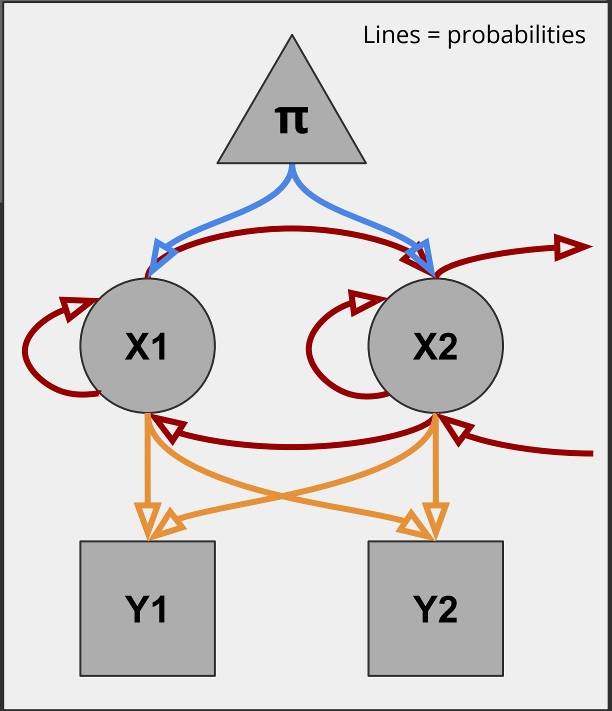
An optimized Baum-Welch algorithm

Cheuk Yu Chan
Franz Knobel
Josua Cantieni
Ramon Witschi

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Baum-Welch algorithm

- Computes **Probability distribution** over a model
- The hidden states of a **hidden Markov model**
- Using **expectation-maximization**



We decide/are given

N: # hidden states

M: # observations

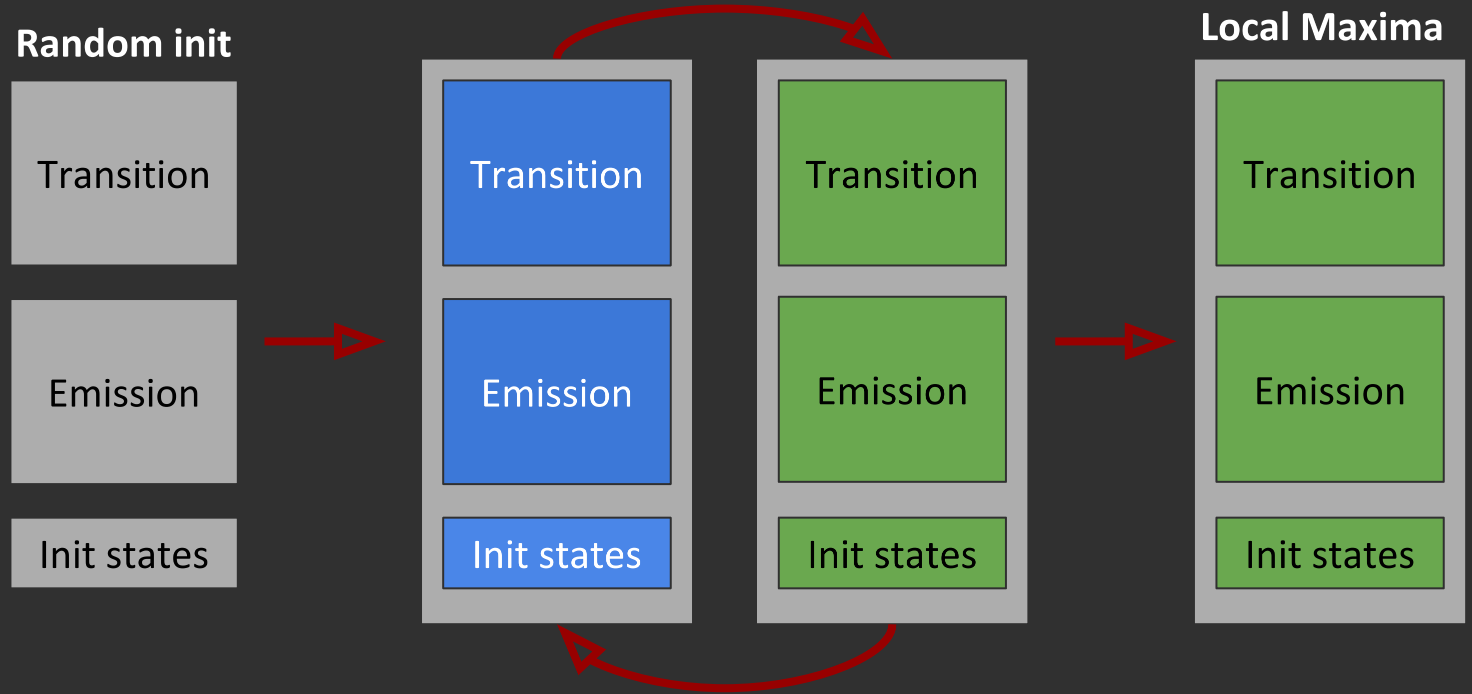
K: # observation sequences

T: # time steps

Max_iterations: # iterations to perform

Task of the Baum-Welch algorithm

Given observation sequence O1 O2 O3 O4 O5 O6 … … … OM



Cost Analysis

Asymptotic runtime per iteration:

$O(KTN * (M+N)) \approx O(N^2)$

Flops per iteration:

9*TKNN - 5*KNN + NN + 8*TKN + 2*KNM + 3*KN + 2*TK + NM + K +N Flops

Memory usage:

(max_iterations + KTNN + KNN + NN + KTN + NM +KN + 2*KT + N)*8 + 144 Bytes

Featured weight: NN

Baseline Implementation + Performance

Implementation:

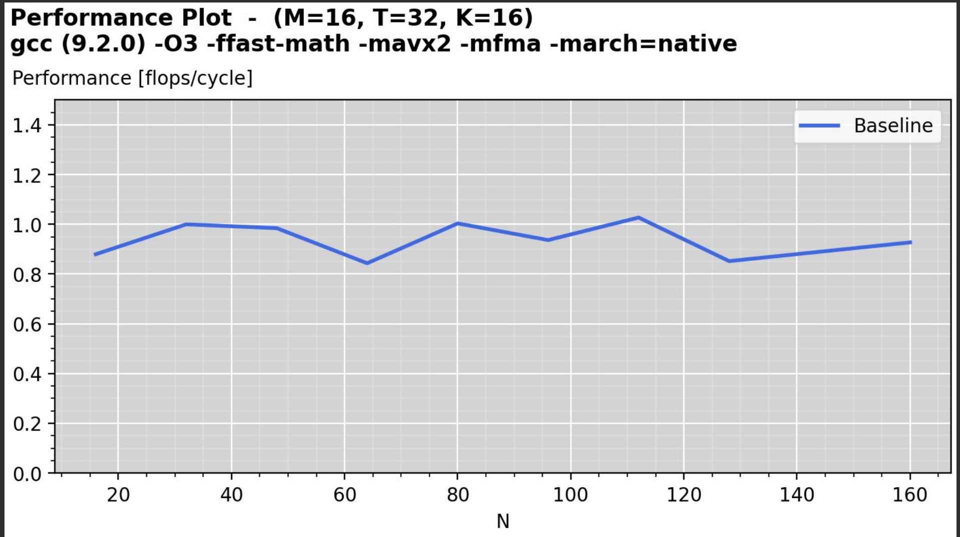
- Implemented code from the referenced tutorial slides
- Struct with pointer to arrays for convenience

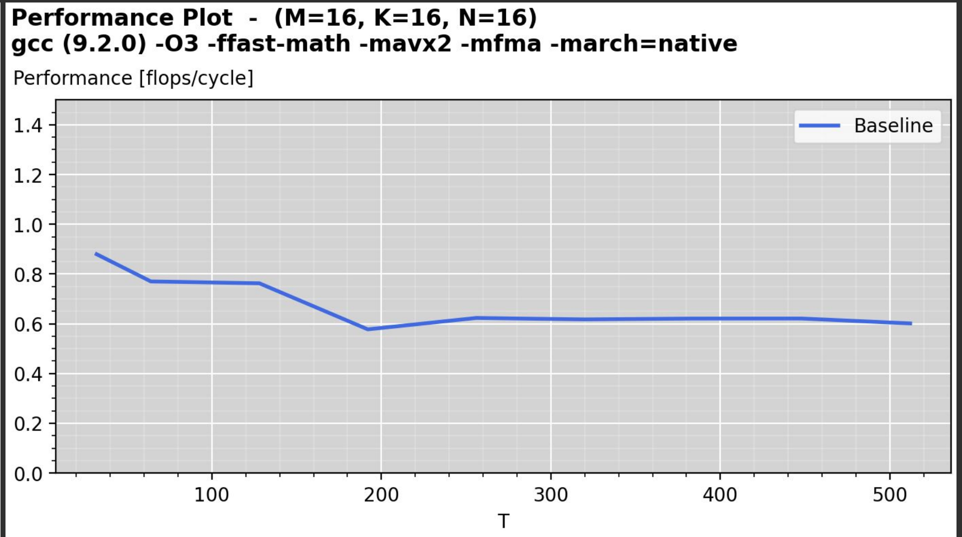
Assumptions:

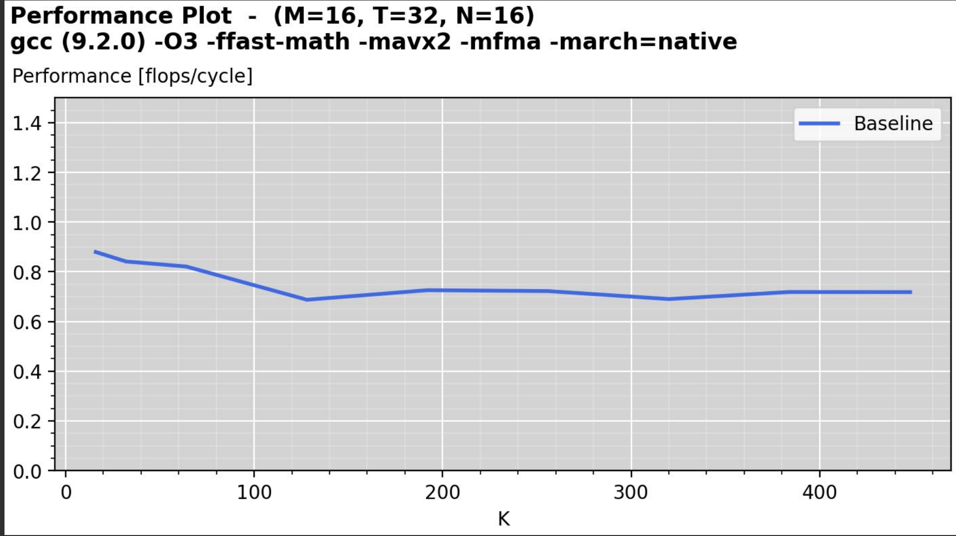
- Divisibility of N, M and K by 16 and T by 32

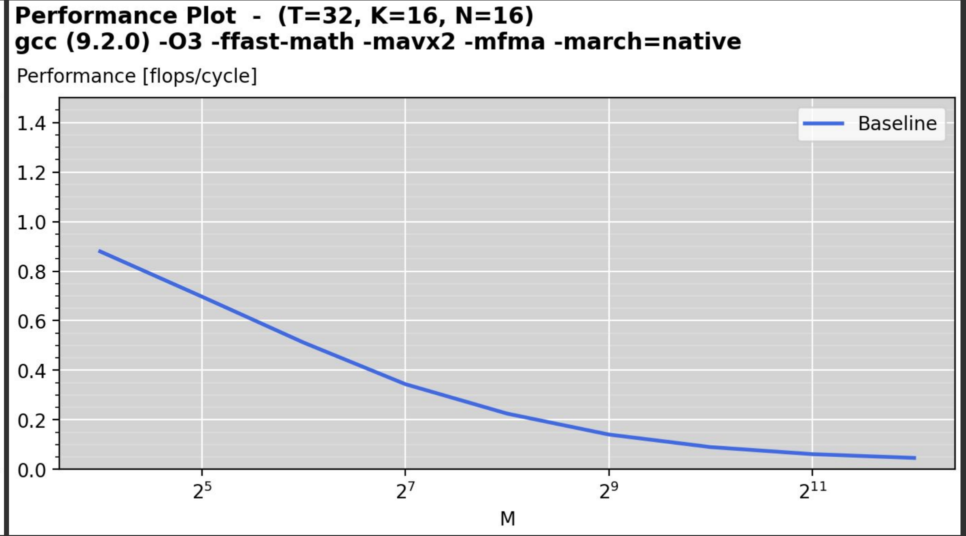
Verification:

1. Compare baseline against other implementations known to be correct
2. Compare our optimizations against our baseline

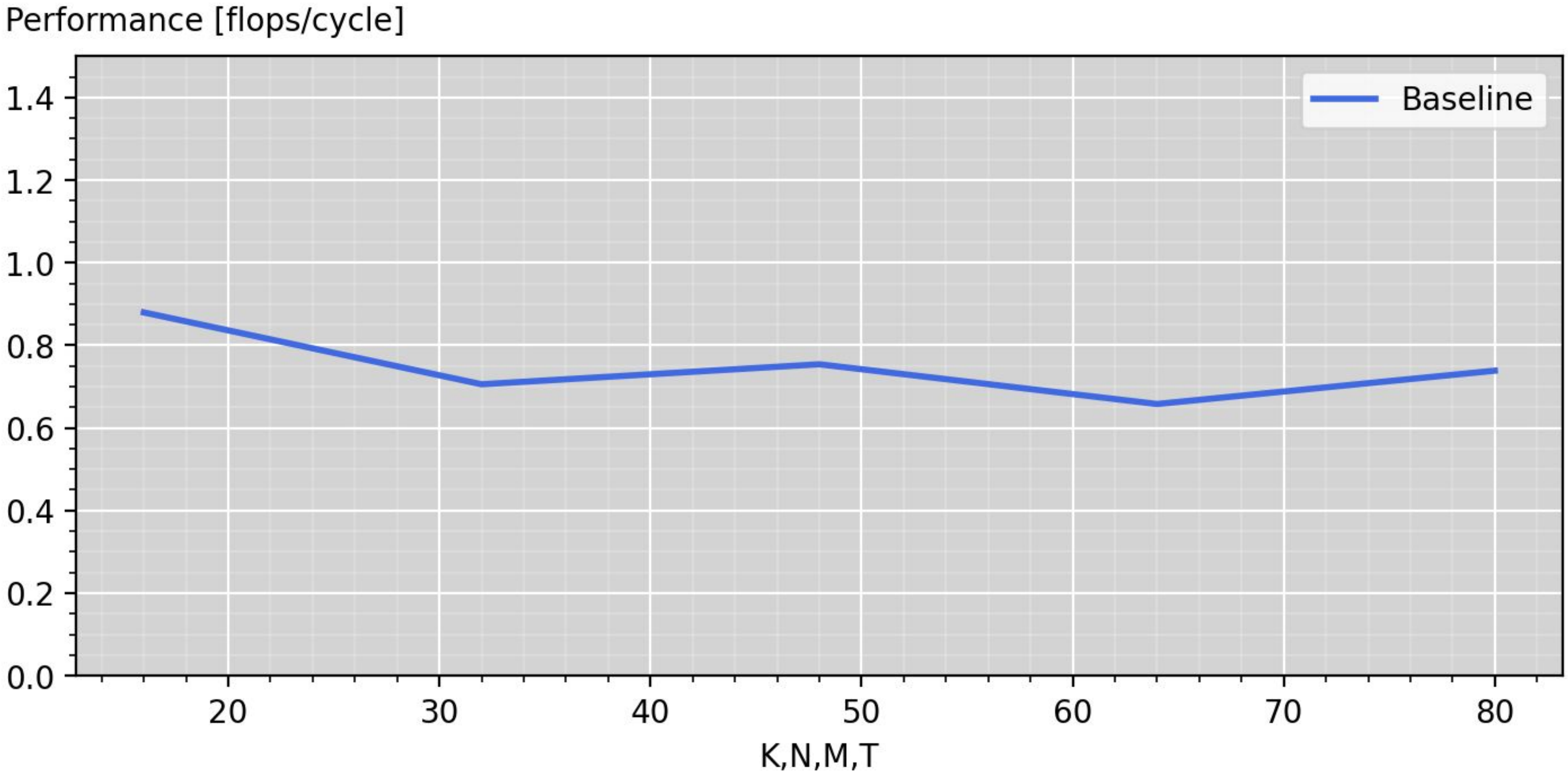

Performance Plot - (M=16, T=32, K=16)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
Baseline
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
20
40
60
80
100
120
140
160
N


Performance Plot - (M=16, K=16, N=16)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
Baseline
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
100
200
300
400
500
T


Performance Plot - (M=16, T=32, N=16)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
Baseline
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
K

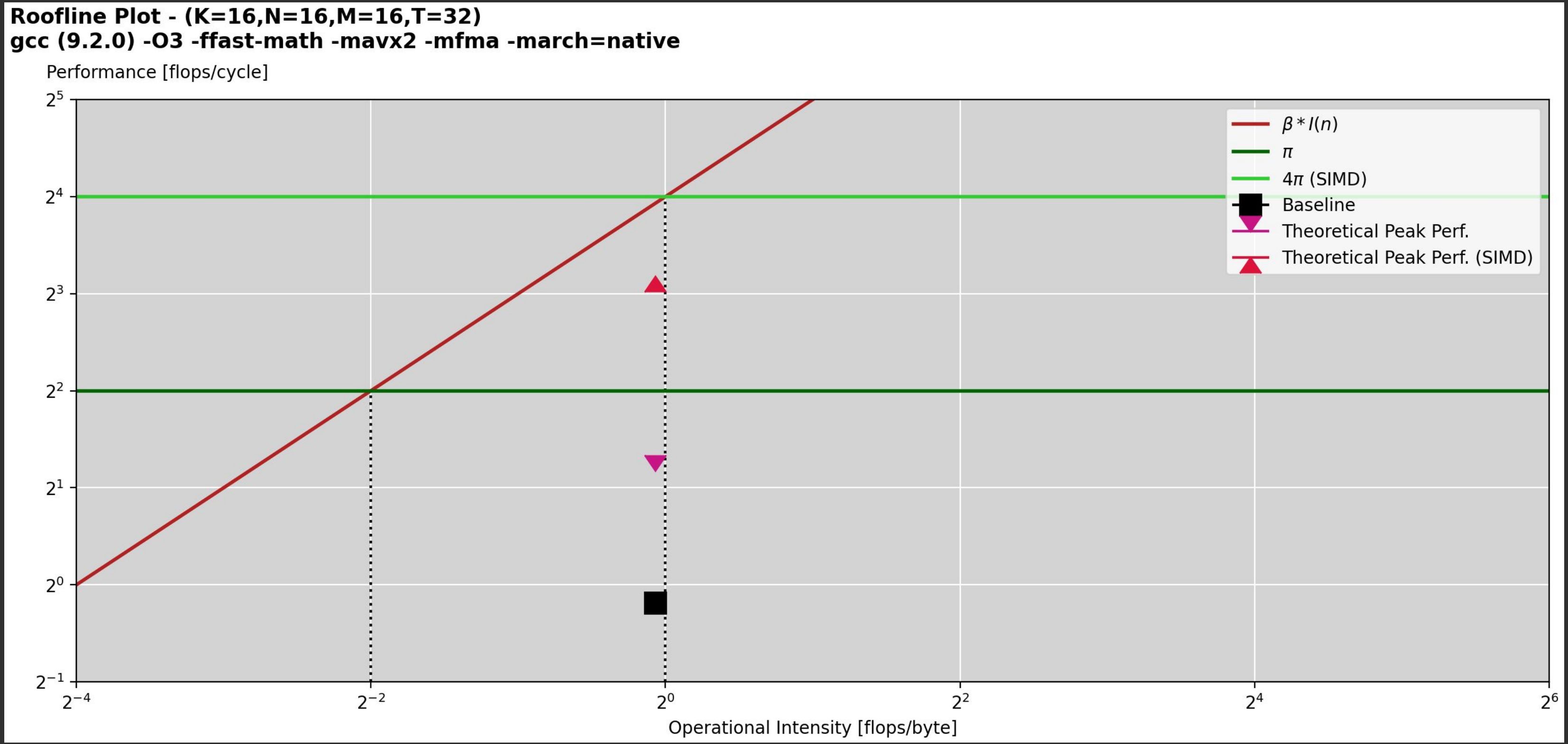

Performance Plot - (T=32, K=16, N=16)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
Baseline
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2^5
2^7
2^9
2^{11}
M

Performance Plot - (T has a step size of 32)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native



Baseline Roofline Plot

Haswell i7-4710MQ, 2.5GHz, 12GB RAM



Optimizations we performed

- Precomputing (taking code out of the loop)
- Reordering of the code (combining loops, transformed some terms)
- Scalar replacement
- Unrolling
- Blocking
- Vectorization
- Using alternative input (transformed a matrix)

Optimizations we performed - reordering

Observation:

- Data dependency (e.g. forwards-step before backwards-step)
- Once all data is computed, dependency can be computed (gamma, sigma)

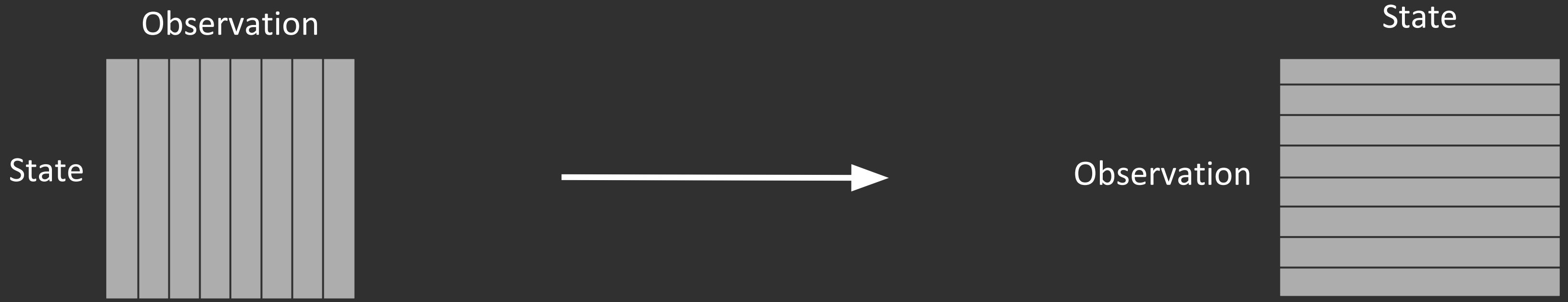
Conclusion: Merging of loops possible

Result:

+ Efficient loop usage
+ Less Division operations
+ Less calls to log

Optimizations we performed - alternative input

Emission Probabilities



An implementation can request the input matrix for the emission probabilities to be transposed. This improves locality.

Optimizations we performed - vectorization

- Application of knowledge gained in exercises
  (e.g. *horizontal addition*)

- Awareness of data dependency during unrolling
  (e.g. *usage of T-1 or T+1 in calculus prevents unrolling T*)

Experiments

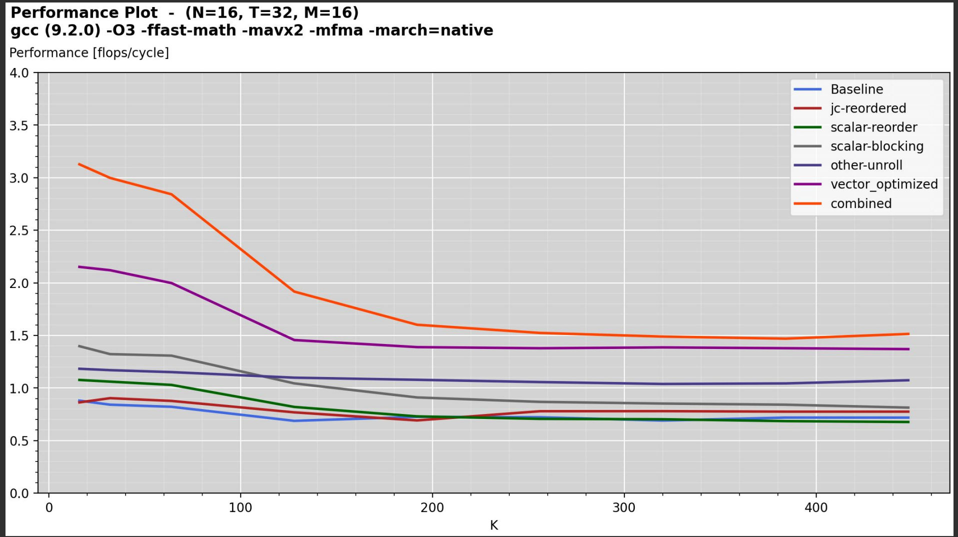
CPU: Haswell i7-4710MQ, 2.5GHz, 12GB RAM

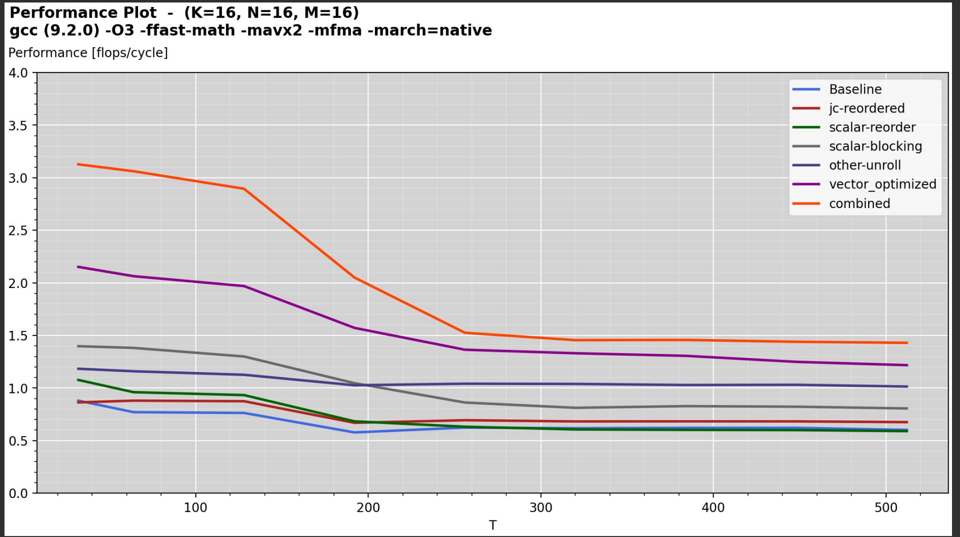
- L1: 32 KB (8-way associative, instruction+data cache)
- L3: 6 MB 12-way set associative shared cache
- AVX2 supported

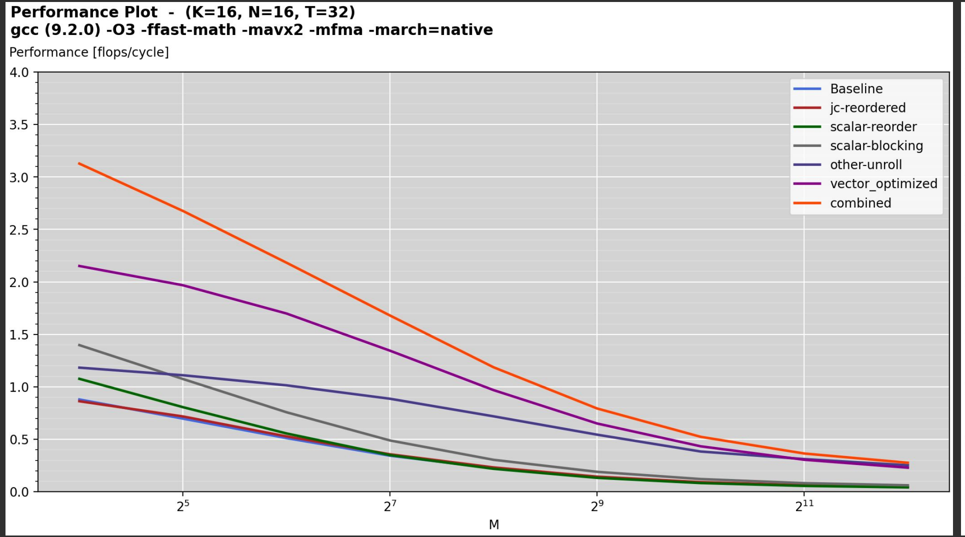
Compiler: gcc 9.2.0 and clang 7.01

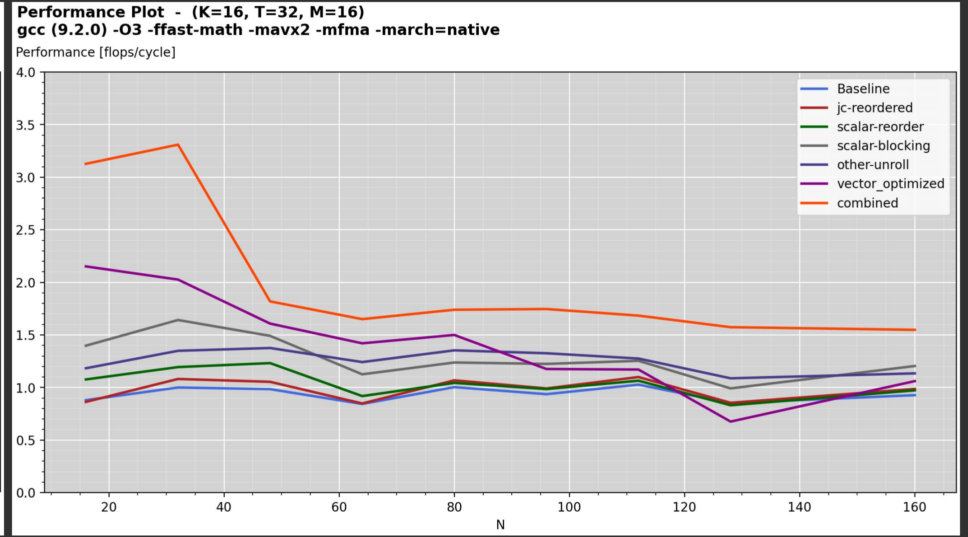
Flags:

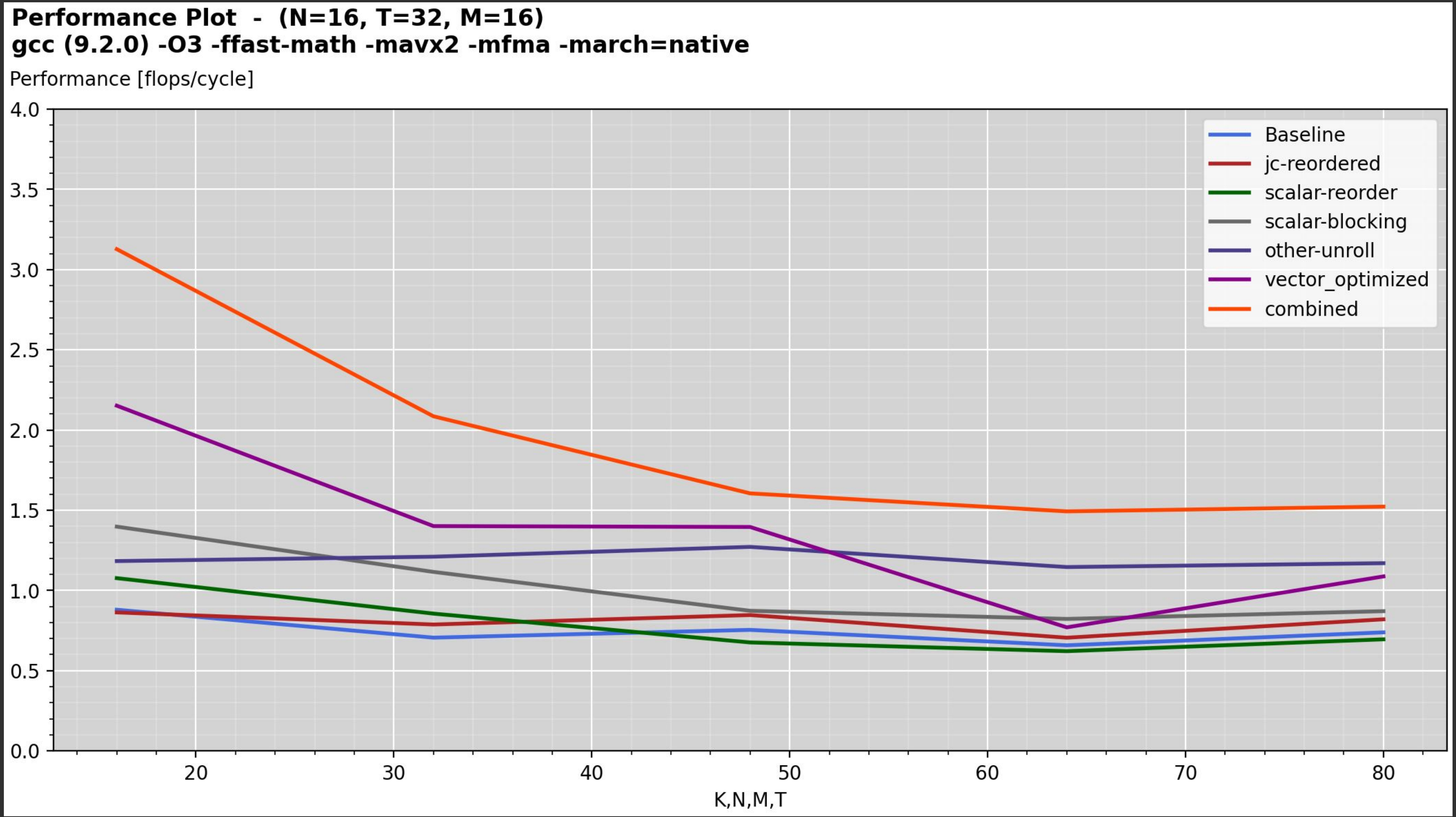
- -O3 -ffast-math -mavx2 -mfma -march=native
- -O3 -fno-vectorize -ffast-math -mavx2 -mfma -march=native
- -O3 -funroll-loops -mavx2 -mfma -march=native


Performance Plot - (N=16, T=32, M=16)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
K
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized
combined


Performance Plot - (K=16, N=16, M=16)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
T
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized
combined

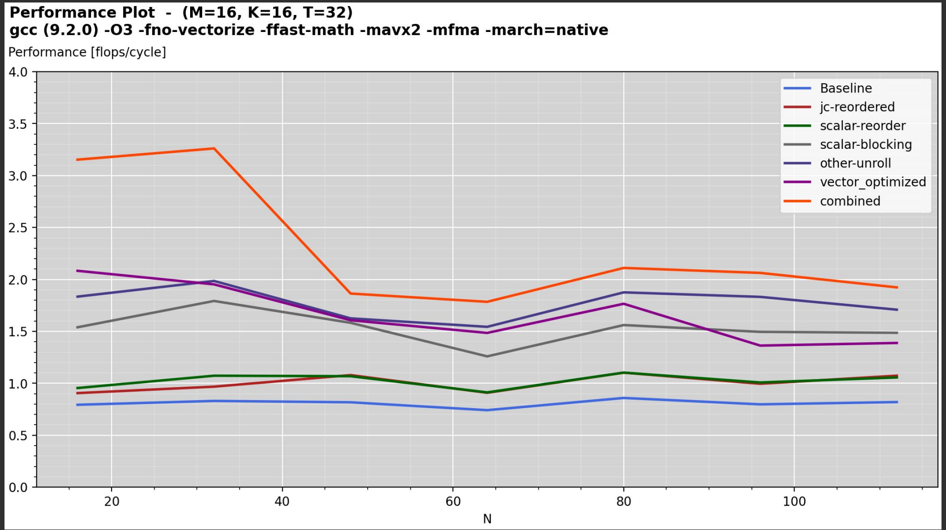

Performance Plot - (K=16, N=16, T=32)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
M
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized
combined

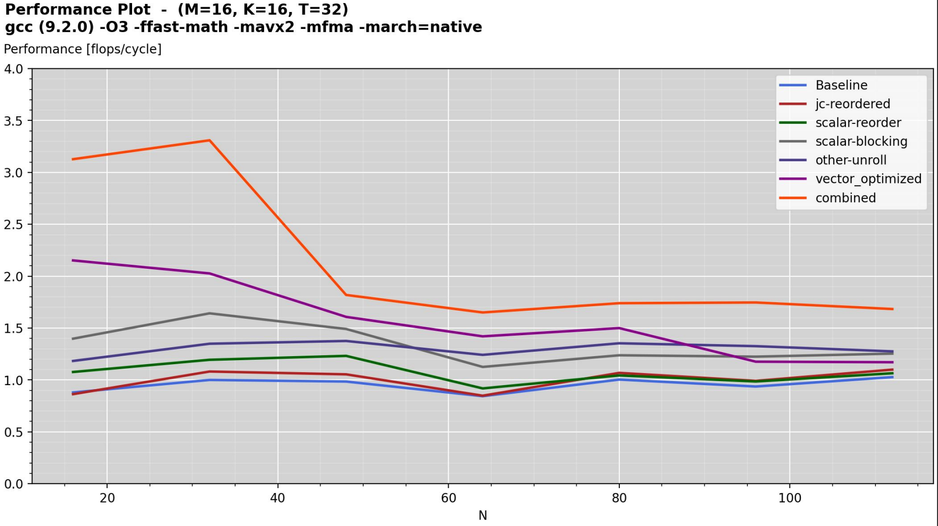

Performance Plot - (K=16, T=32, M=16)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
N
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized
combined

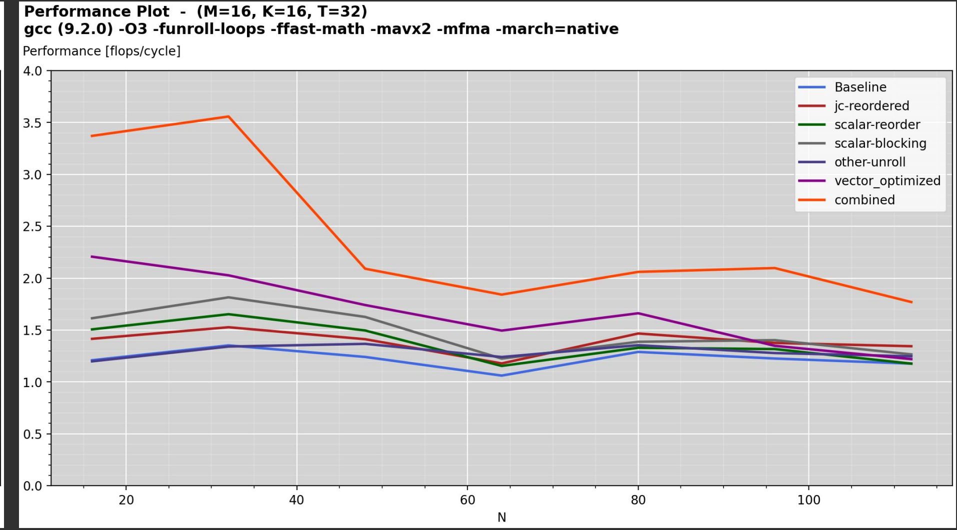


Performance Plot - (N=16, T=32, M=16)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
20
30
40
50
60
70
80
K,N,M,T
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized
combined

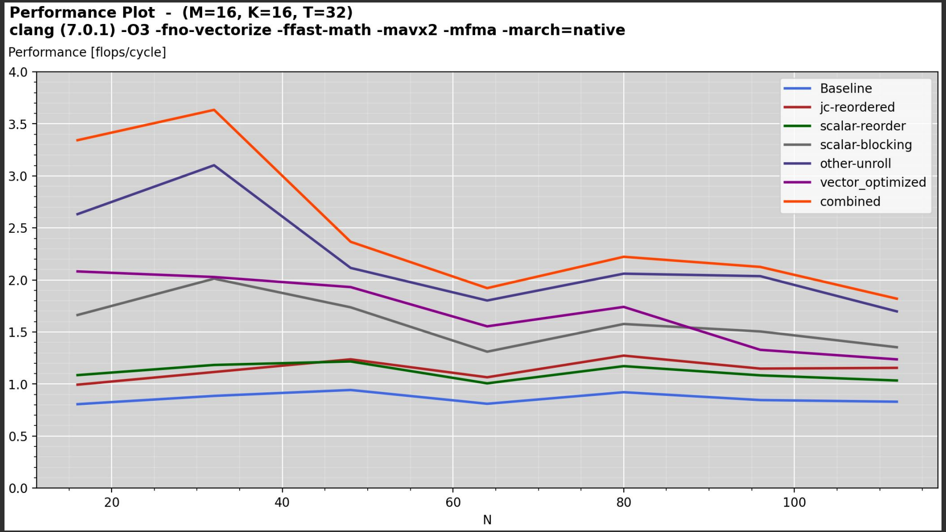
Different Flags (gcc)

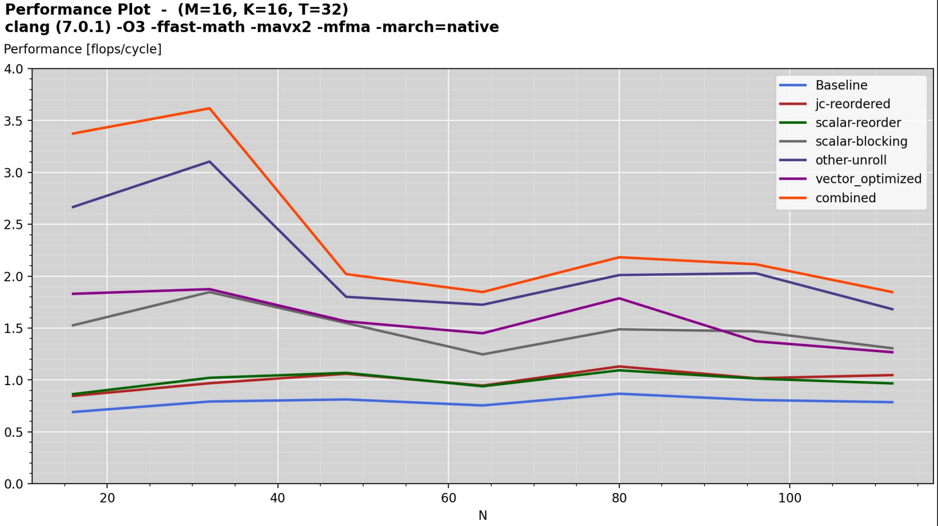


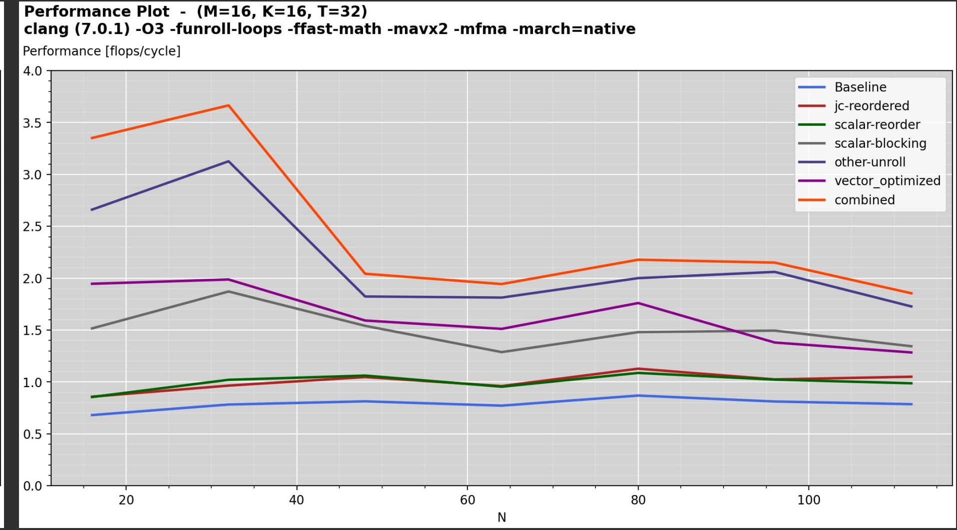


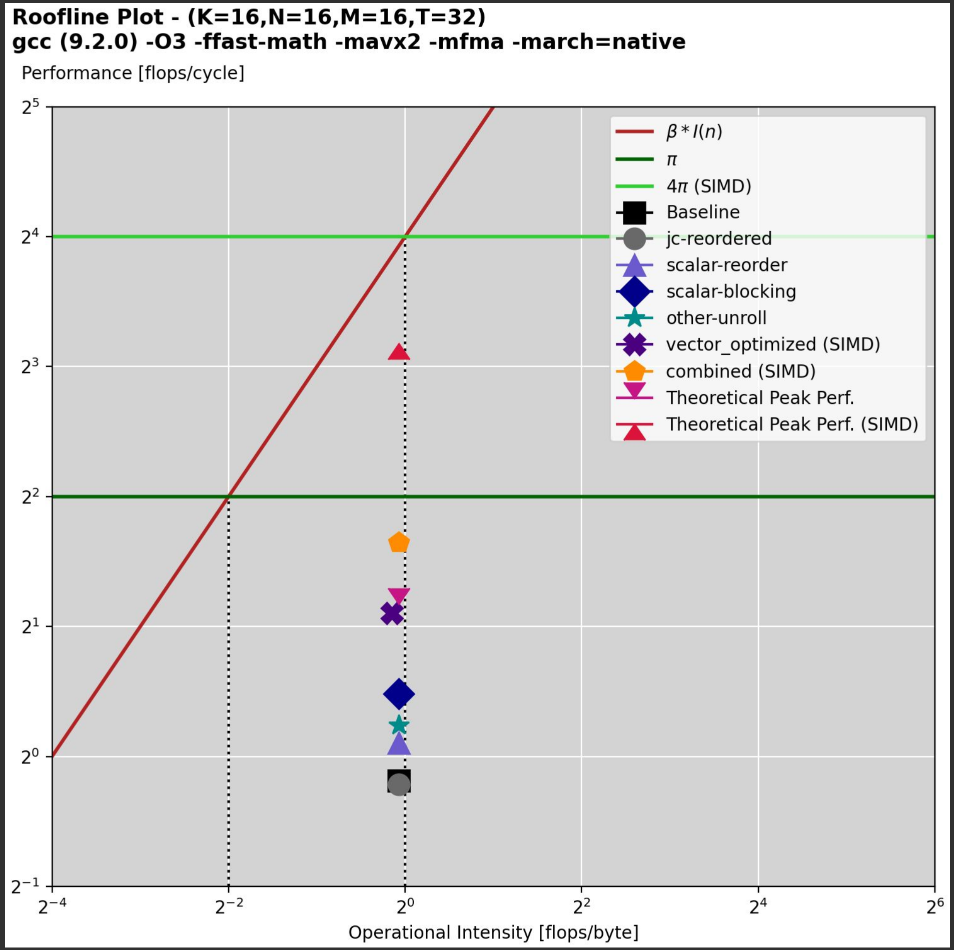


Different Flags (clang)

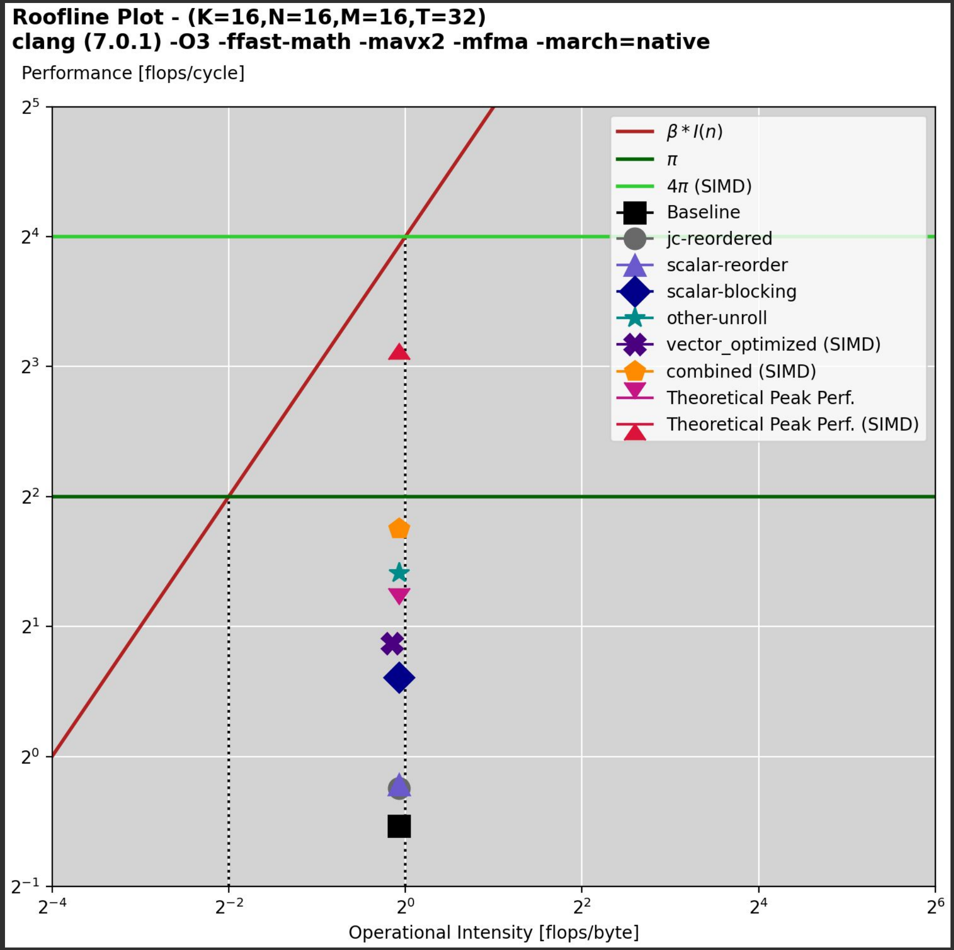




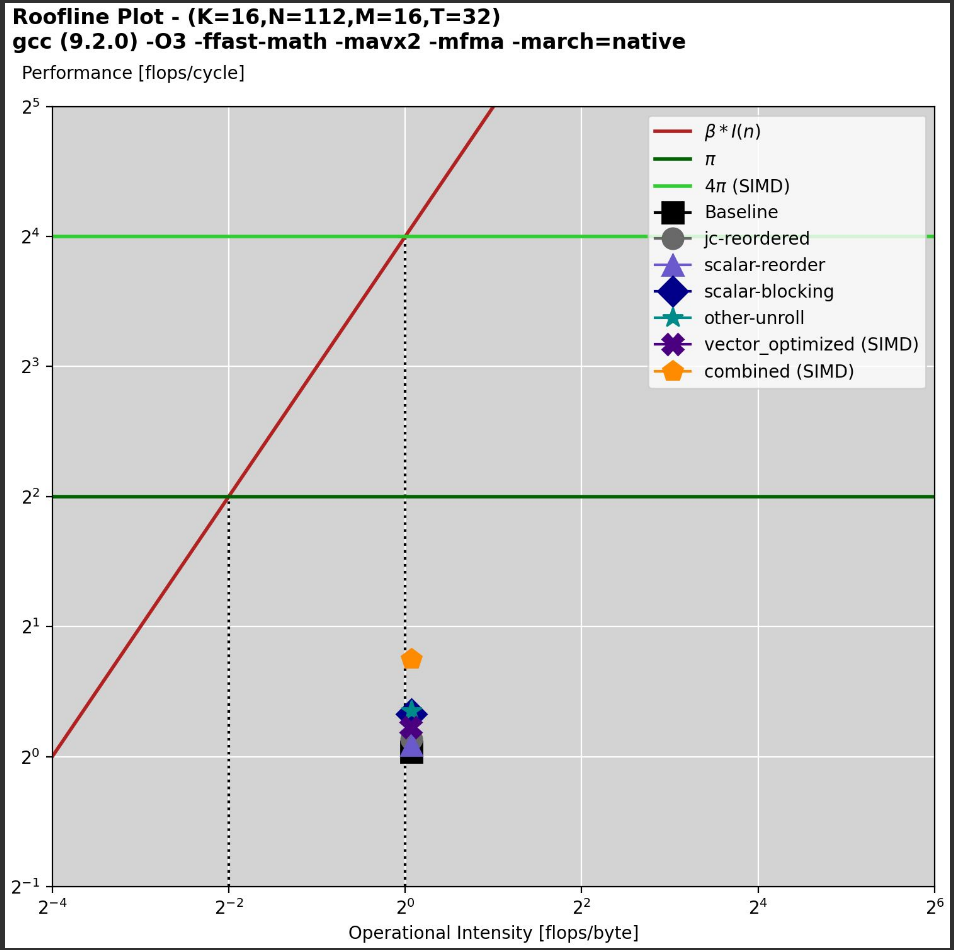


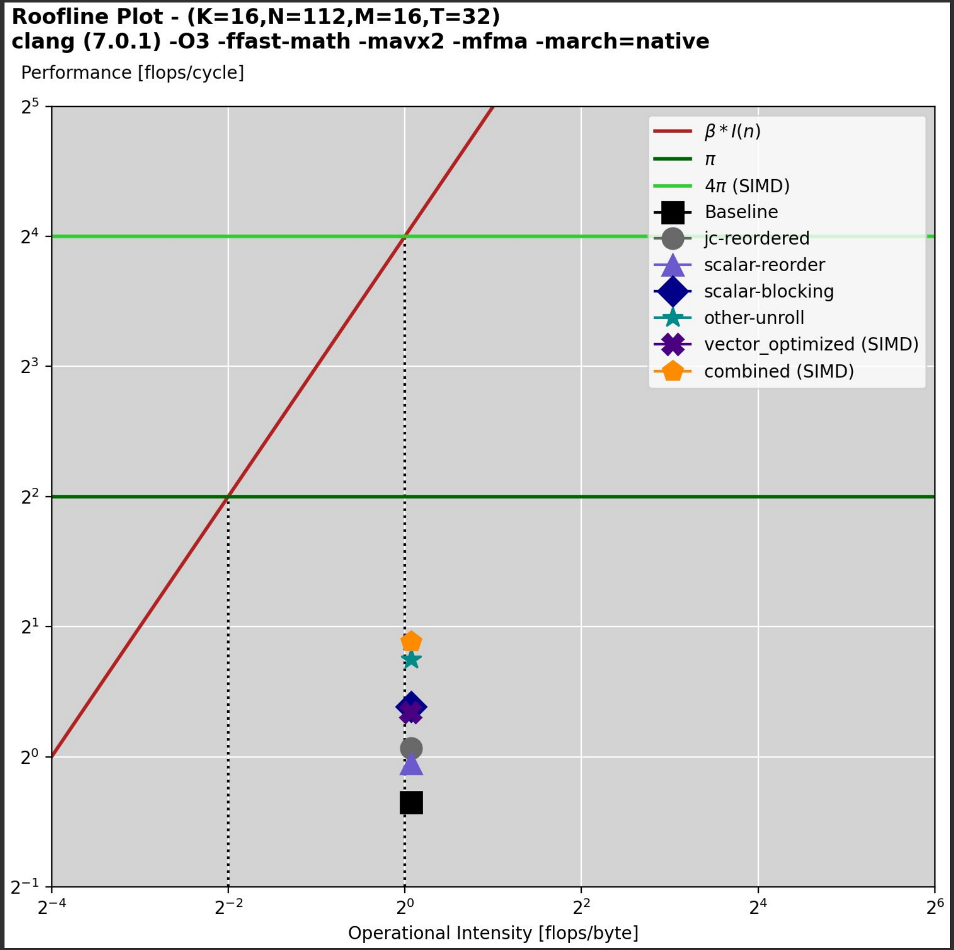


Roofline Plot - (K=16,N=16,M=16,T=32)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
2^5
2^4
2^3
2^2
2^1
2^0
2^{-1}
2^{-4}
2^{-2}
2^0
2^2
2^4
2^6
Operational Intensity [flops/byte]
$\beta * I(n)$
π
4π (SIMD)
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized (SIMD)
combined (SIMD)
Theoretical Peak Perf.
Theoretical Peak Perf. (SIMD)



Roofline Plot - (K=16,N=16,M=16,T=32)
clang (7.0.1) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
2^5
2^4
2^3
2^2
2^1
2^0
2^{-1}
2^{-4}
2^{-2}
2^0
2^2
2^4
2^6
Operational Intensity [flops/byte]
$\beta * I(n)$
π
4π (SIMD)
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized (SIMD)
combined (SIMD)
Theoretical Peak Perf.
Theoretical Peak Perf. (SIMD)


Roofline Plot - (K=16,N=112,M=16,T=32)
gcc (9.2.0) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
$\beta * I(n)$
π
4π (SIMD)
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized (SIMD)
combined (SIMD)
Operational Intensity [flops/byte]


Roofline Plot - (K=16,N=112,M=16,T=32)
clang (7.0.1) -O3 -ffast-math -mavx2 -mfma -march=native
Performance [flops/cycle]
$\beta * I(n)$
π
4π (SIMD)
Baseline
jc-reordered
scalar-reorder
scalar-blocking
other-unroll
vector_optimized (SIMD)
combined (SIMD)
Operational Intensity [flops/byte]

Conclusion

- Combination makes the difference.
- Dependant on the compiler and optimization flags, optimizations can differ or even worsen the performance
- Plots as analysation for further optimizations