ikj)
Unrolled
Tiled/Blocked
SIMD Vectorized
Speedup (x)
6
5
4
3
2
1
500
1000
1500
2000
2500
3000
3500
4000
Matrix Size (N x N)


Speedup Comparison for All Sizes and Implementations
Basic(ijk)
Reordered(ikj)
Unrolled
Tiled/Blocked
SIMD Vectorized
Speedup (×)
0
1
2
3
4
5
6
256
512
1024
2048
4096
Matrix Size (N x N)