

Non-Negative Matrix Factorization

Daniele Coppola

Viktor Gsteiger

Maša Nešić

Matteo Oldani



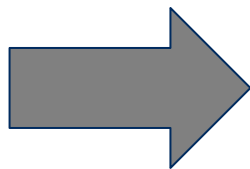
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

The Algorithm

$$V = W \times H$$

$$H_{[i,j]}^{n+1} = H_{[i,j]}^n \cdot \frac{((W^n)^T V)_{[i,j]}}{((W^n)^T W^n H^n)_{[i,j]}}$$

$$W_{[i,j]}^{n+1} = W_{[i,j]}^n \cdot \frac{(V (H^{n+1})^T)_{[i,j]}}{(W^n H^{n+1} (H^{n+1})^T)_{[i,j]}}$$



Cost analysis

- Asymptotic runtime complexity :

$$O(m \cdot n \cdot r + m \cdot r^2 + n \cdot r^2 + n^2 \cdot r + m \cdot n)$$

$$V : m \times n$$

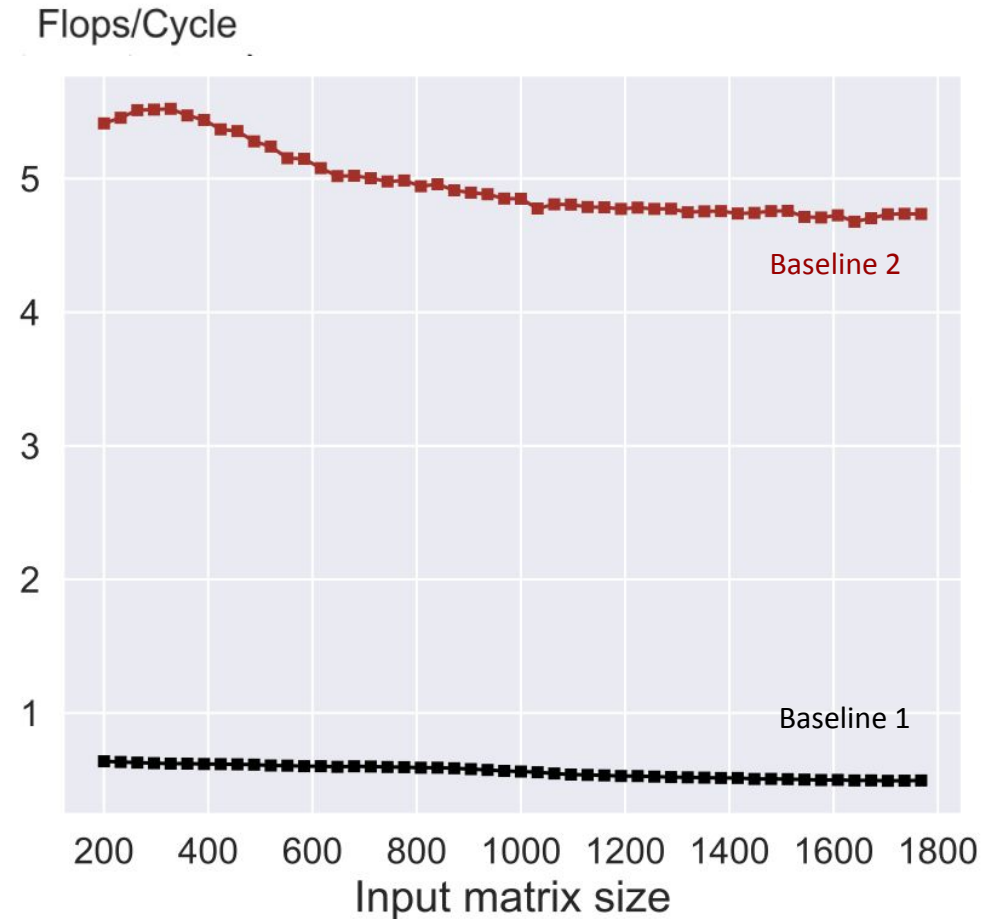
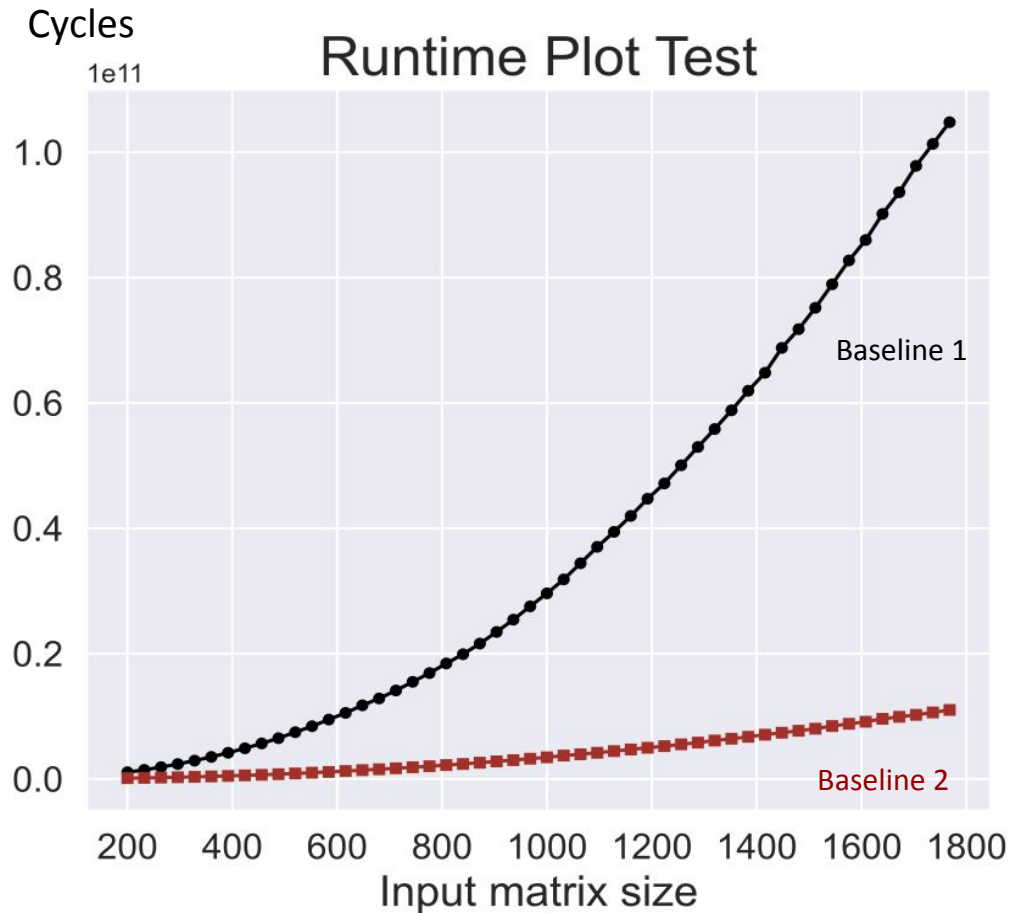
$$W : m \times r$$

$$H : r \times n$$

- Cost - measured by counting operations inside the code:

$$\begin{aligned} &2 * m * n * r + 5 * m * n + 3 + \\ &\text{number_of_iterations} * (6 * m * n * r + \\ &\quad 3 * m * r * r + \\ &\quad 3 * n * r * r + \\ &\quad 5 * m * n + \\ &\quad 2 * m * r + \\ &\quad 2 * n * r + 3) \end{aligned}$$

Baseline Implementations

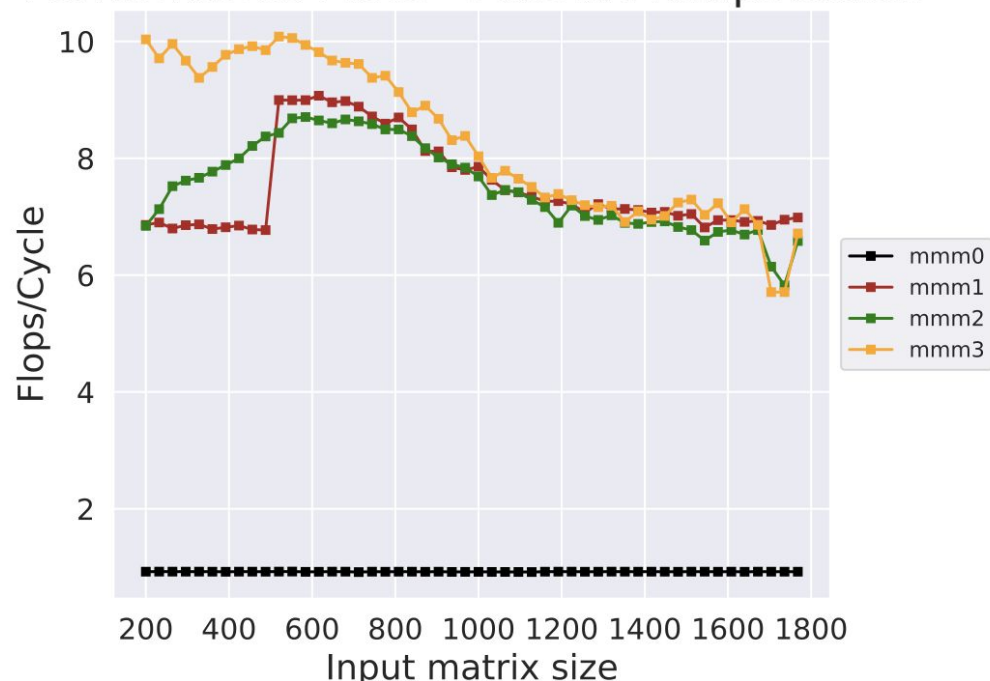


NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

Matrix Multiplication Optimizations

- Blocking (block size = 16)
- Loop unrolling to have 8 independent lines of computations
- Vectorization of the computation

Performance Plots - Matrix Multiplication



NNMF (double precision) on i5-6600K, 3.5 GHz, r = 16

```
int idx_b = k * B_n_col + jj;
for (kk = k; kk < k + nB; kk++) {
    a0 = _mm256_set1_pd(A[Aii + kk]);
    a1 = _mm256_set1_pd(A[Aii + A_n_col + kk]);

    b0 = _mm256_load_pd((double *) &B[idx_b]);
    b1 = _mm256_load_pd((double *) &B[idx_b + 4]);
    b2 = _mm256_load_pd((double *) &B[idx_b + 8]);
    b3 = _mm256_load_pd((double *) &B[idx_b + 12]);

    r0 = _mm256_fmadd_pd(a0, b0, r0);
    r1 = _mm256_fmadd_pd(a0, b1, r1);
    r2 = _mm256_fmadd_pd(a0, b2, r2);
    r3 = _mm256_fmadd_pd(a0, b3, r3);

    r4 = _mm256_fmadd_pd(a1, b0, r4);
    r5 = _mm256_fmadd_pd(a1, b1, r5);
    r6 = _mm256_fmadd_pd(a1, b2, r6);
    r7 = _mm256_fmadd_pd(a1, b3, r7);

    idx_b += B_n_col;
}
```

Matrix Padding

- **R multiple of block size**

- Outperforming BLAS

- **R not multiple of block size**

- Blas outperforming us



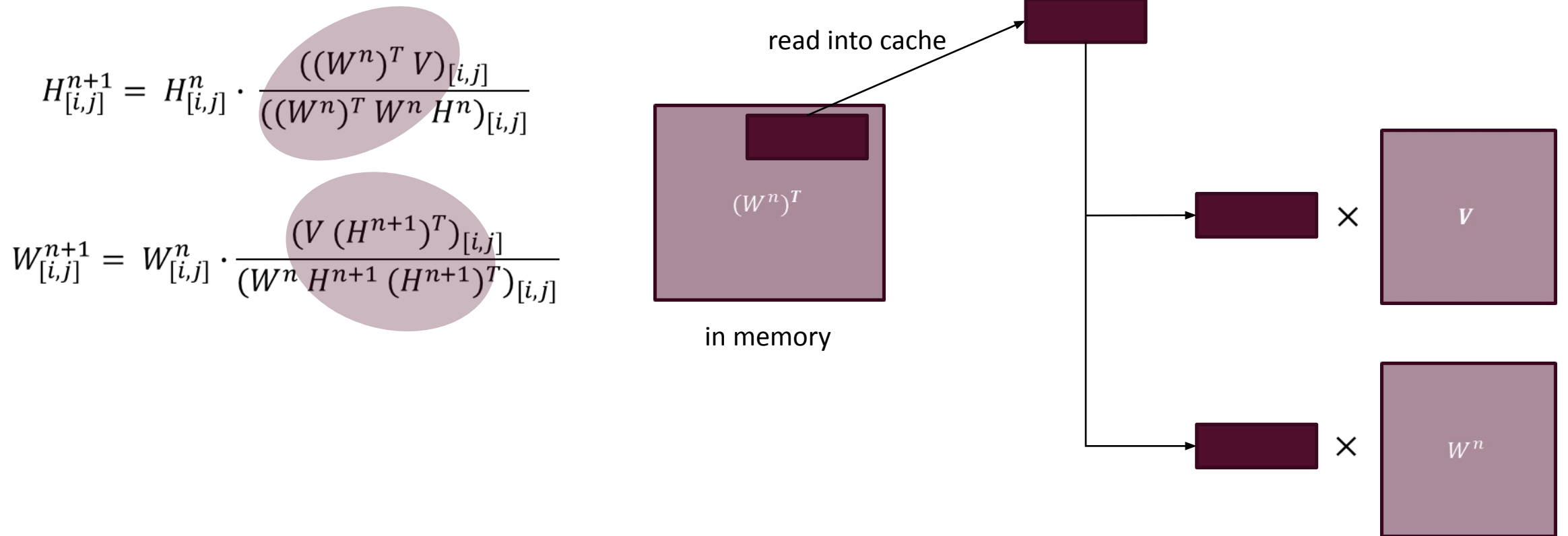
- **Matrix padding to the block size**

- More computation but better microarchitecture usage
- Outperforming BLAS
- Padding with 0s



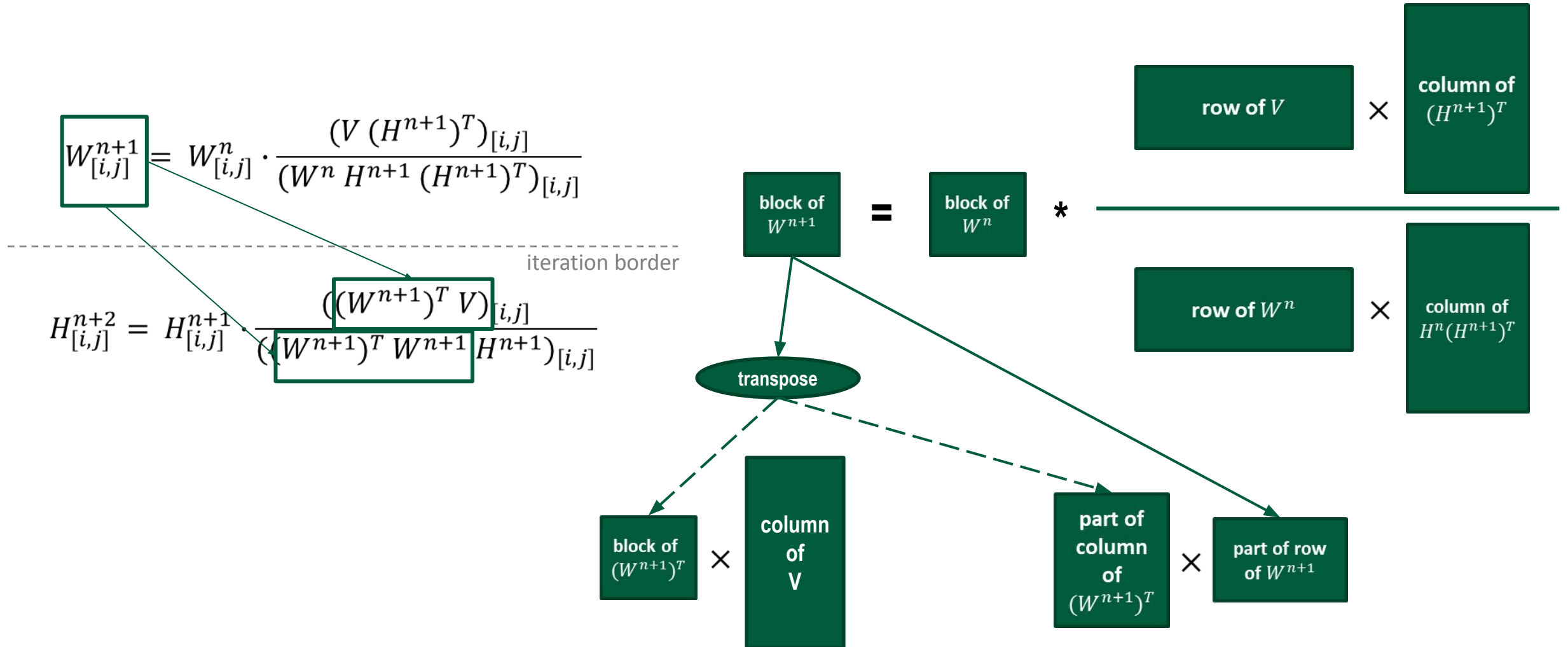
Algorithmic optimization 1 – Reuse entries of W

- Reduces the number of read accesses to $(W^n)^T$ in the computation of H^{n+1} , and to $(H^{n+1})^T$ in the computation of W^{n+1} by half



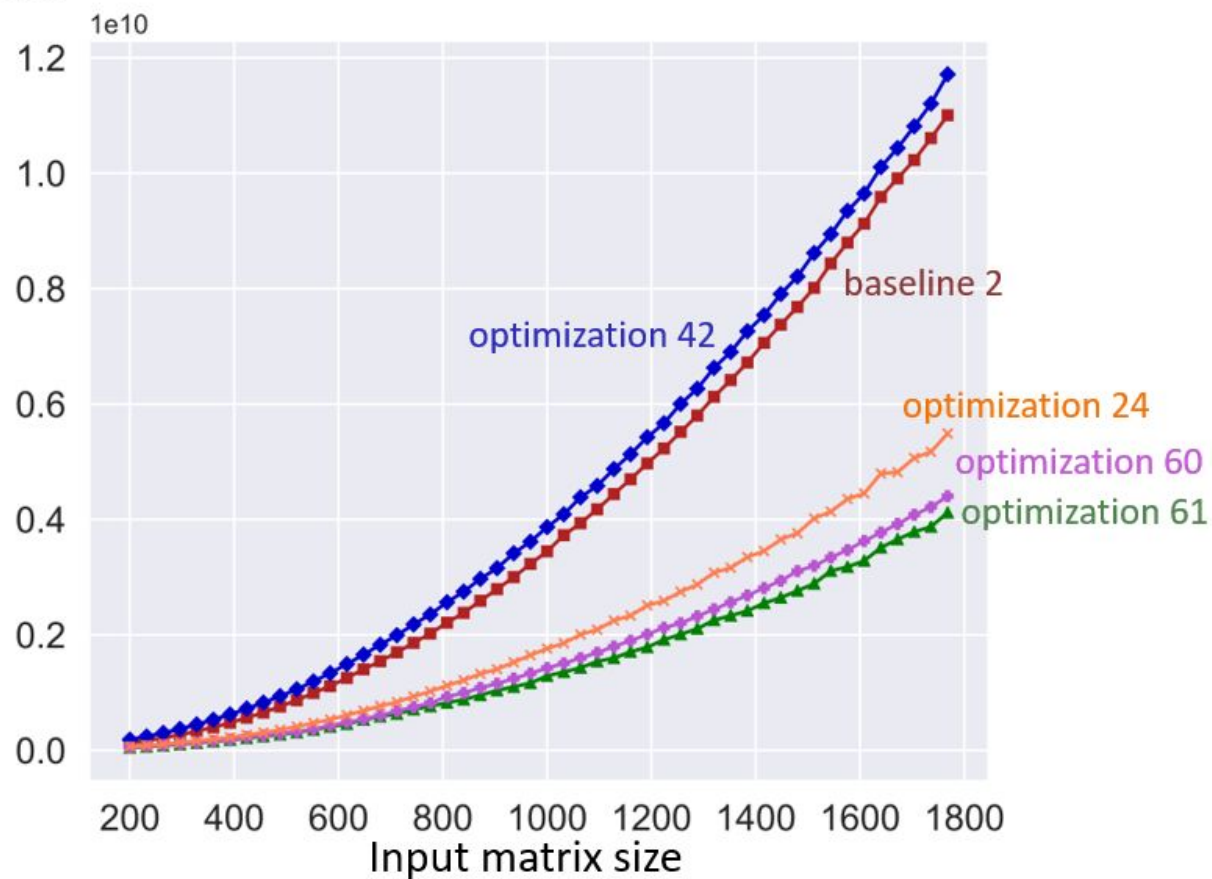
Algorithmic optimization 4 – Reuse block of W across 2 iterations

- The calculated block of W^{n+1} is immediately used in the calculation of $(W^{n+1})^T V$ and $(W^{n+1})^T W^{n+1}$

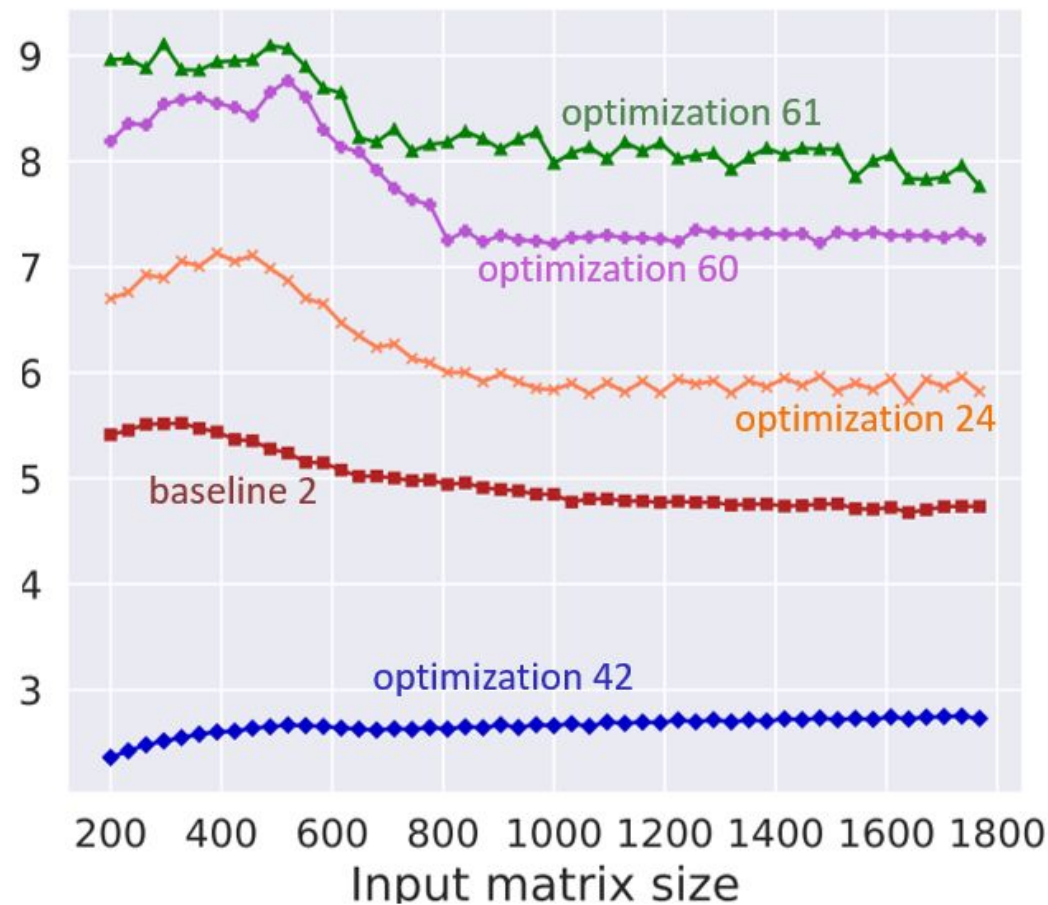


Runtime and Performance Plots

Cycles

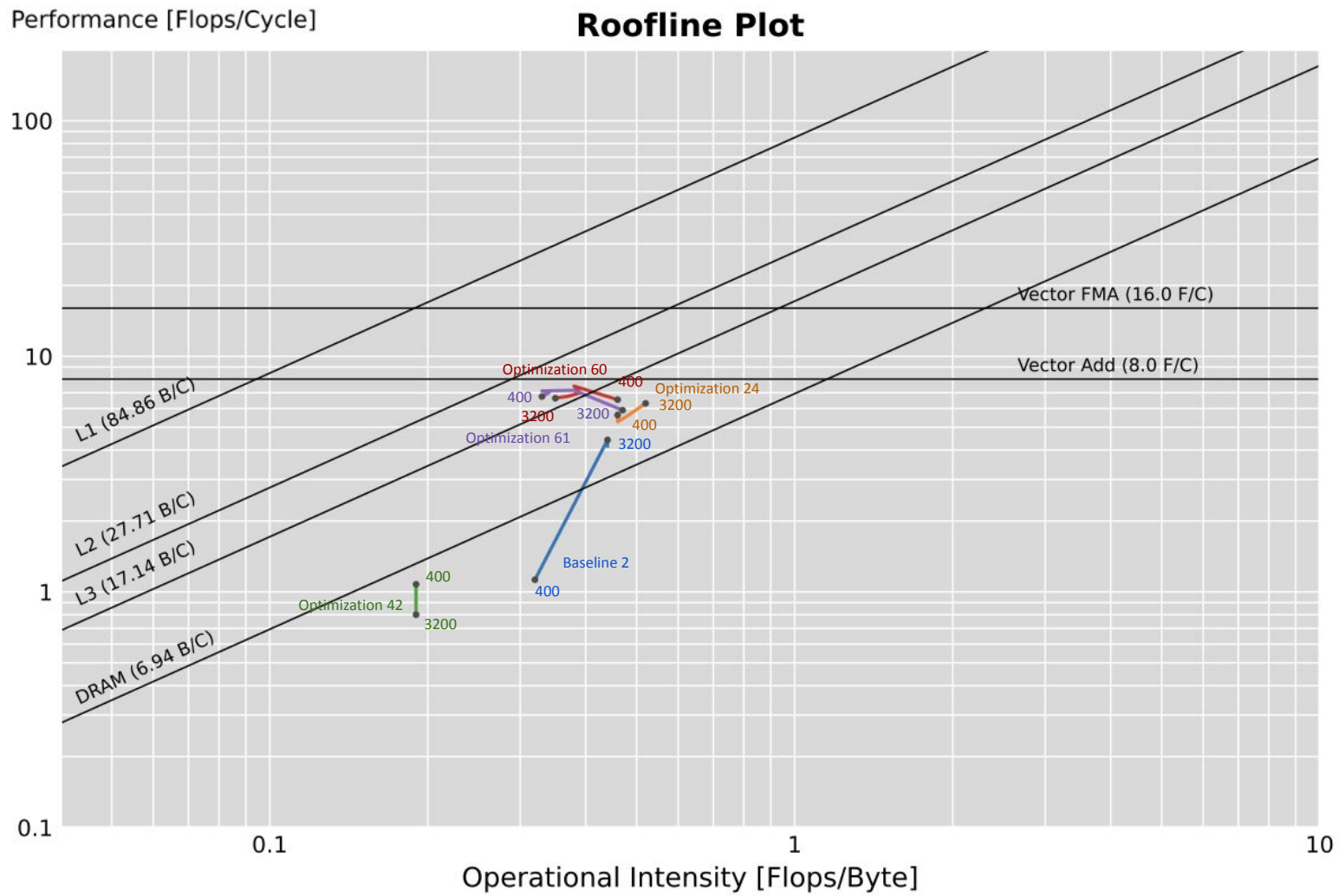


Flops/cycle



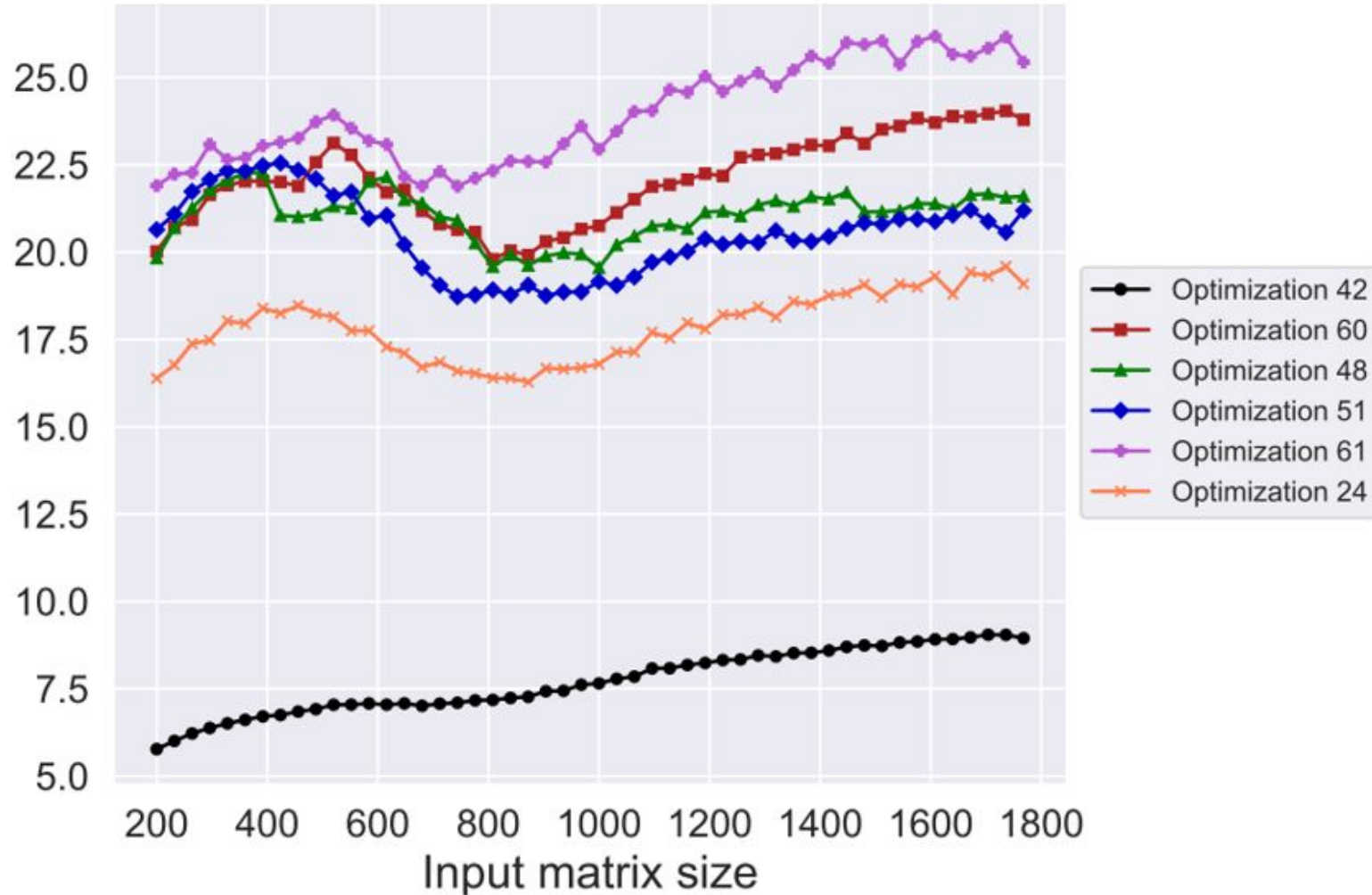
NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

Roofline Plot



NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

Speedup

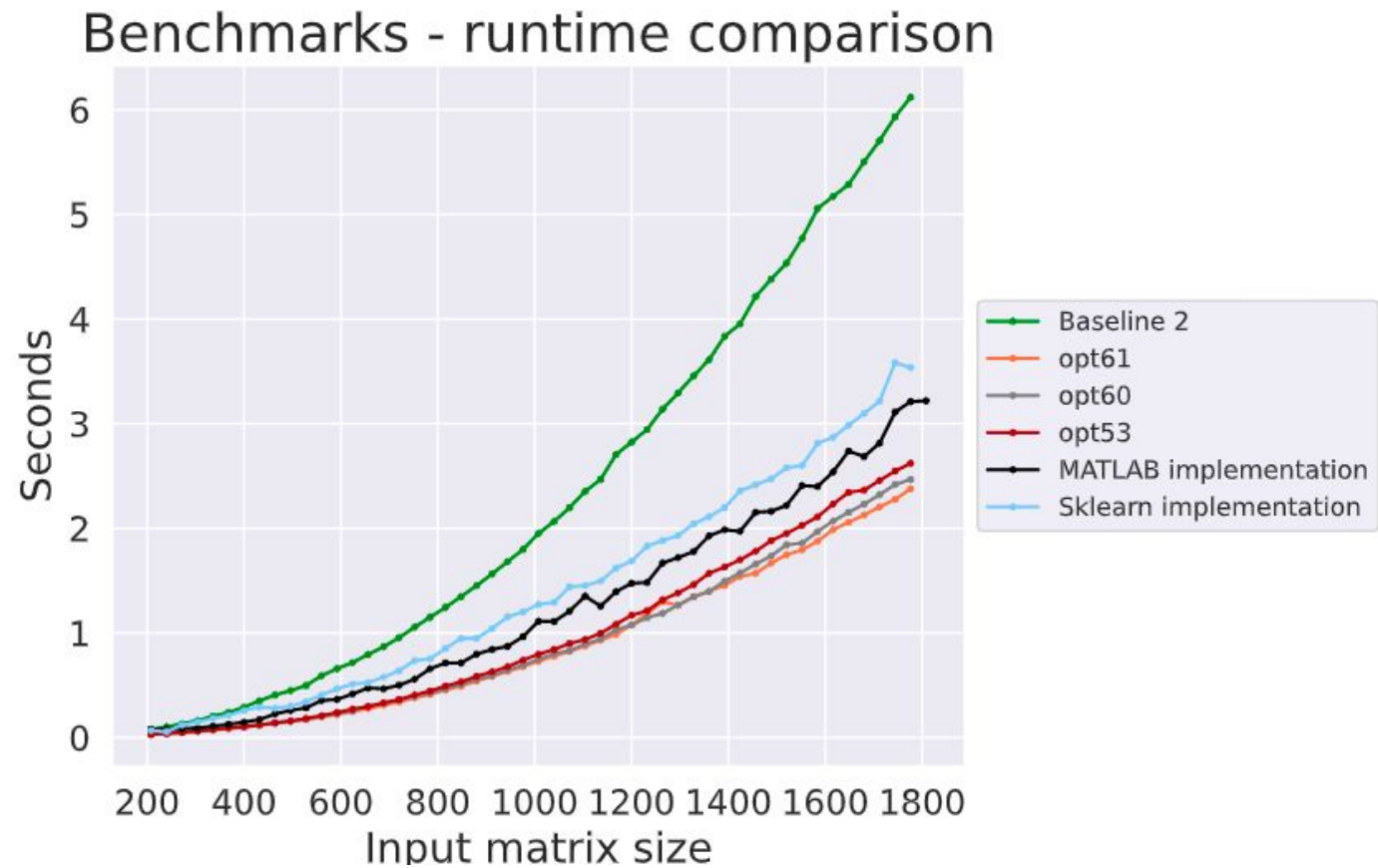


NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

■ Average speedup:

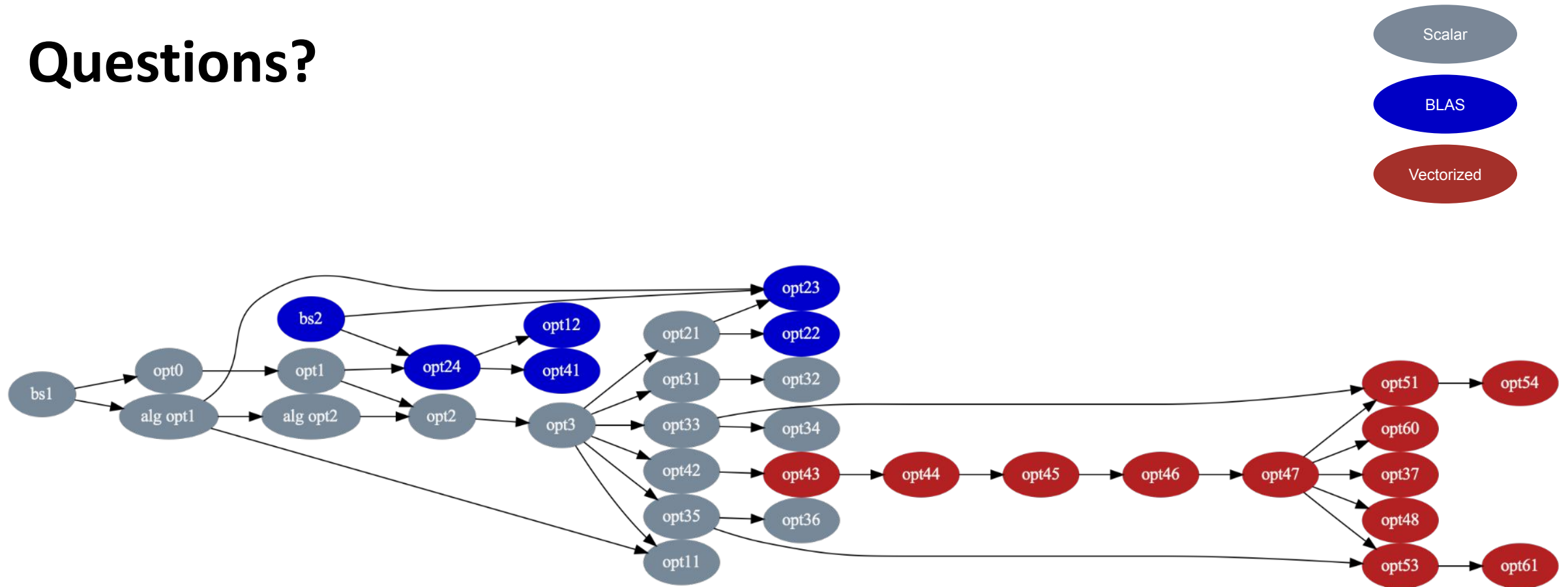
- **Optimization 61 - 23.95**
- Optimization 60 - 22.07
- Optimization 48 - 21.05
- Optimization 51 - 20.49
- Optimization 54 - 19.24
- Optimization 24 - 17.84
- Optimization 42 - 7.71

Benchmarking plot



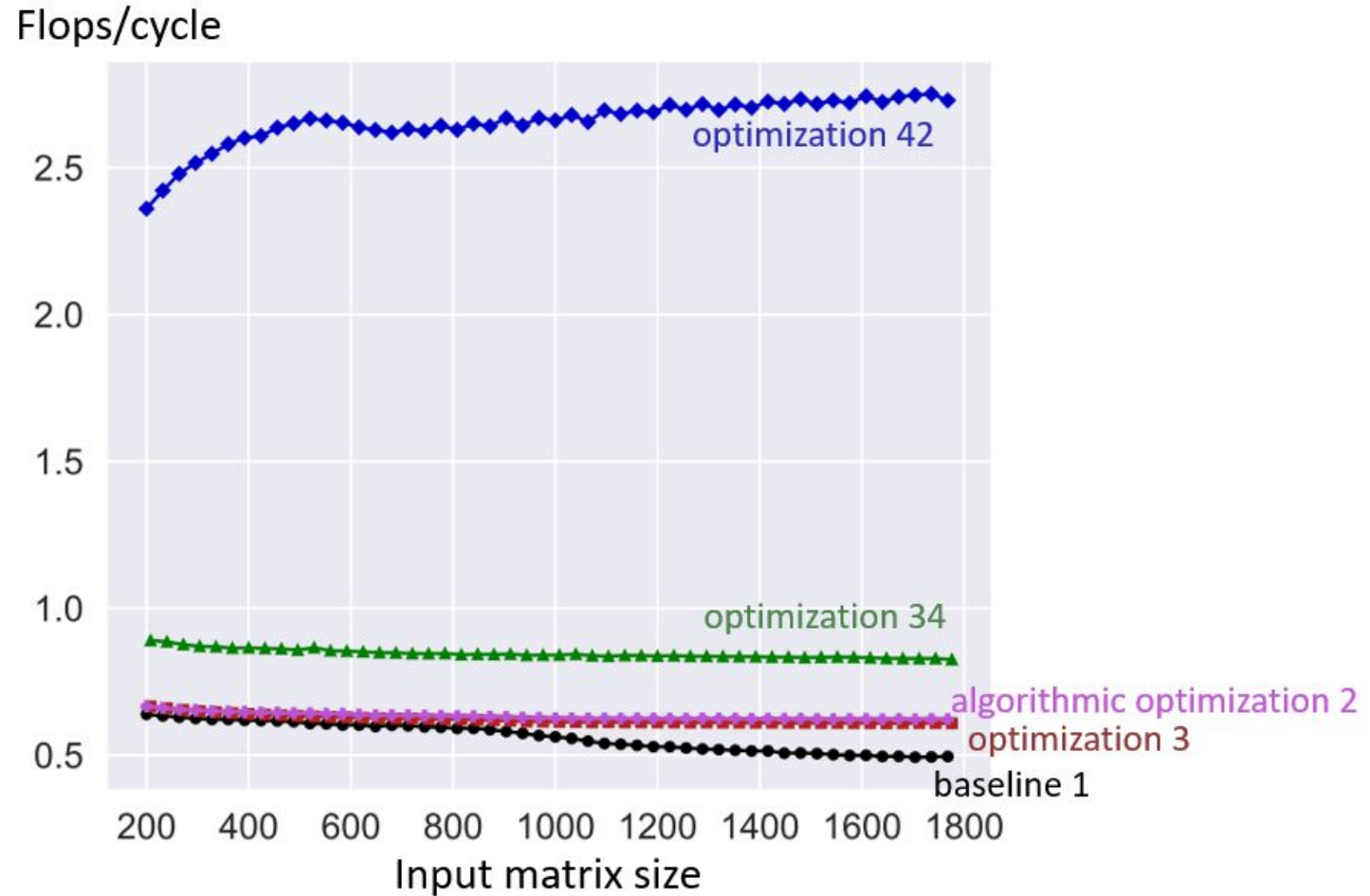
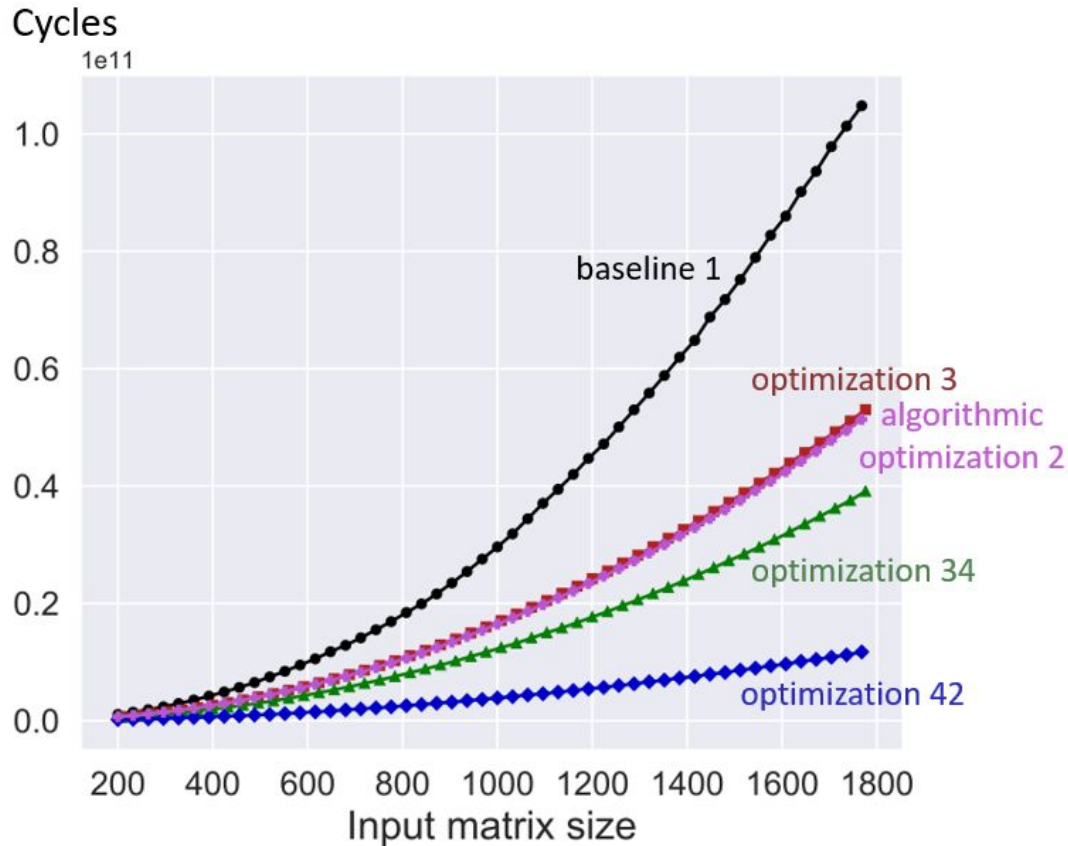
NNMF (double precision) on i7-8565U, 2 GHz, $r = 16$

Questions?



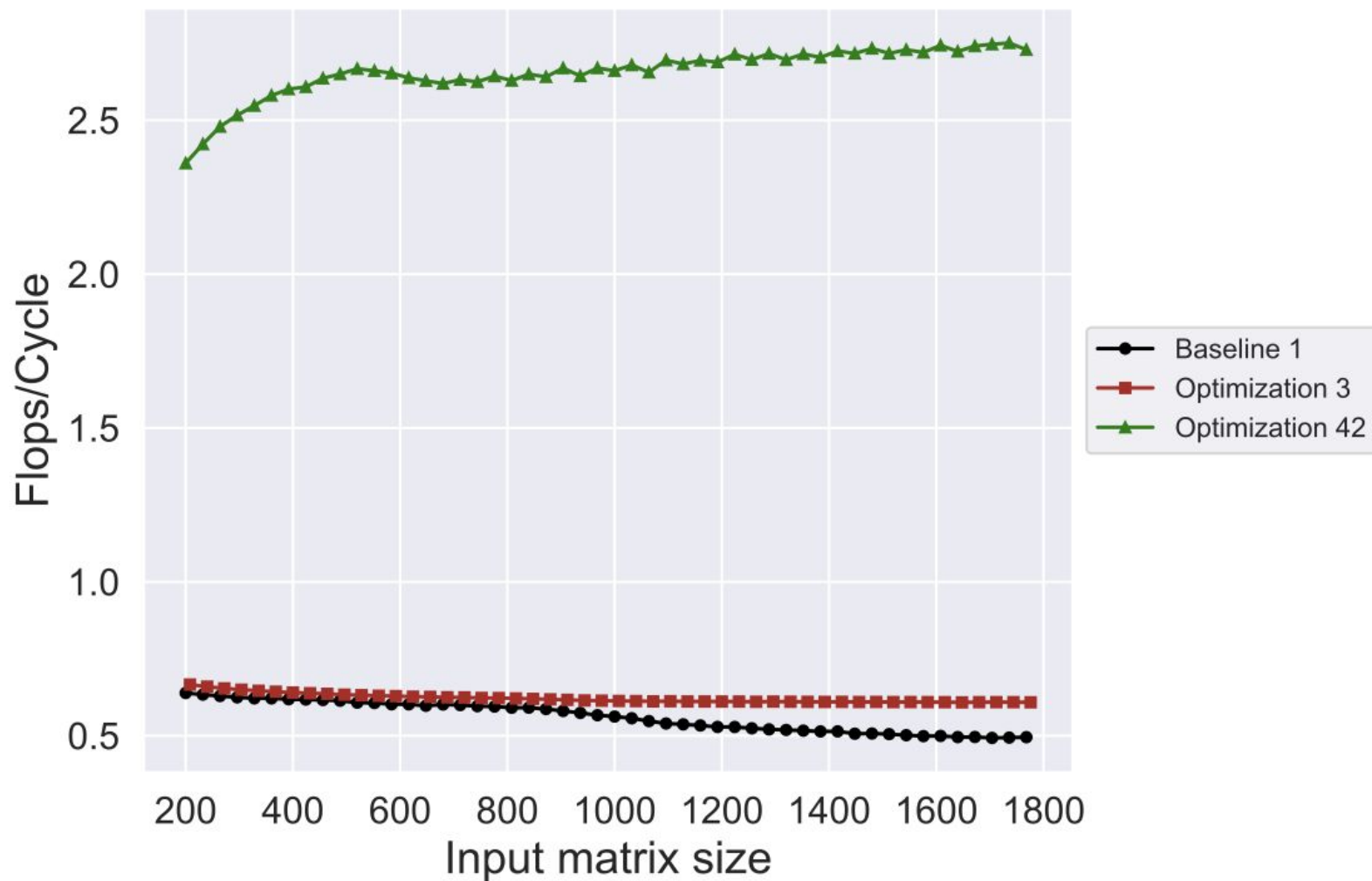
Appendix

Runtime and Performance Plots - scalar

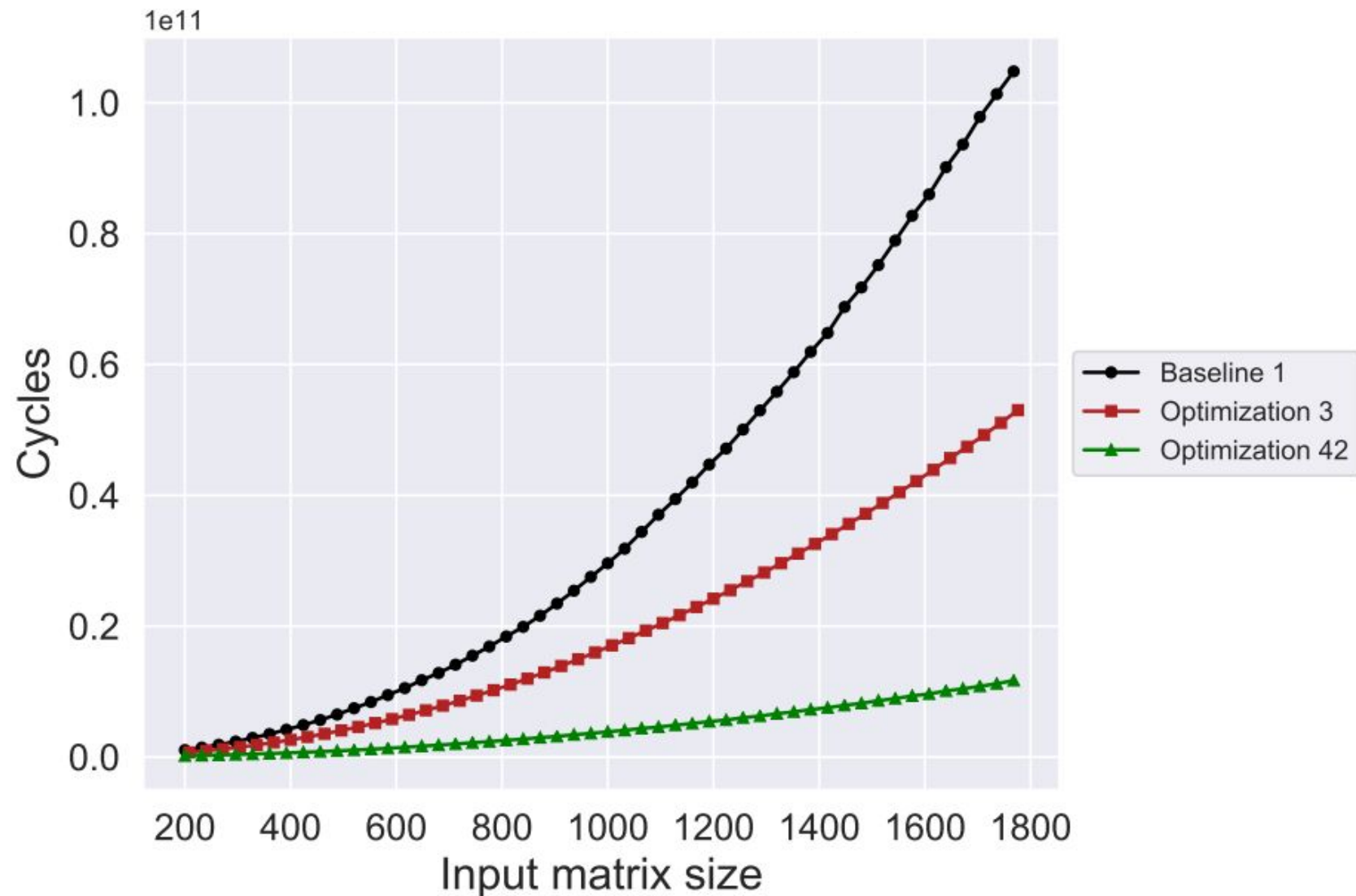


NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

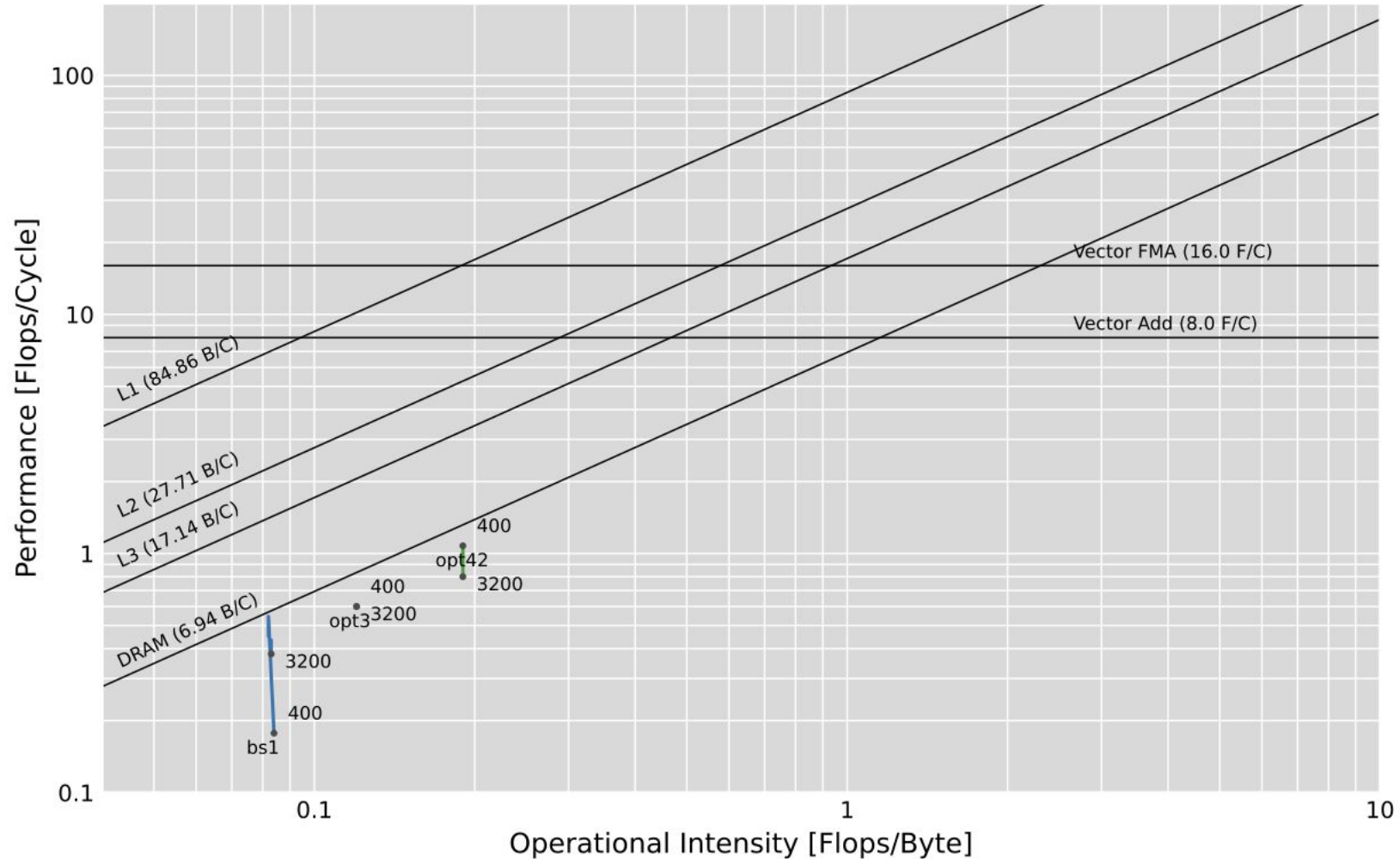
Performance Scalar Optimizations



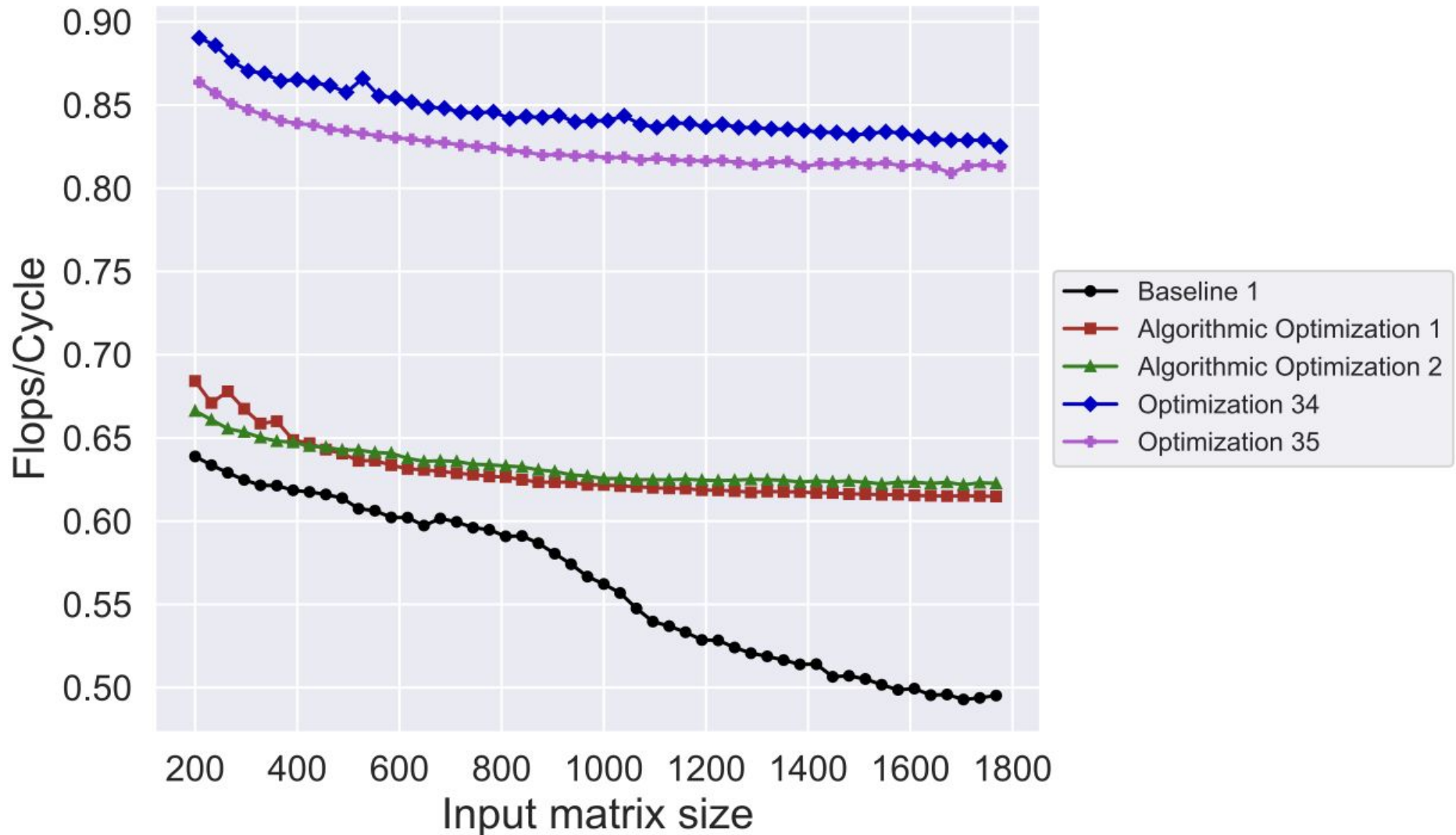
Runtime Scalar Optimizations



Roofline Scalar Optimizations

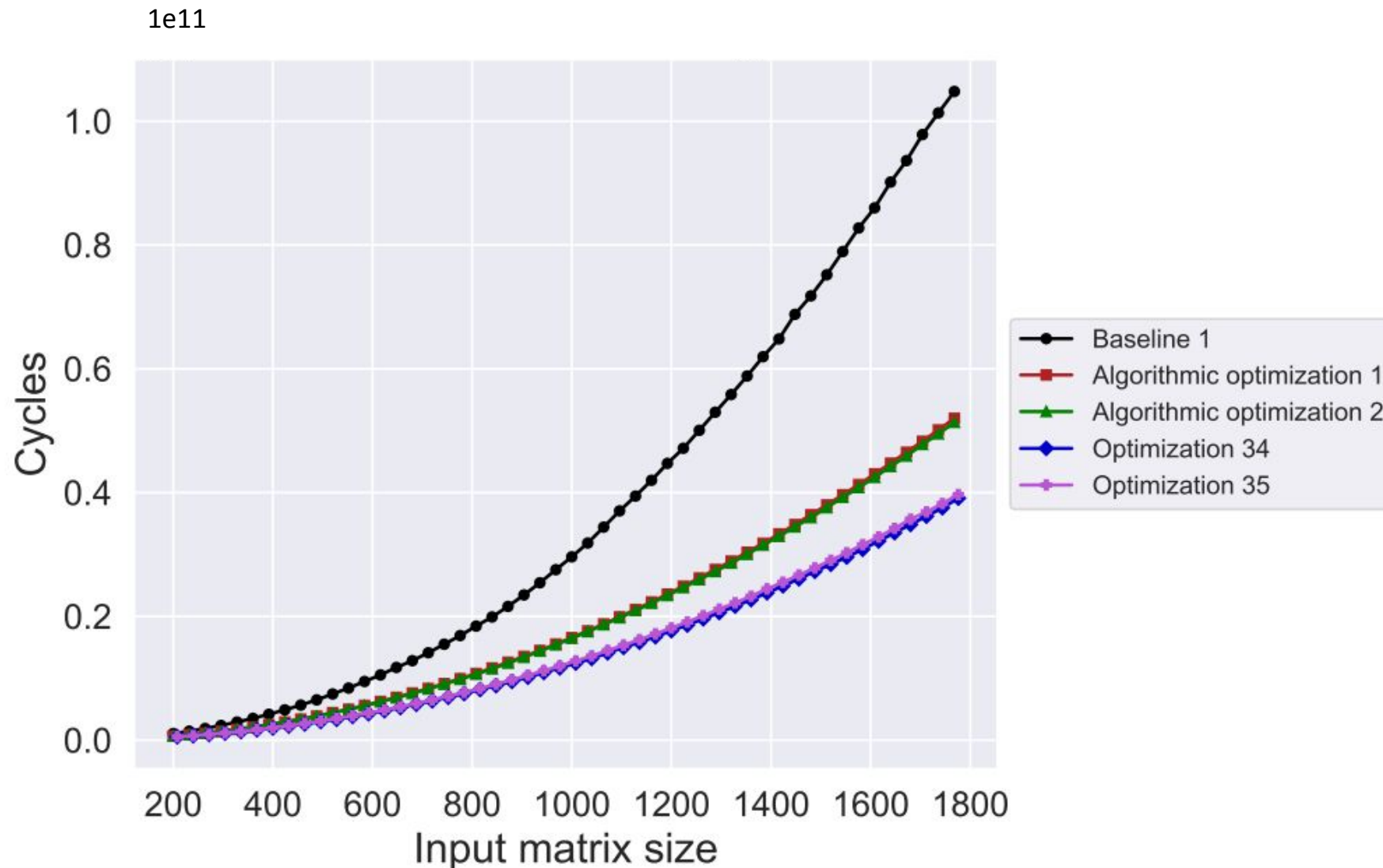


Performance Scalar Algorithmic Optimizations



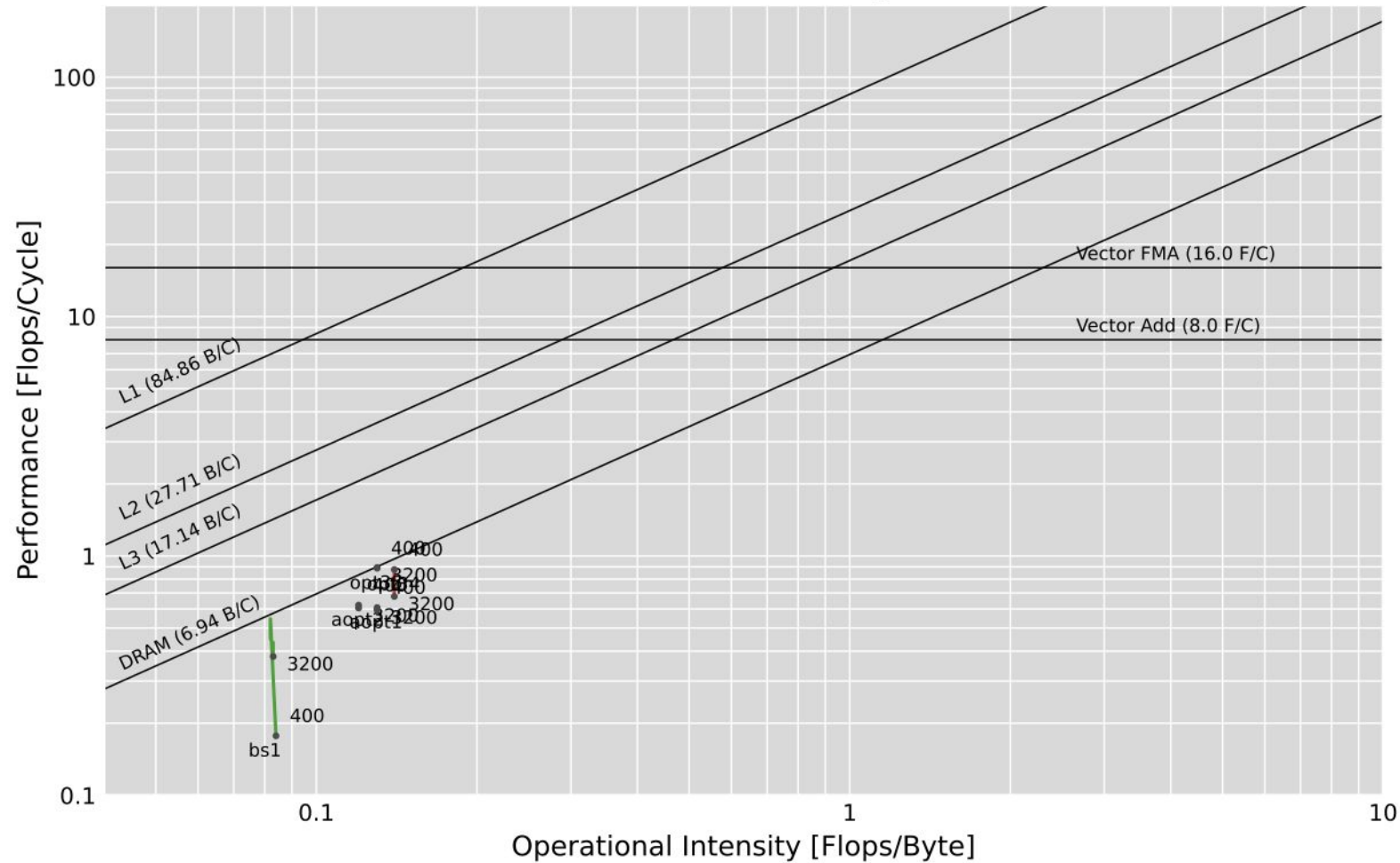
NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

Runtime Scalar Algorithmic Optimizations

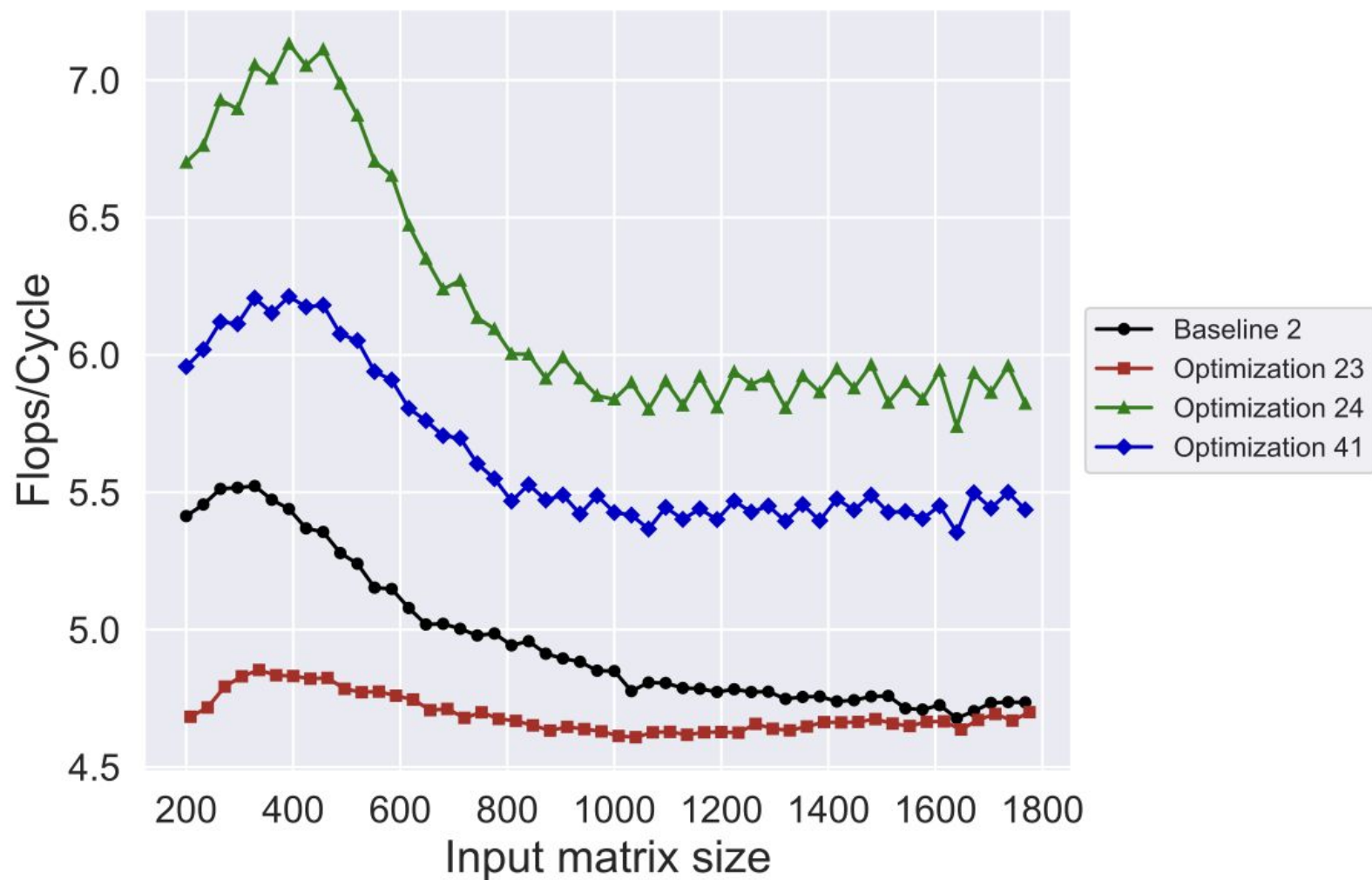


NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

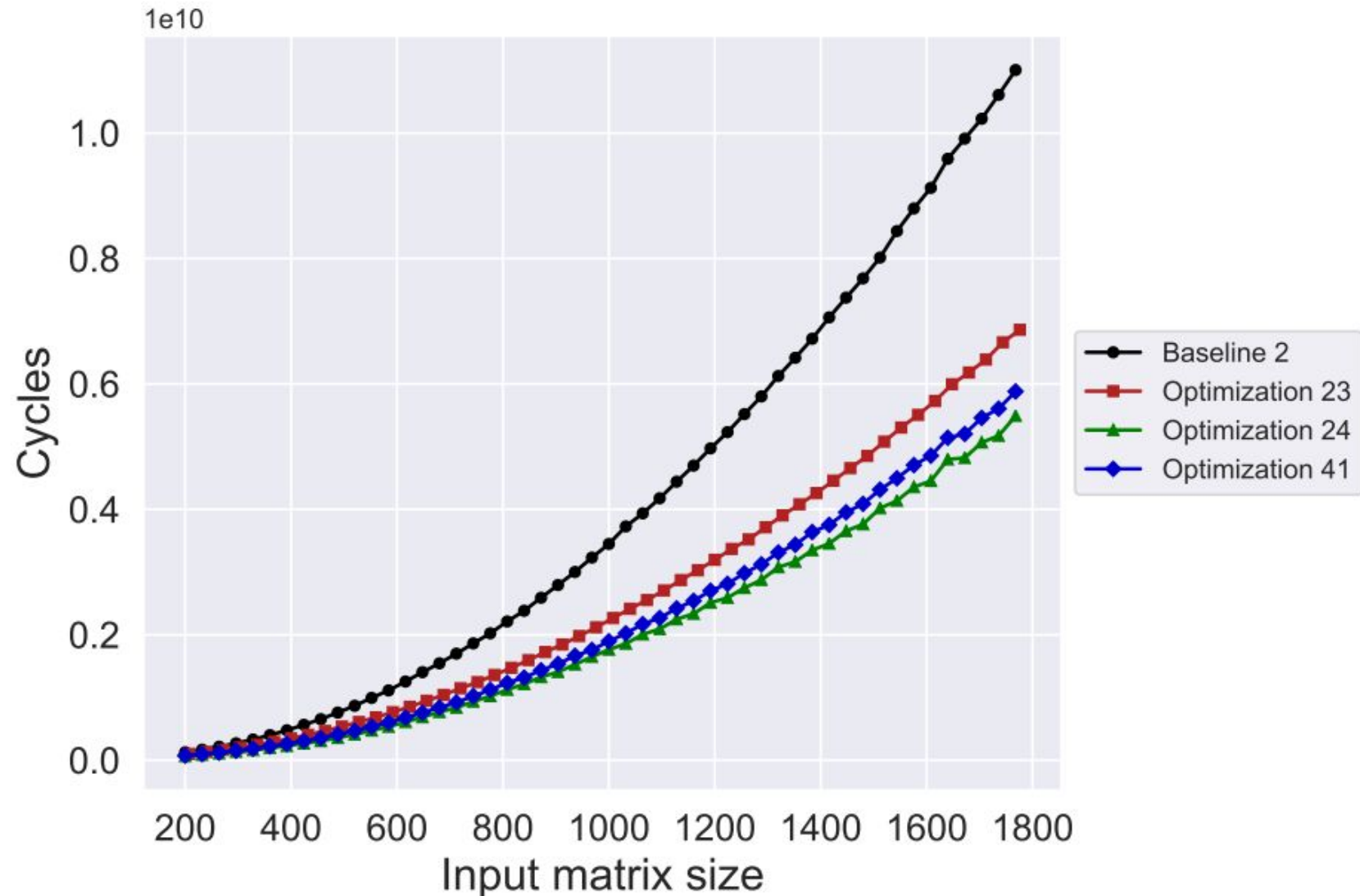
Roofline Scalar Algorithmic Optimizations



Performance BLAS Optimizations

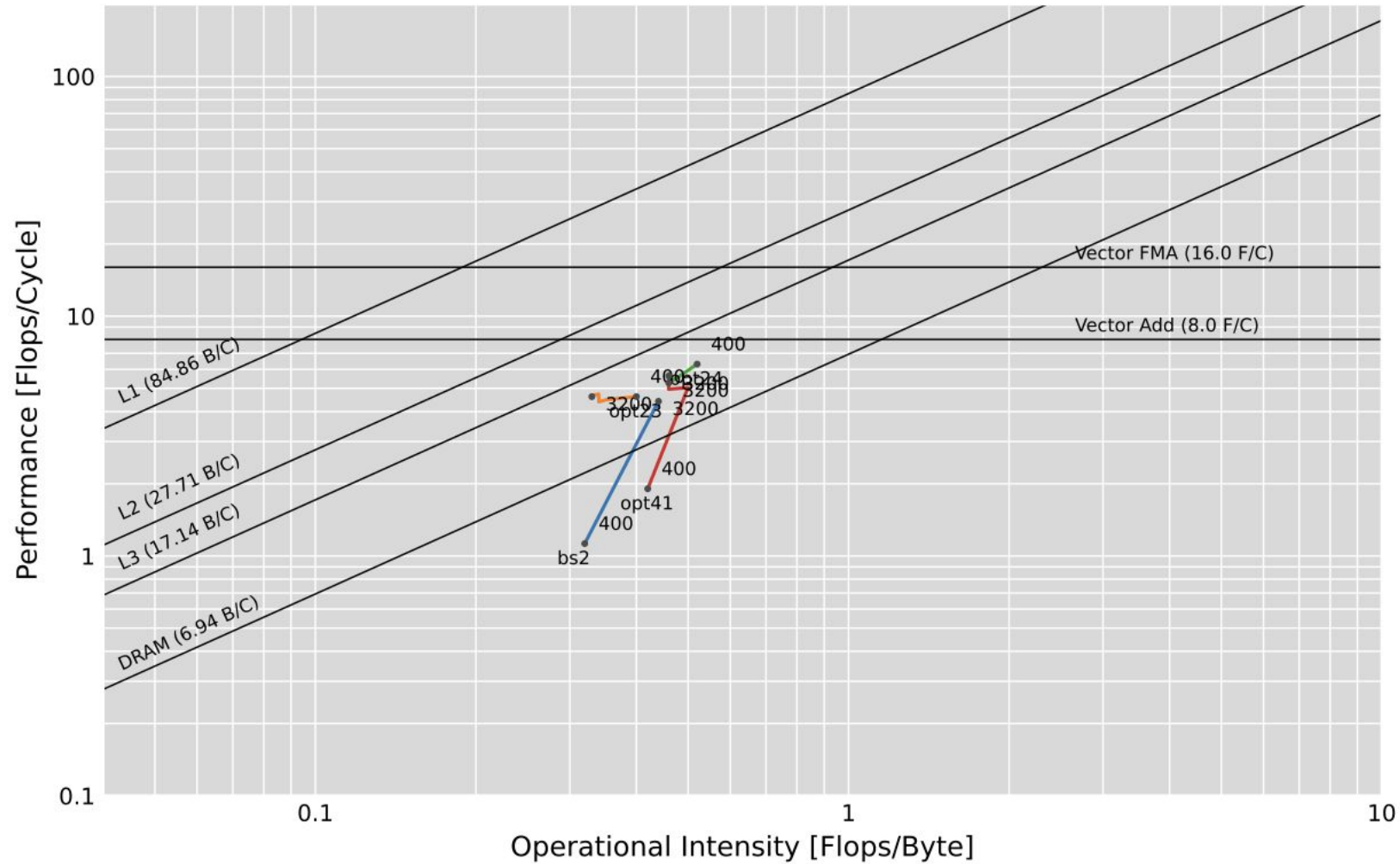


Runtime BLAS Optimizations



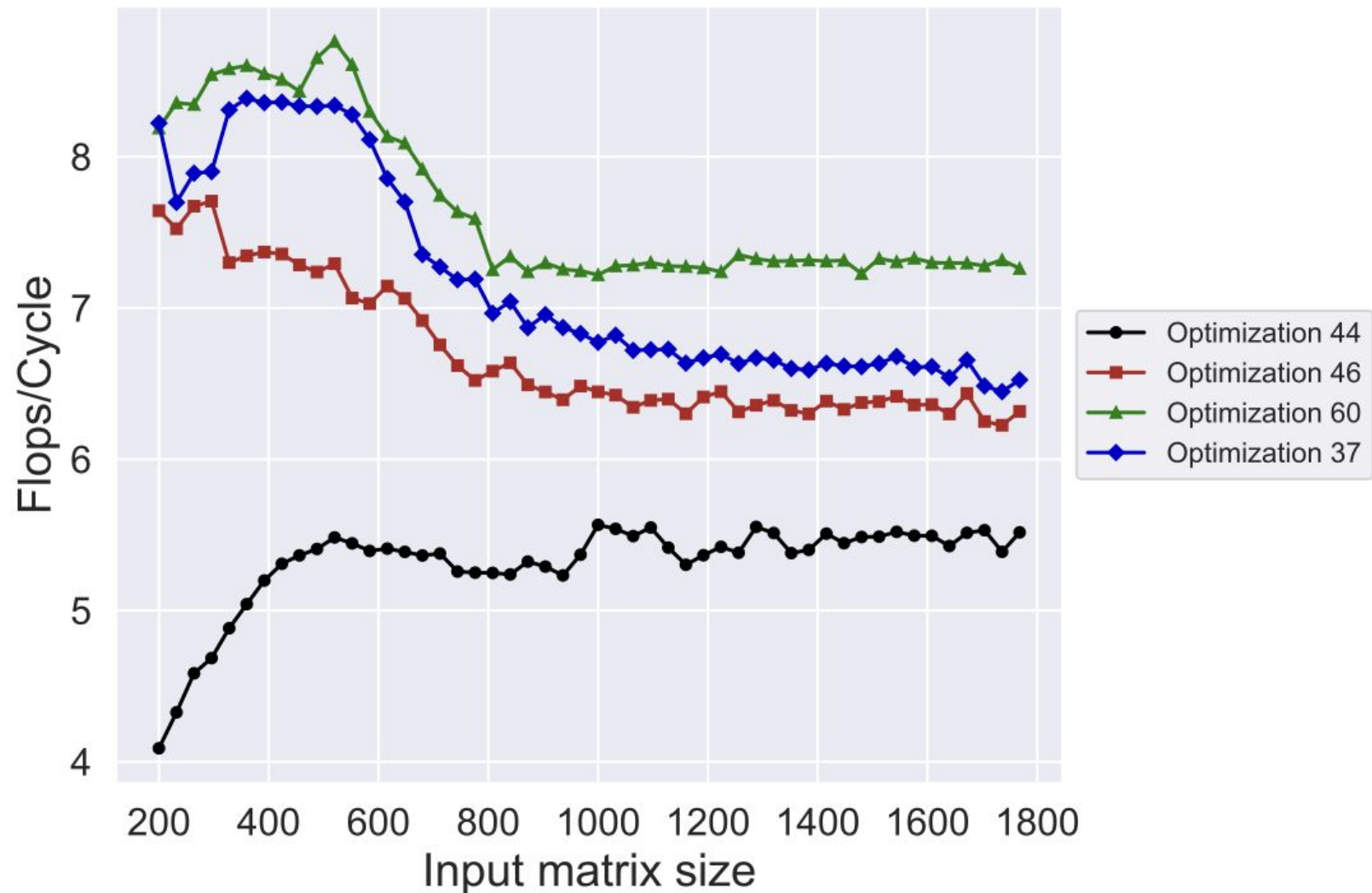
NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

Roofline BLAS Optimizations



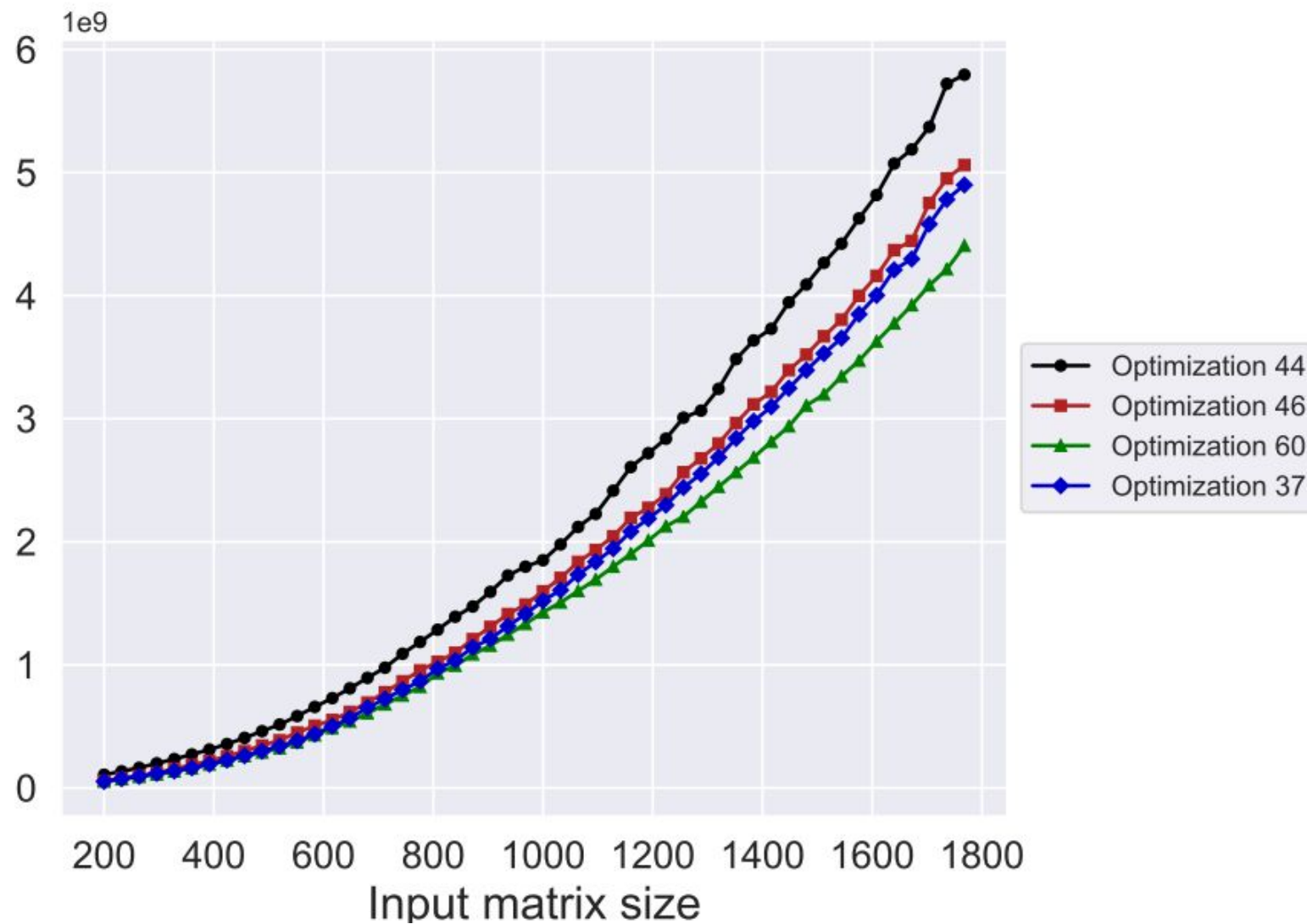
NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

Performance Vectorized Optimizations

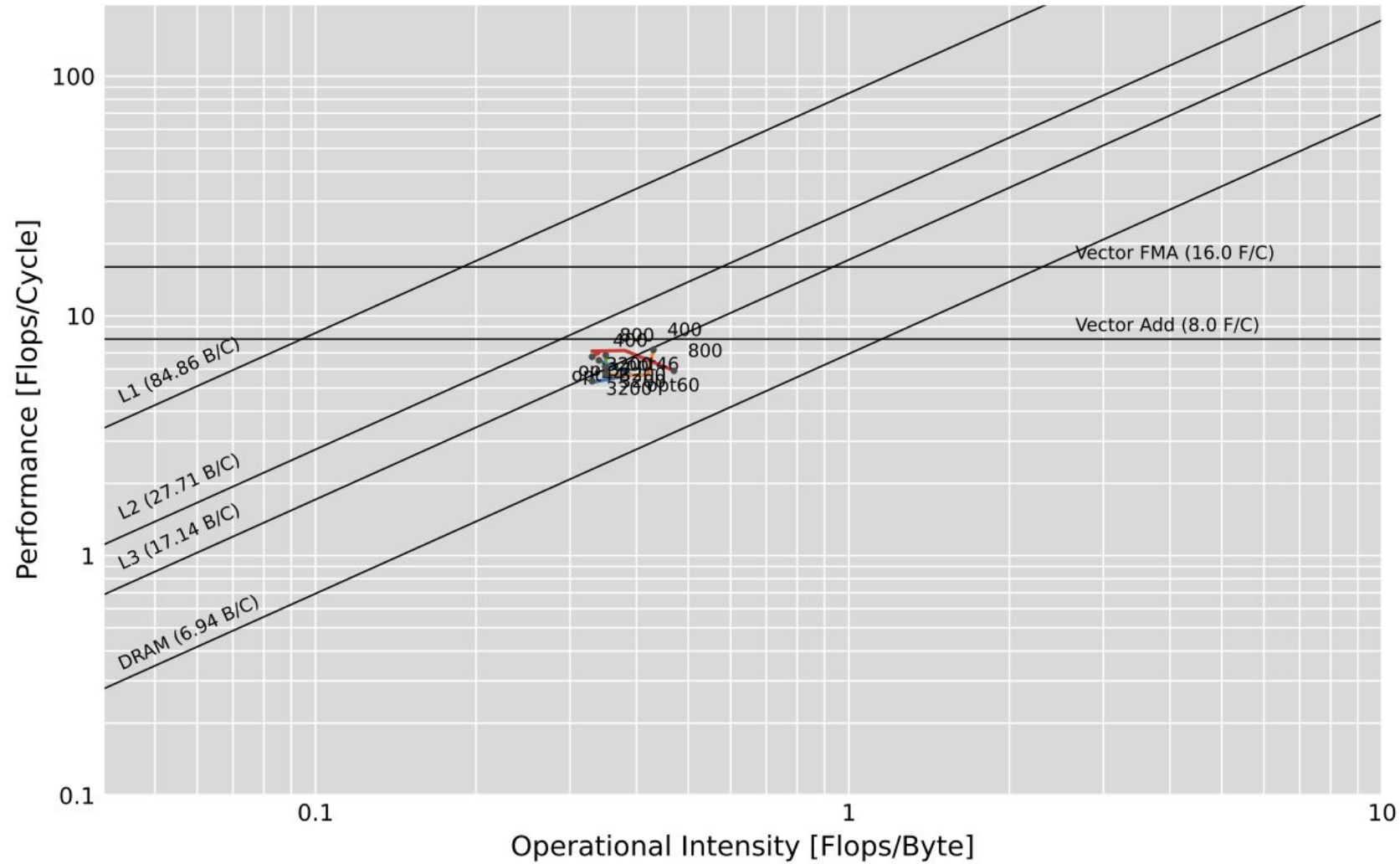


NNMF (double precision) on i5-6600K, 3.5 GHz, $r = 16$

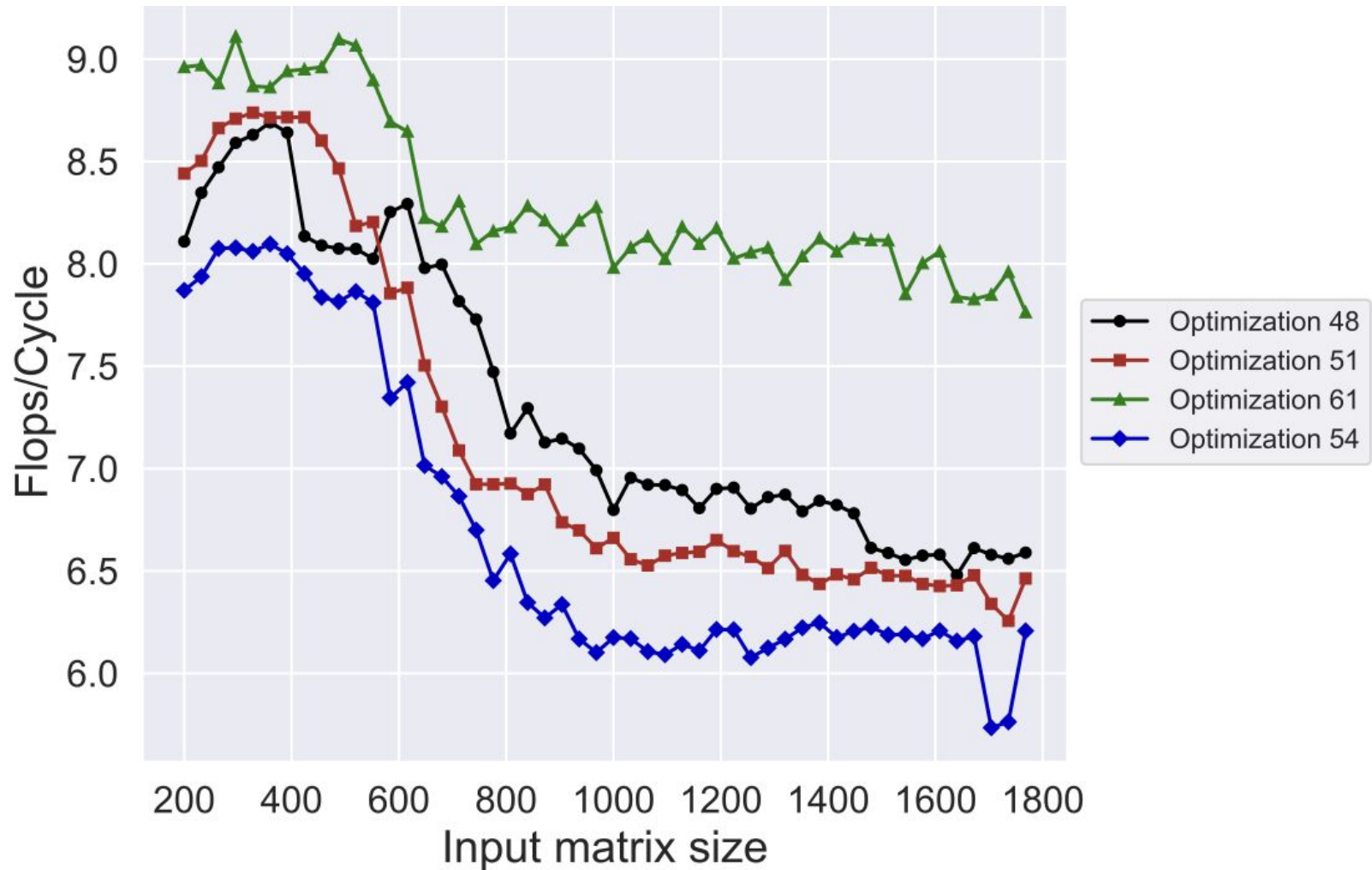
Runtime Vectorized Optimizations



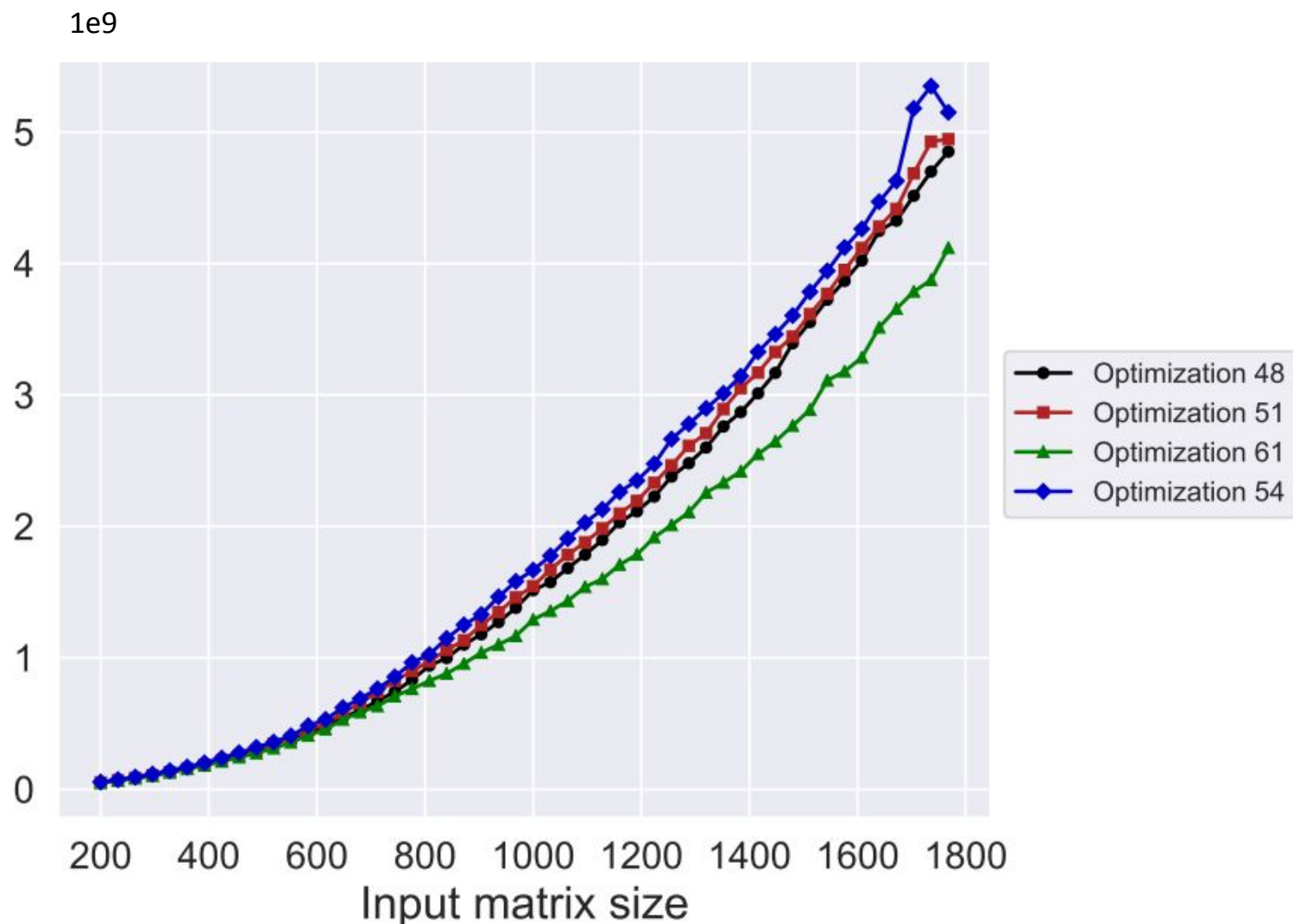
Roofline Vectorized Optimizations



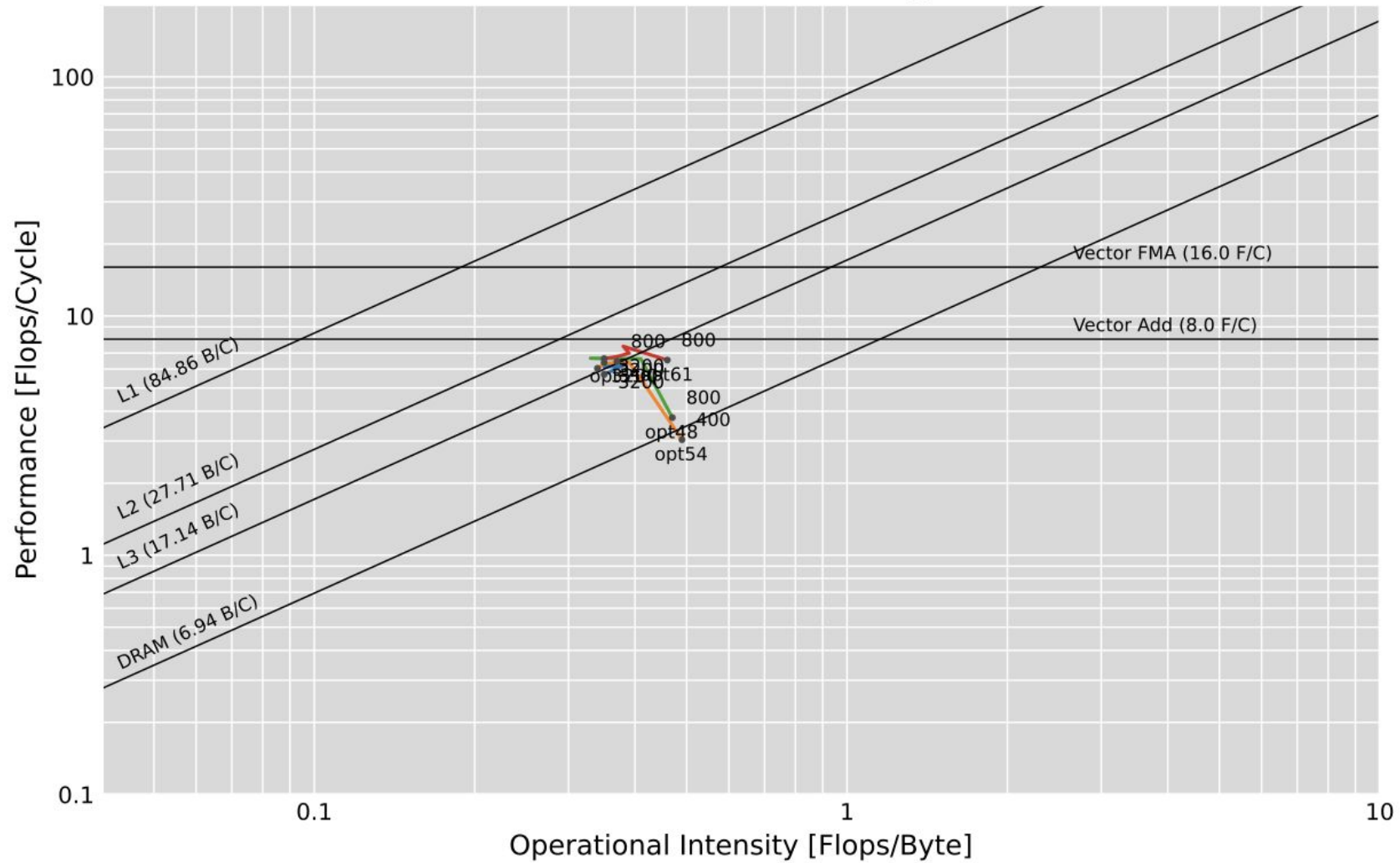
Performance Vectorized Algorithmic Optimizations



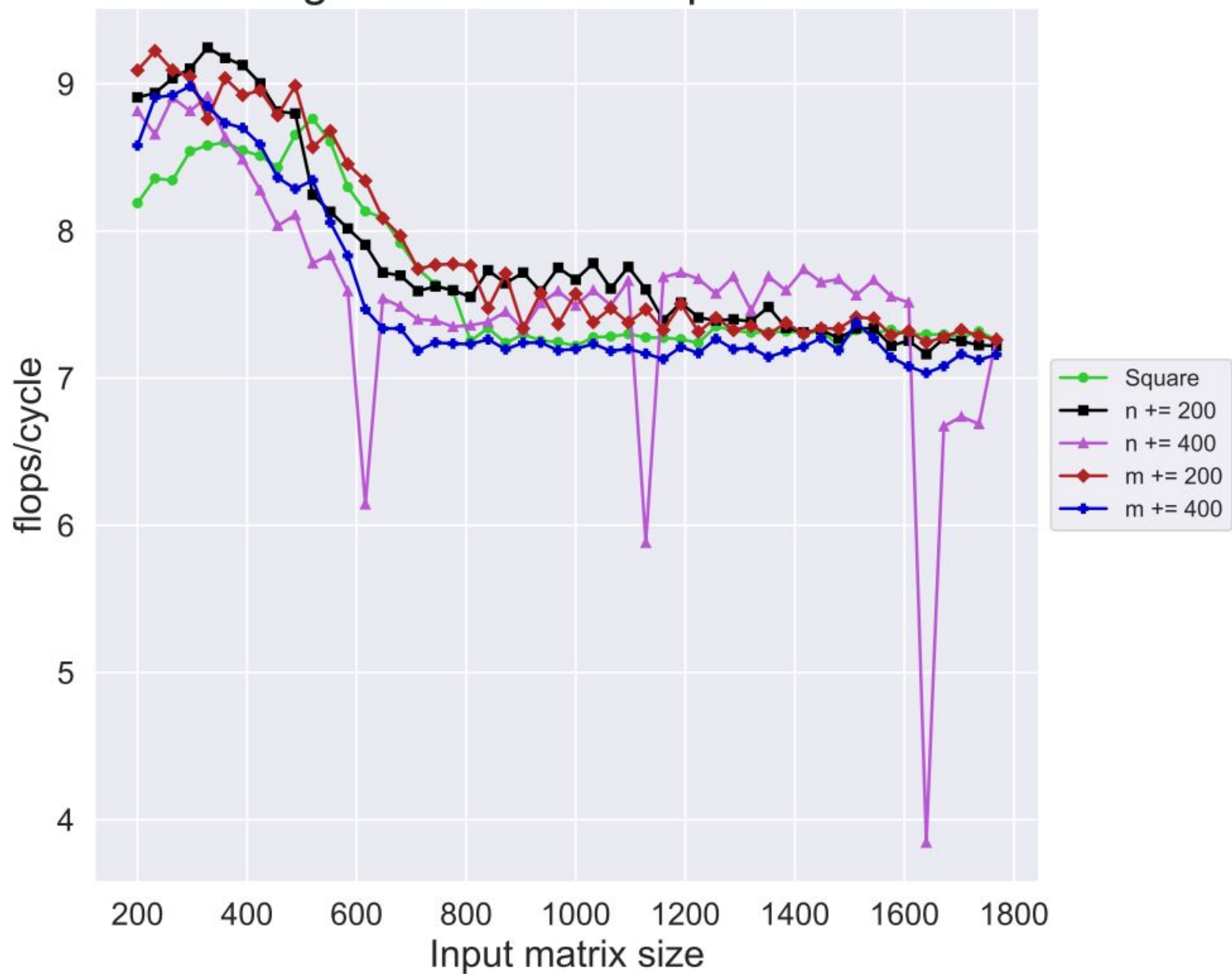
Runtime Vectorized Algorithmic Optimizations



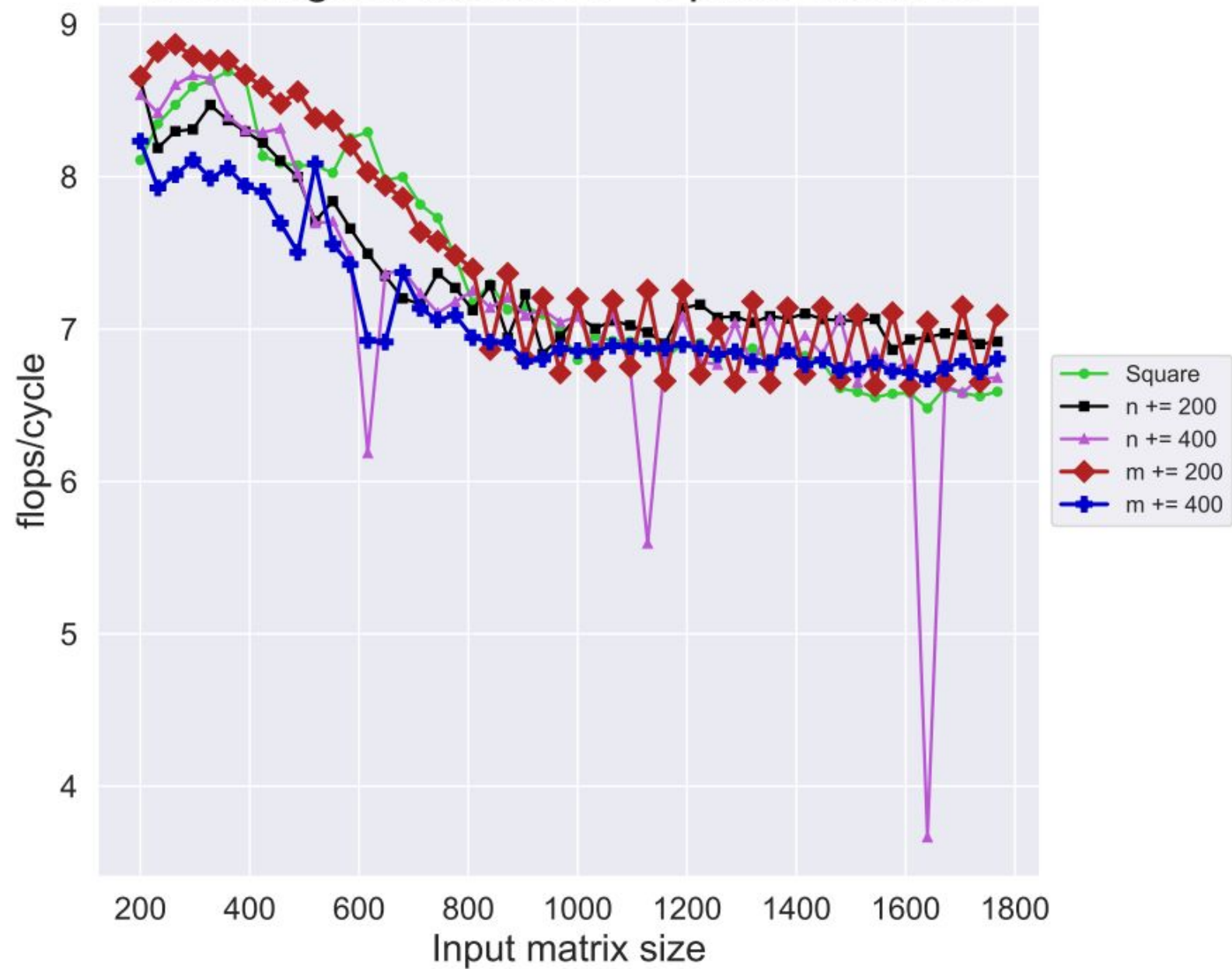
Roofline Vectorized Algorithmic Optimizations



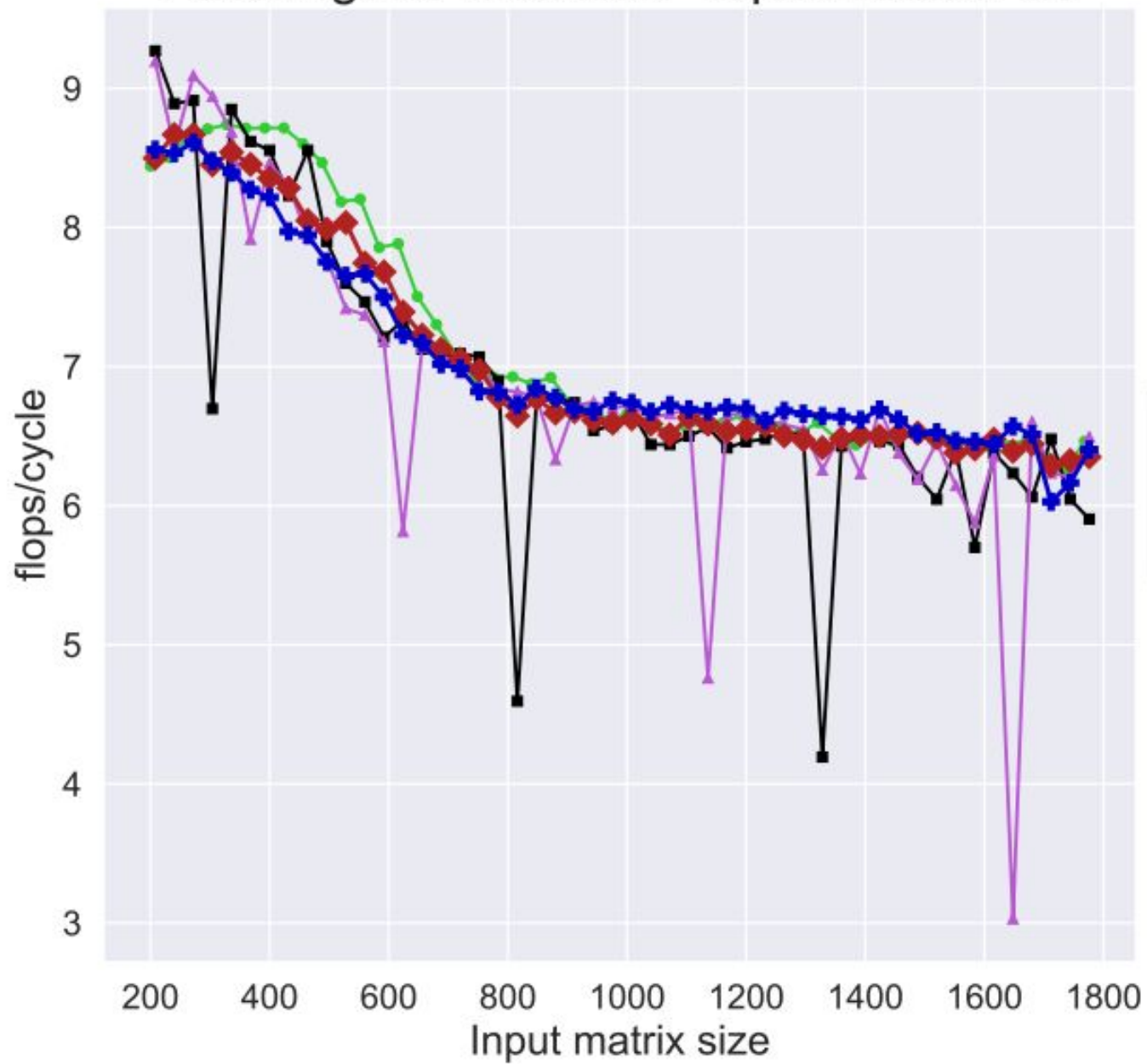
Rectangular matrices - Optimization 60



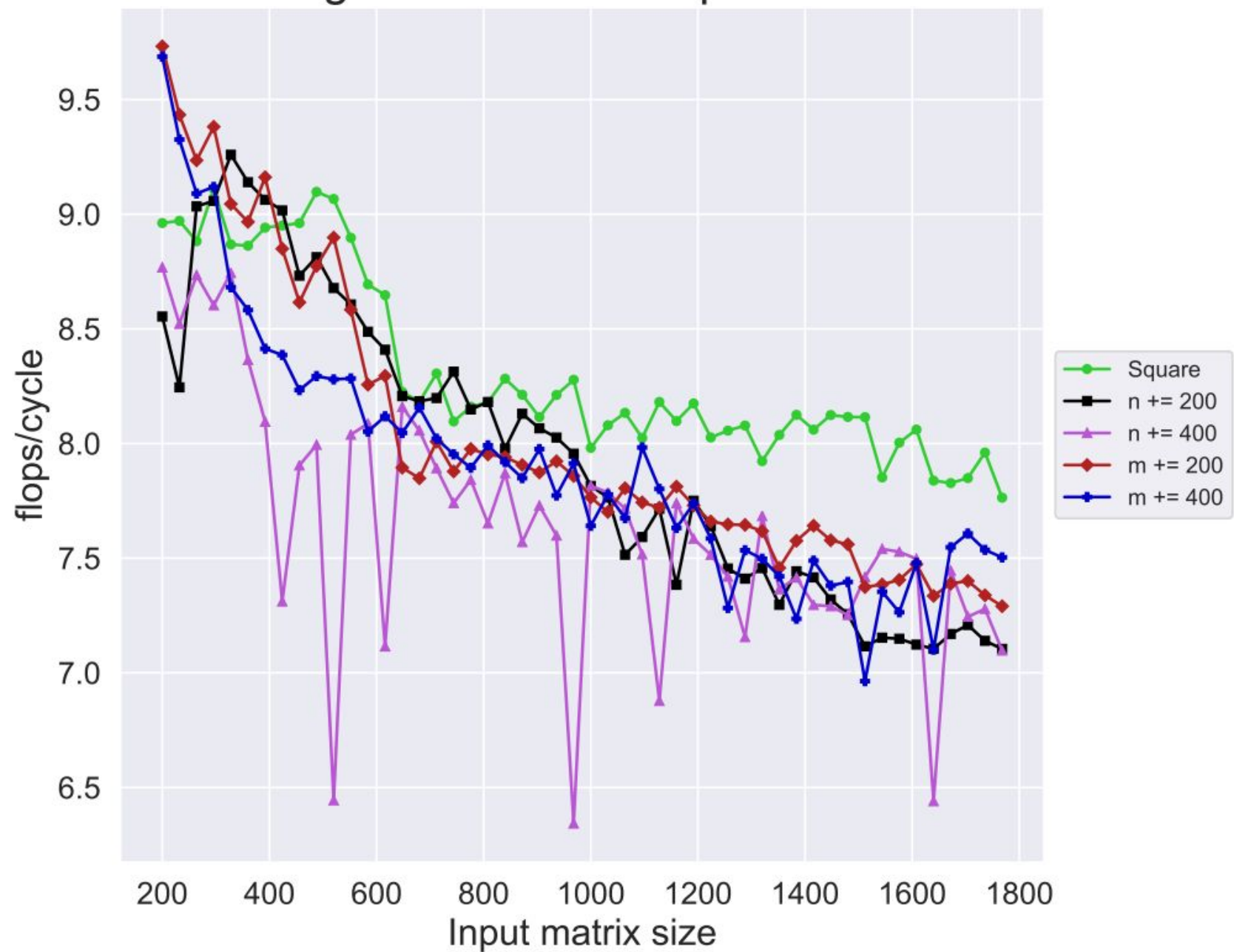
Rectangular matrices - Optimization 48



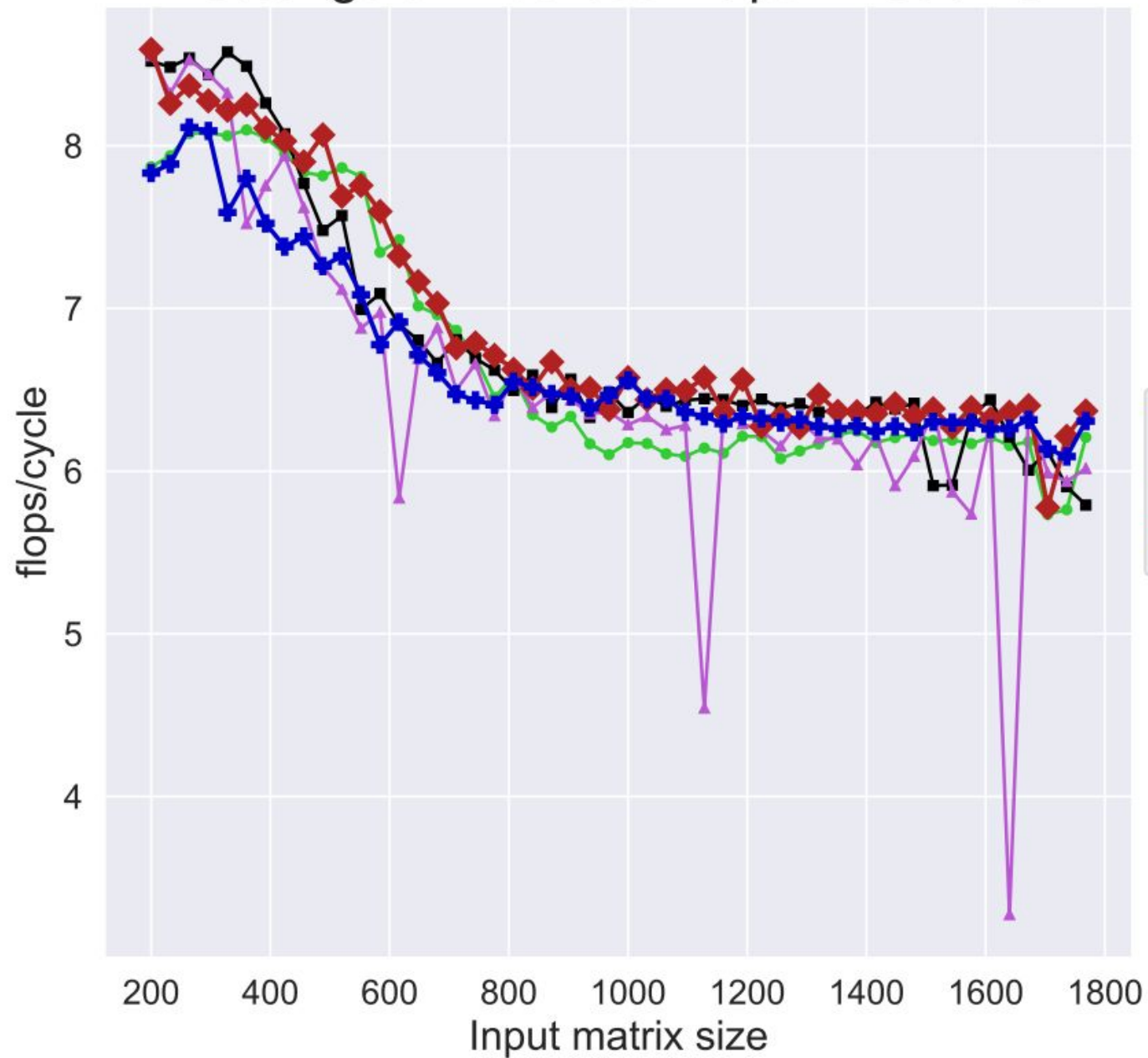
Rectangular matrices - Optimization 51



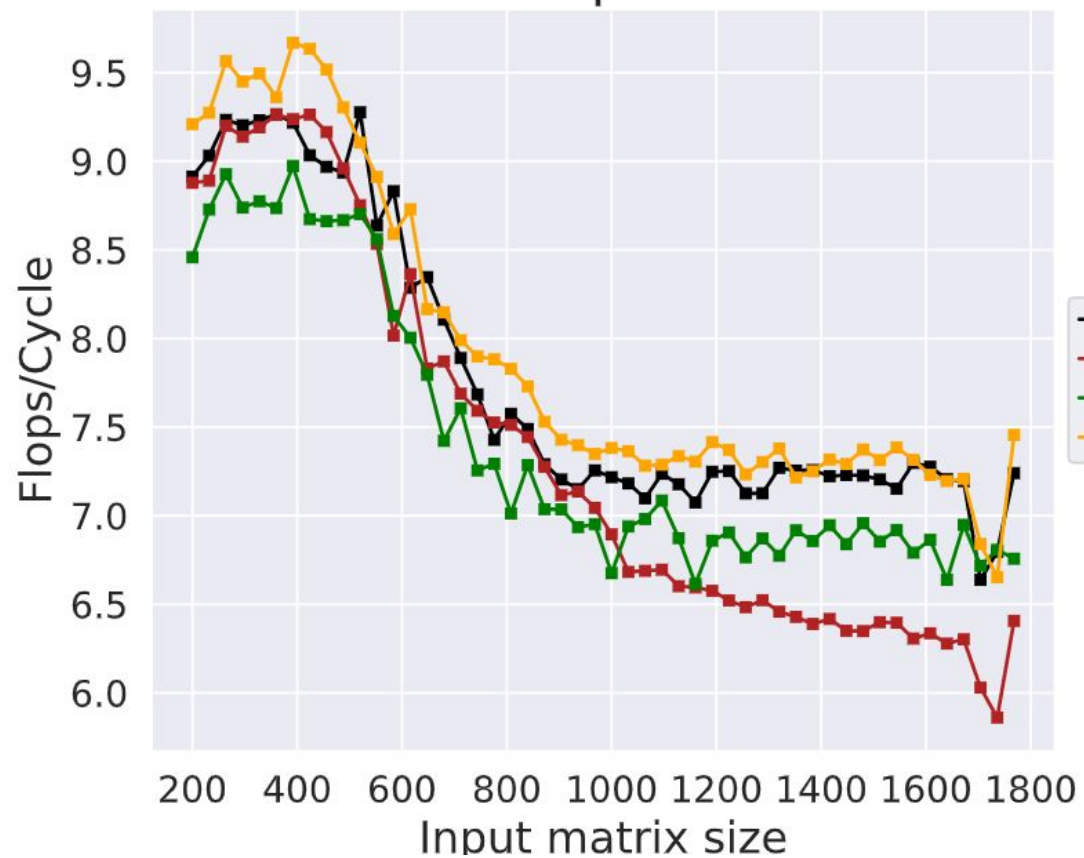
Rectangular matrices - Optimization 61



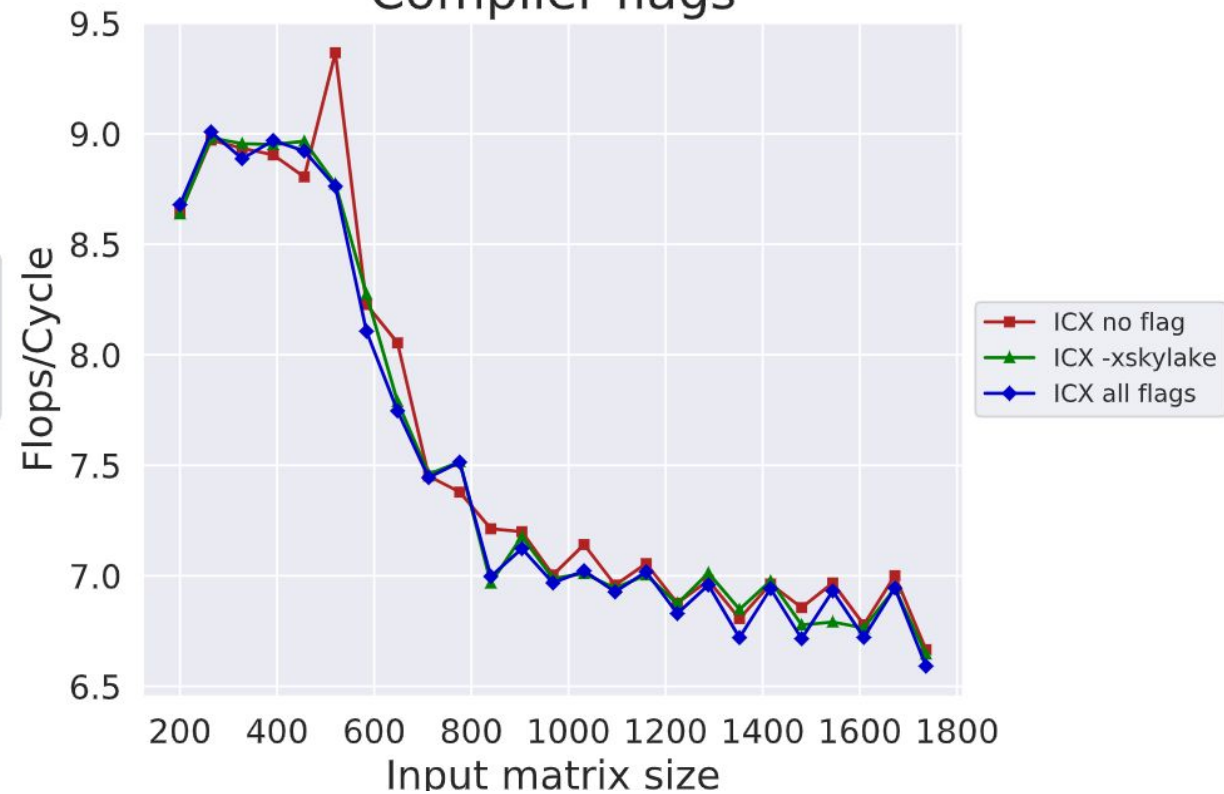
Rectangular matrices - Optimization 54



Compilers



Compiler flags



Algorithmic optimization 2 – Interleave matrix multiplications

- Avoids having to store and reread the numerator and the denominator
- The same approach is used in the calculation of W^{n+1} as well

$$H_{[i,j]}^{n+1} = H_{[i,j]}^n \cdot \frac{\overbrace{((W^n)^T V)_{[i,j]}}^{\text{numerator - N}}}{\underbrace{((W^n)^T W^n H^n)_{[i,j]}}_{\substack{\text{denominator_left - Dl} \\ \text{denominator - D}}}}$$

$V : m \times n$
 $W : m \times r$
 $H : r \times n$
 $Dl : r \times r$
 $N : r \times n$
 $D : r \times n$

```

N = matrix_mul(WT, V)
D = matrix_mul(Dl, H)
for(i=0; i<r; i++)
    for(j=0; j<n; j++)
        H[i][j] = H[i][j] * N[i][j] / D[i][j]
    
```

contain triple loops

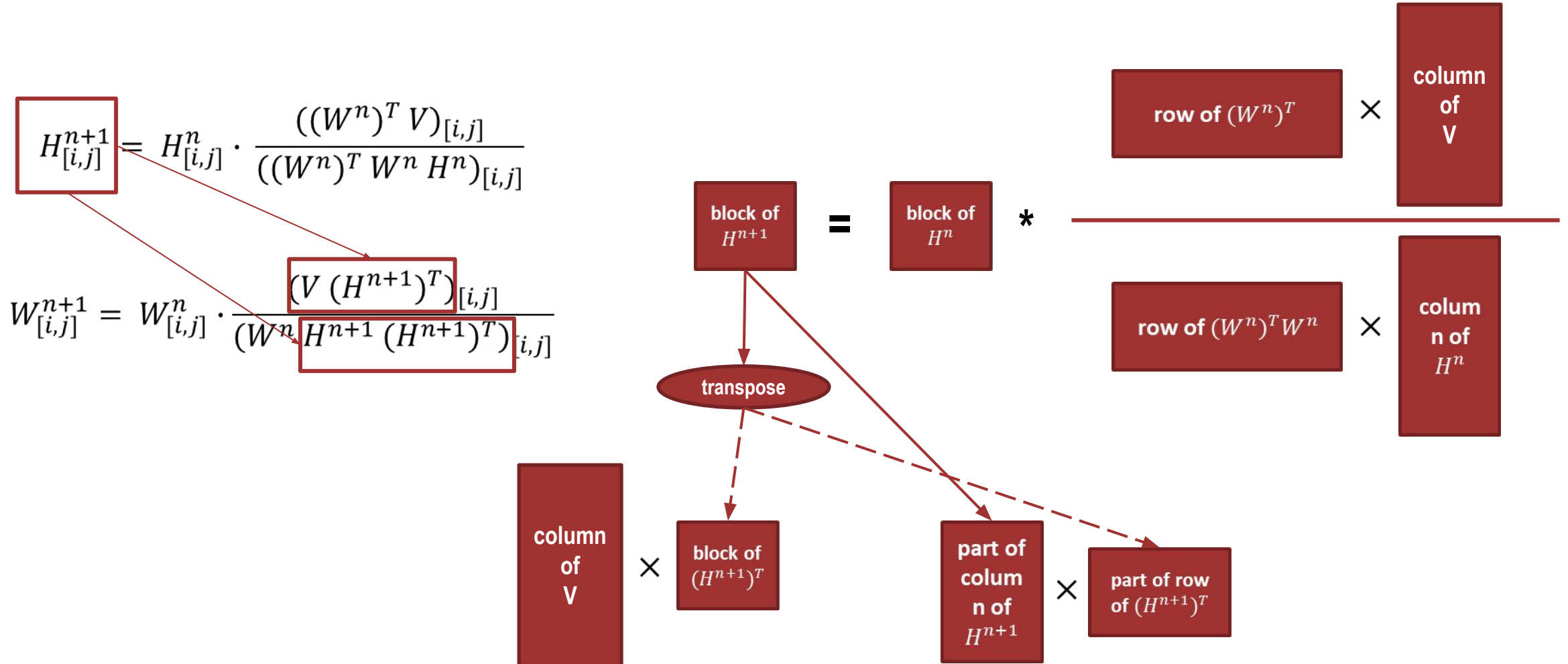


```

for(i=0; i<r; i++)
    for(j=0; j<n; j++)
        accumulator_N = 0
        accumulator_D = 0
        for(k=0; k<m; k++)
            accumulator_N += WT[i][k] * V[k][j]
            if(k<r)
                accumulator_D += Dl[i][k] * H[k][j]
        H[i][j] = H[i][j] * accumulator_N / accumulator_D
    
```

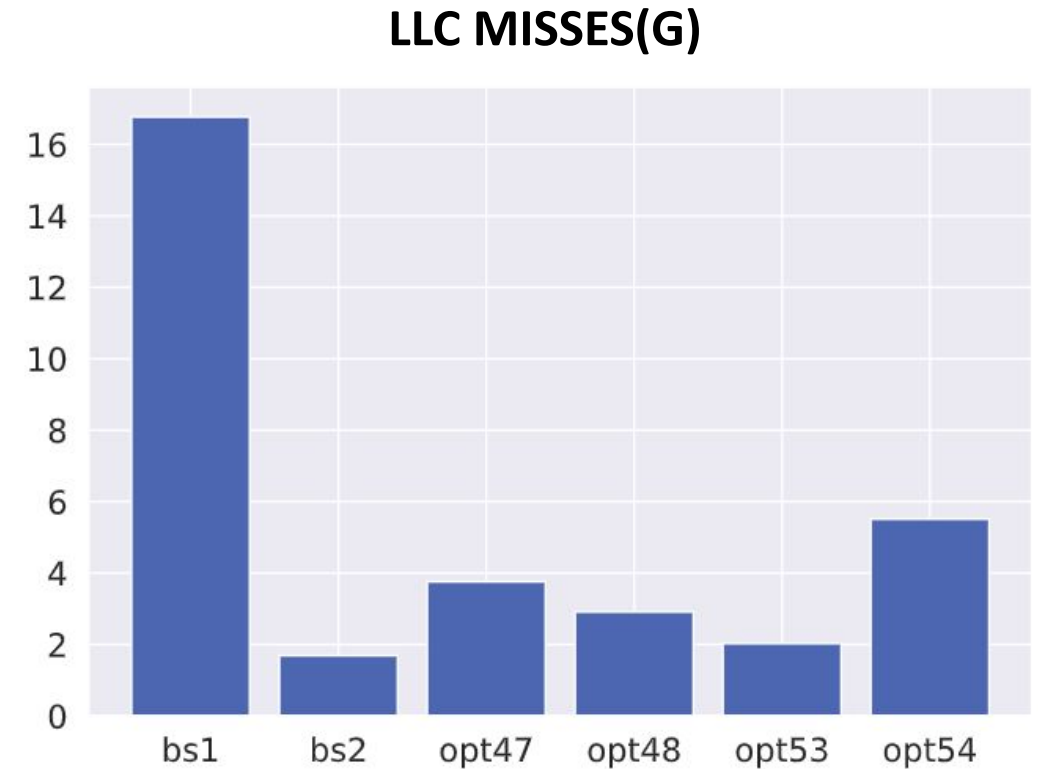
Algorithmic optimization 3 – Reuse block of H

- The calculated block of H^{n+1} is immediately used in the calculation of $V(H^{n+1})^T$ and $H^{n+1}(H^{n+1})^T$



Additional analysis

- opt47 reduces cache misses by blocking
- opt53 reduces cache misses reusing W
- opt60 computes the approximation matrix WH one block at the time
- opt61 decrease in cache misses is higher than 60 and 53 combined



V : 800x800

W: 800x16

H: 800x16