Volume Estimation

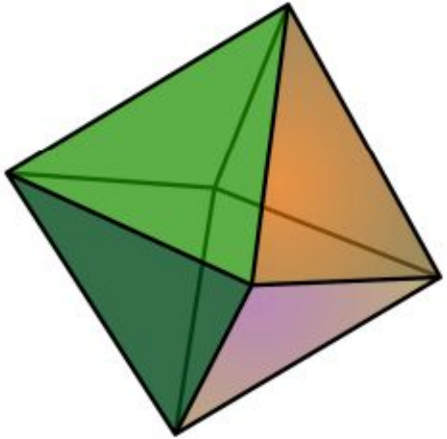
Constantin Dragancea
David Buzatu
François Costa
Roxana Stiuca

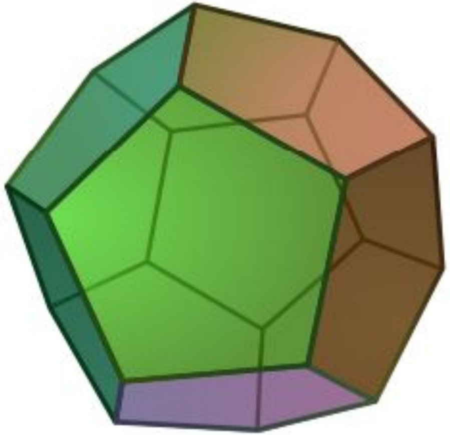
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Algorithm

Input: convex polytope P = {Ax <= b}, A matrix MxN, b vector of length M.

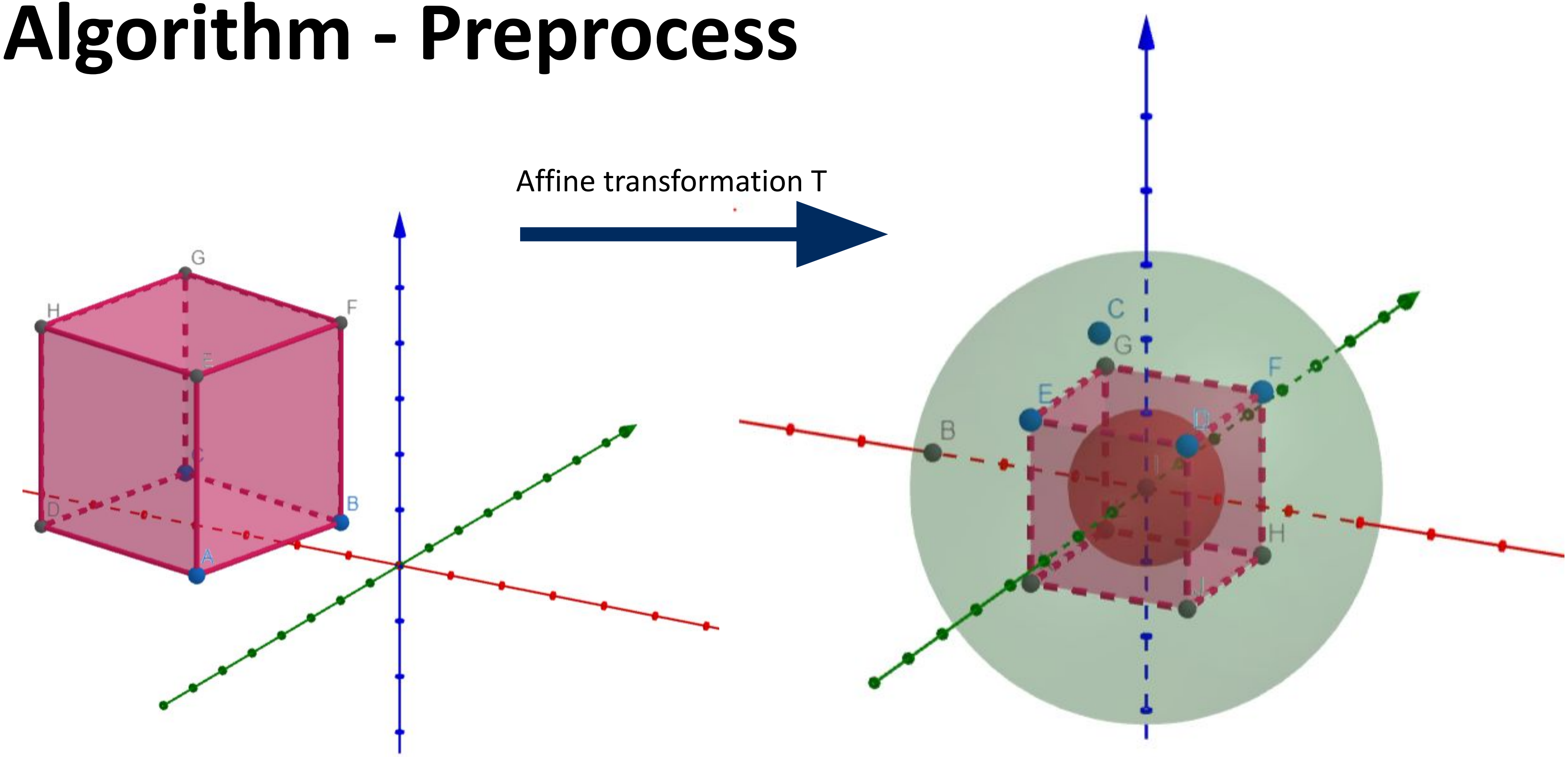
Output: volume.





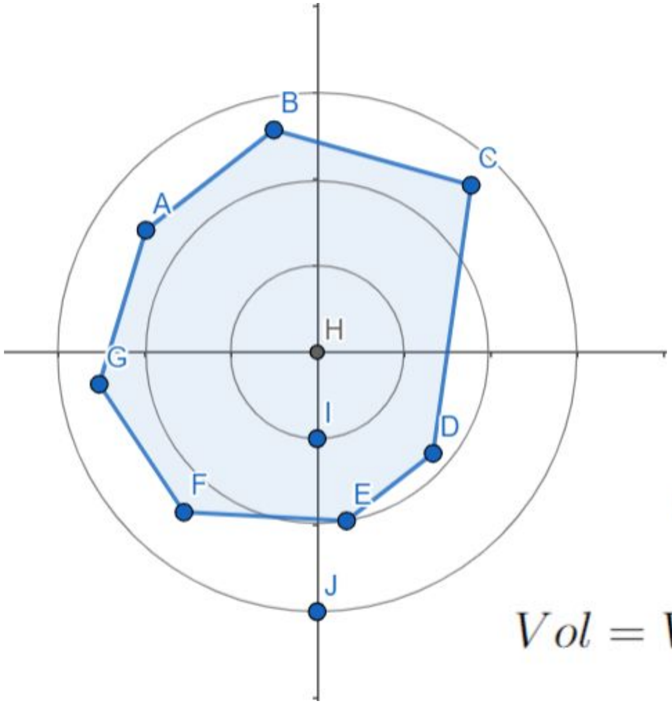
https://ti.inf.ethz.ch/ew/courses/Geo20/lecture/gca20-9.pdf

Algorithm - Preprocess



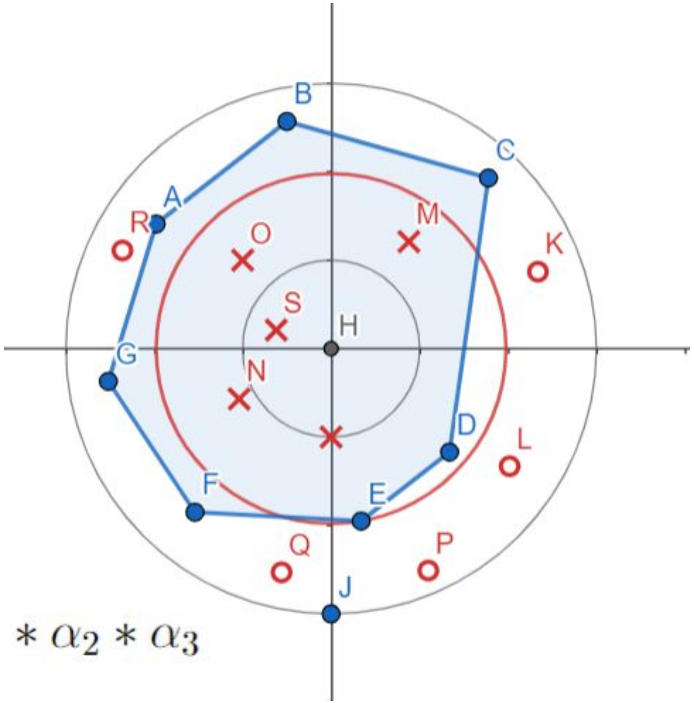
Affine transformation T

Algorithm - Estimate Volume



$$\alpha_i = \frac{step\_size}{count_{\times}}$$

$$Vol = Vol(unit\_ball) * \alpha_1 * \alpha_2 * \alpha_3$$

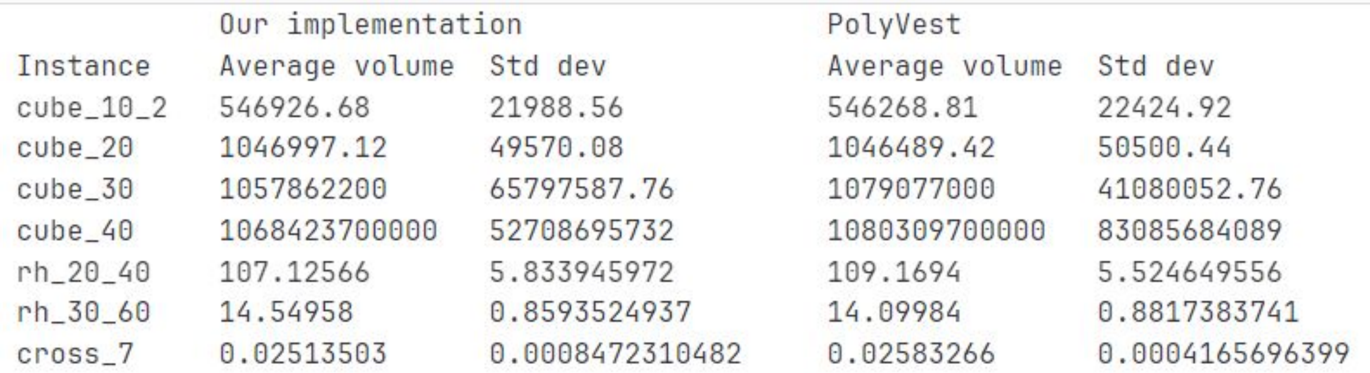


Experimental Setup

Machine: Intel(R) Core(™) i7-10750H CPU@2.60GHz

Compiler: GCC 13.1.1. Flags: -O3 -ffast-math -mfma

Correctness checks: using C++ test suite & statistical results.



	Our implementation		PolyVest	
Instance	Average volume	Std dev	Average volume	Std dev
cube_10_2	546926.68	21988.56	546268.81	22424.92
cube_20	1046997.12	49570.08	1046489.42	50500.44
cube_30	1057862200	65797587.76	1079077000	41080052.76
cube_40	1068423700000	52708695732	1080309700000	83085684089
rh_20_40	107.12566	5.833945972	109.1694	5.524649556
rh_30_60	14.54958	0.8593524937	14.09984	0.8817383741
cross_7	0.02513503	0.0008472310482	0.02583266	0.0004165696399

Baseline implementation

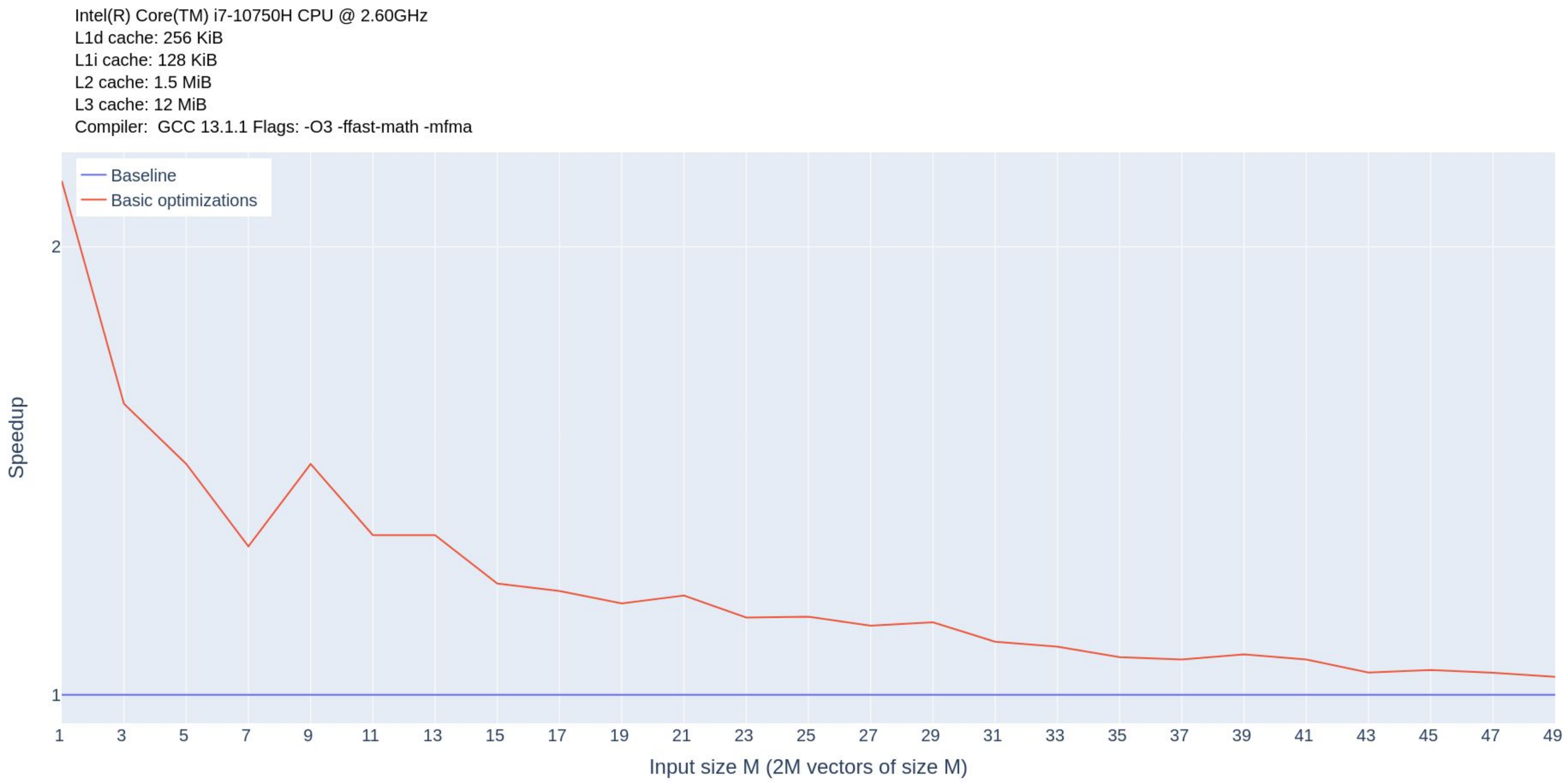
- **Preprocess - O(N^3)**
- **Estimate Volume - O(N^4 (logN)^2)**
  - **Walk** - O(N^2) called 1600*(N*logN)^2 times

Bottleneck: Walk!

Basic optimizations

- Inline functions
- Remove repeated operations (eg. memory allocations on every function call)
- Pre-compute constants
- Simplify mathematical expressions
- Scalar replacement (eg. n instead of p->n)
- Replace matrix-matrix operations with matrix-vector, in Preprocess
- Use xoshiro random number generator

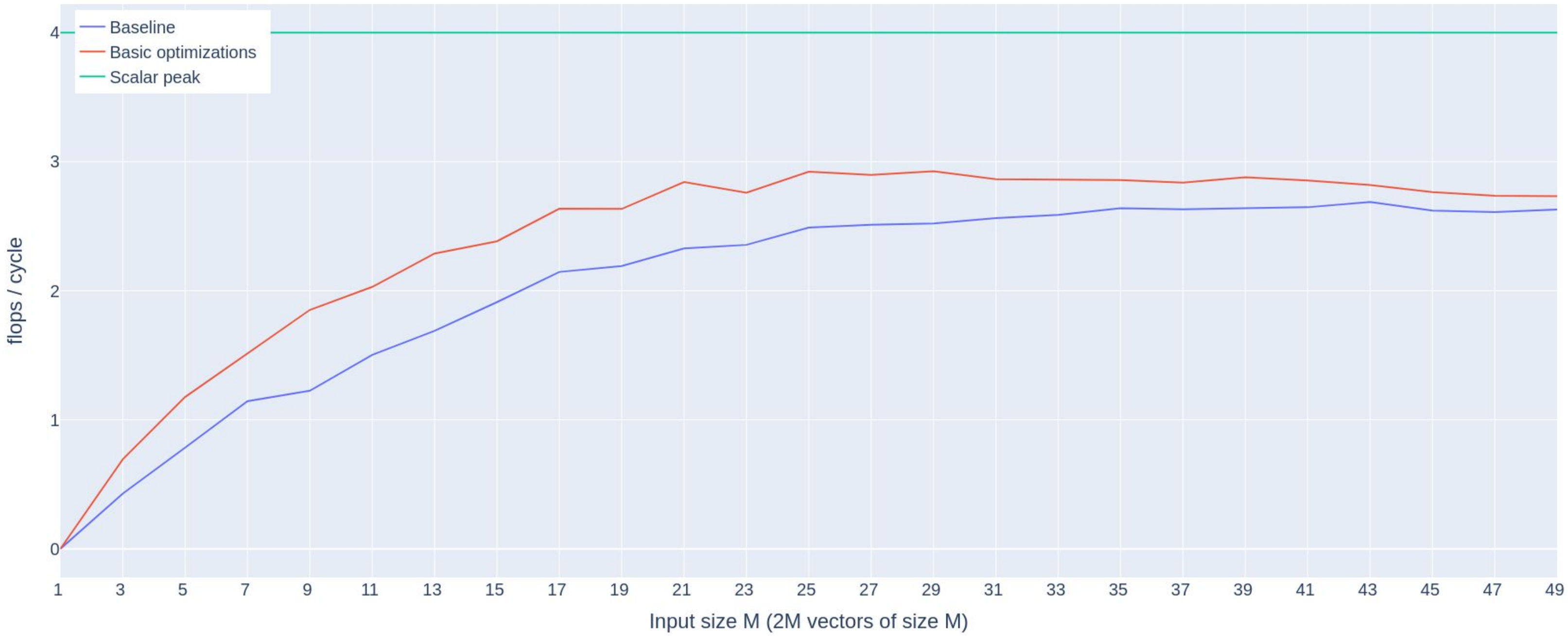
Basic optimizations



Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
L1d cache: 256 KiB
L1i cache: 128 KiB
L2 cache: 1.5 MiB
L3 cache: 12 MiB
Compiler: GCC 13.1.1 Flags: -O3 -ffast-math -mfma

Basic optimizations

Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
L1d cache: 256 KiB
L1i cache: 128 KiB
L2 cache: 1.5 MiB
L3 cache: 12 MiB
Compiler: GCC 13.1.1 Flags: -O3 -ffast-math -mfma

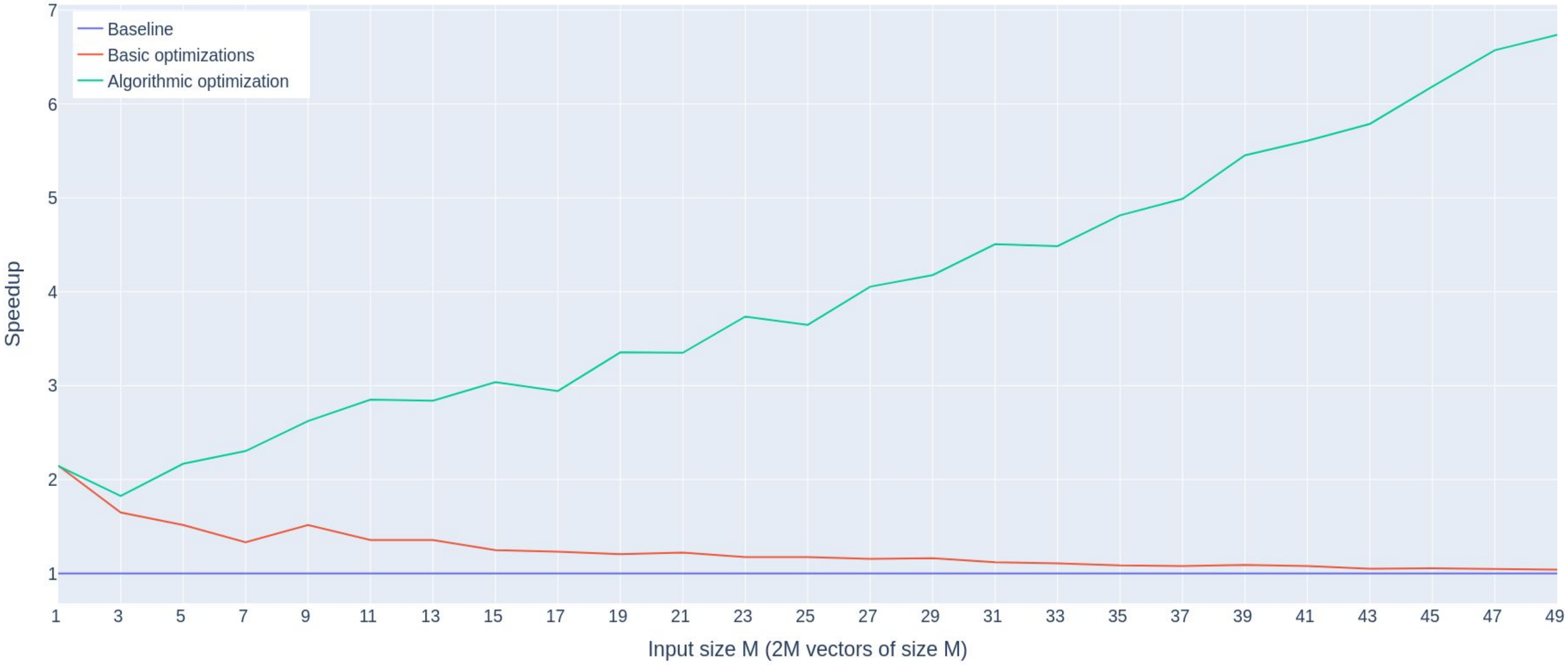


Algorithmic optimization

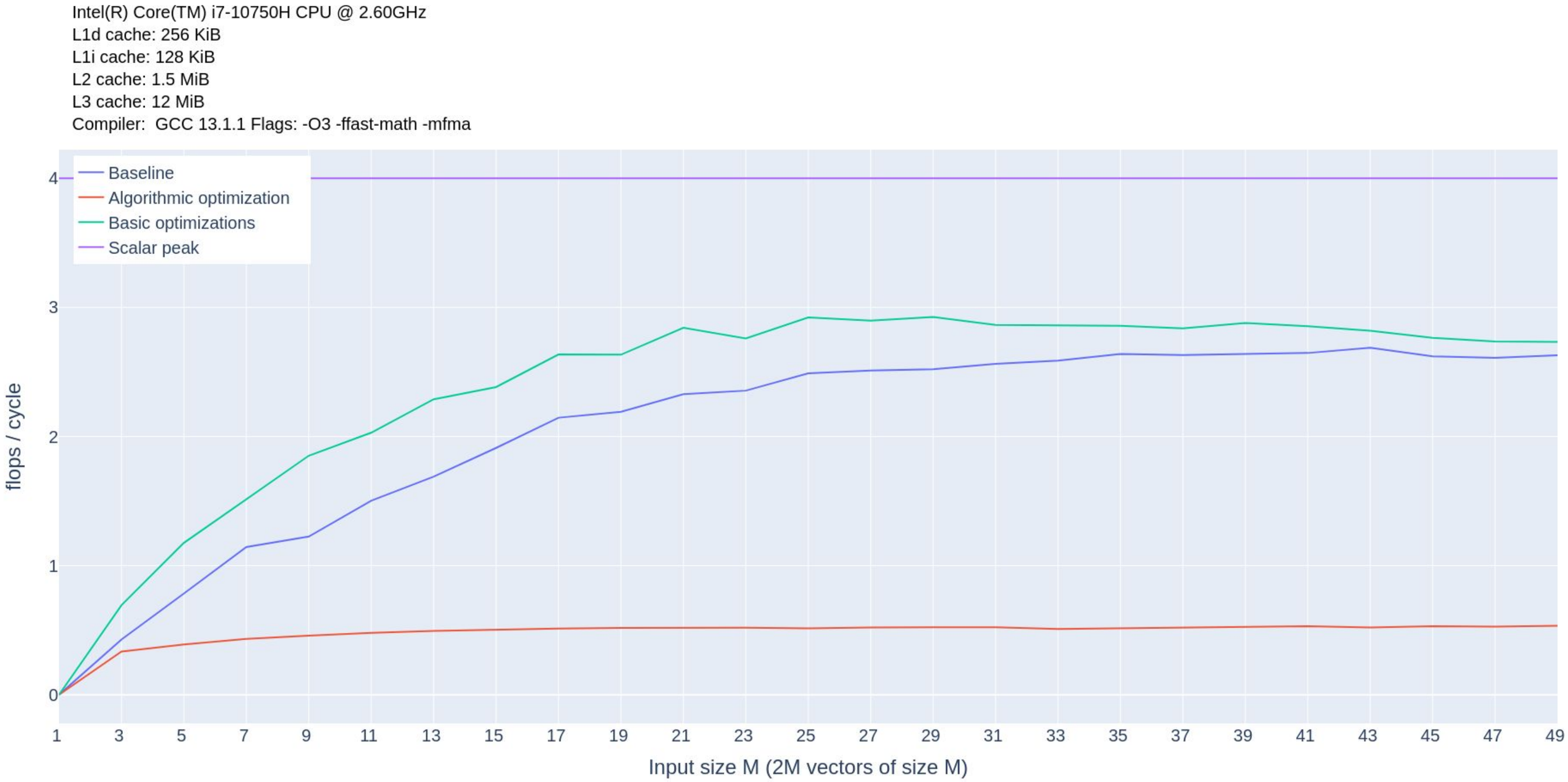
- The walk functions computes the matrix-vector product (A / A.col(rand_dir)) * sampling_points in O(N^2)
- Idea: pre-compute A * sampling_points once, take the division as a common factor. The result can be updated in O(N)
- Overall, reduce complexity from O*(N^4) to O*(N^3).

Algorithmic optimization

Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
L1d cache: 256 KiB
L1i cache: 128 KiB
L2 cache: 1.5 MiB
L3 cache: 12 MiB
Compiler: GCC 13.1.1 Flags: -O3 -ffast-math -mfma



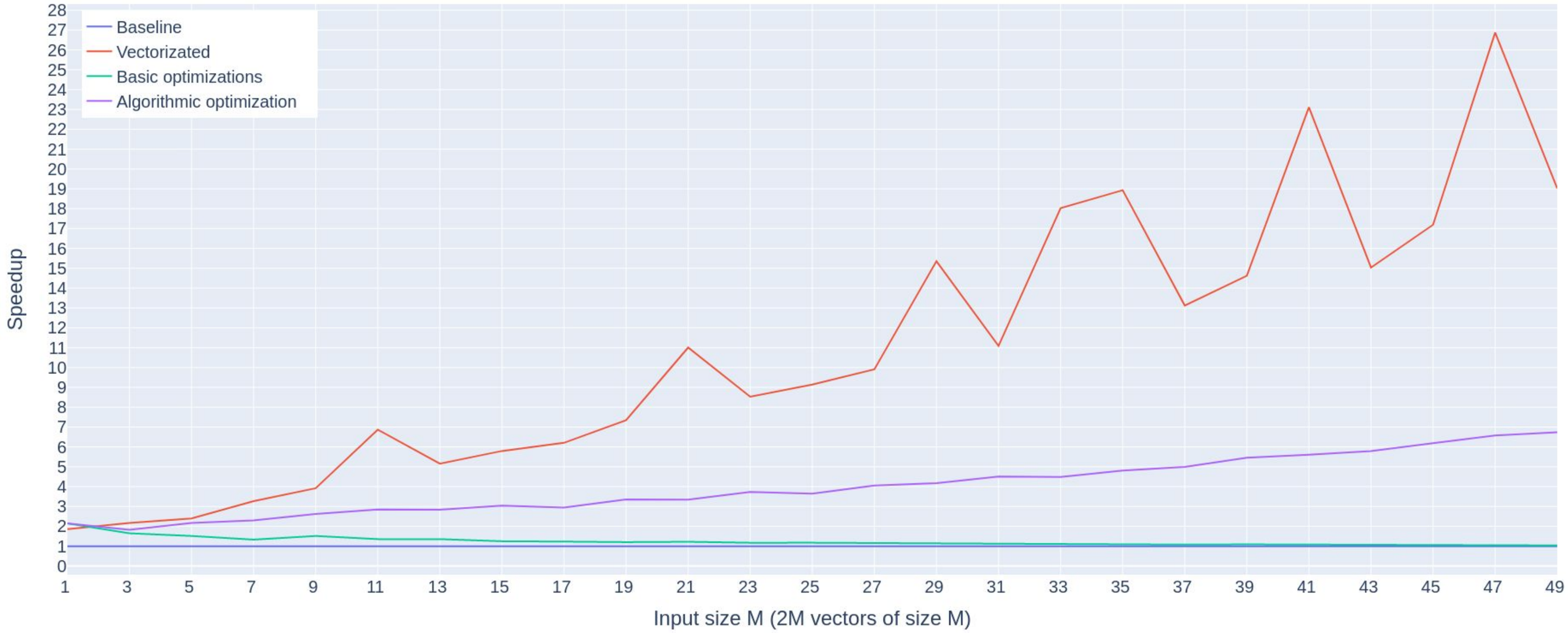
Algorithmic optimization



Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
L1d cache: 256 KiB
L1i cache: 128 KiB
L2 cache: 1.5 MiB
L3 cache: 12 MiB
Compiler: GCC 13.1.1 Flags: -O3 -ffast-math -mfma

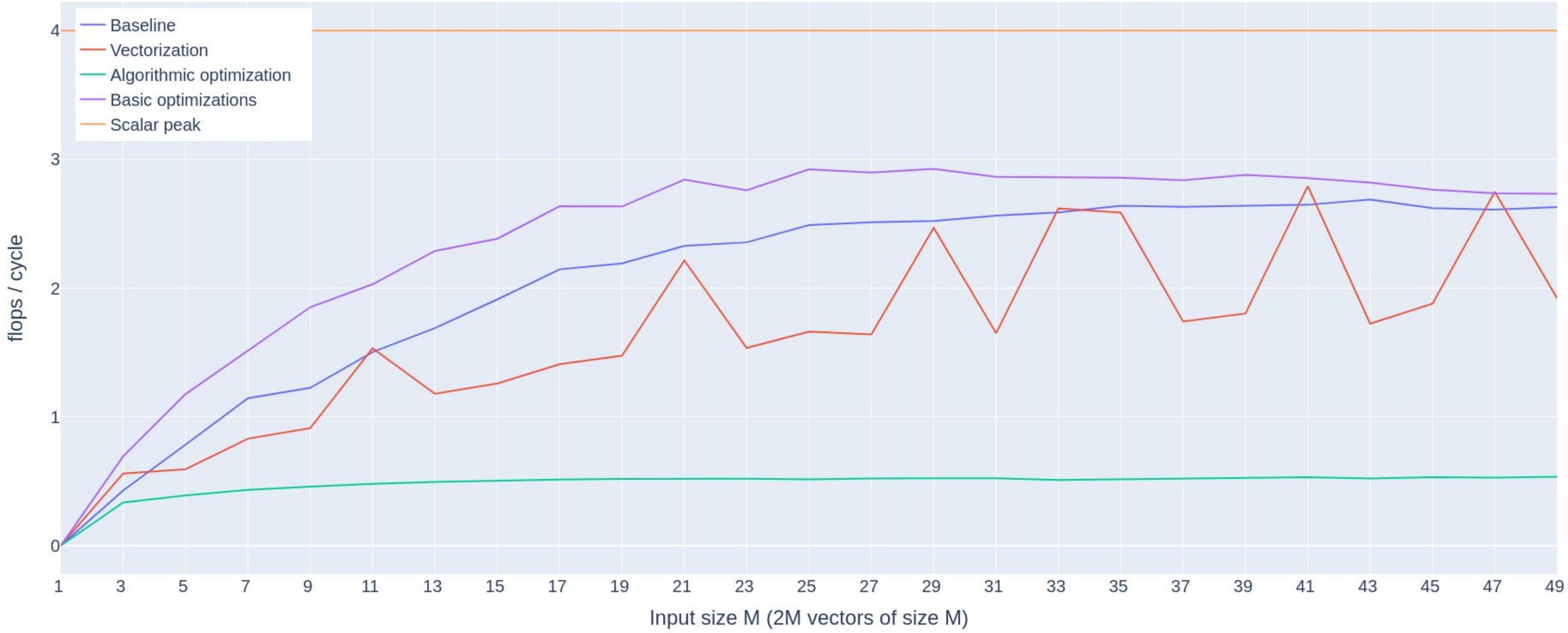
Vectorization

Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
L1d cache: 256 KiB
L1i cache: 128 KiB
L2 cache: 1.5 MiB
L3 cache: 12 MiB
Compiler: GCC 13.1.1 Flags: -O3 -ffast-math -mfma

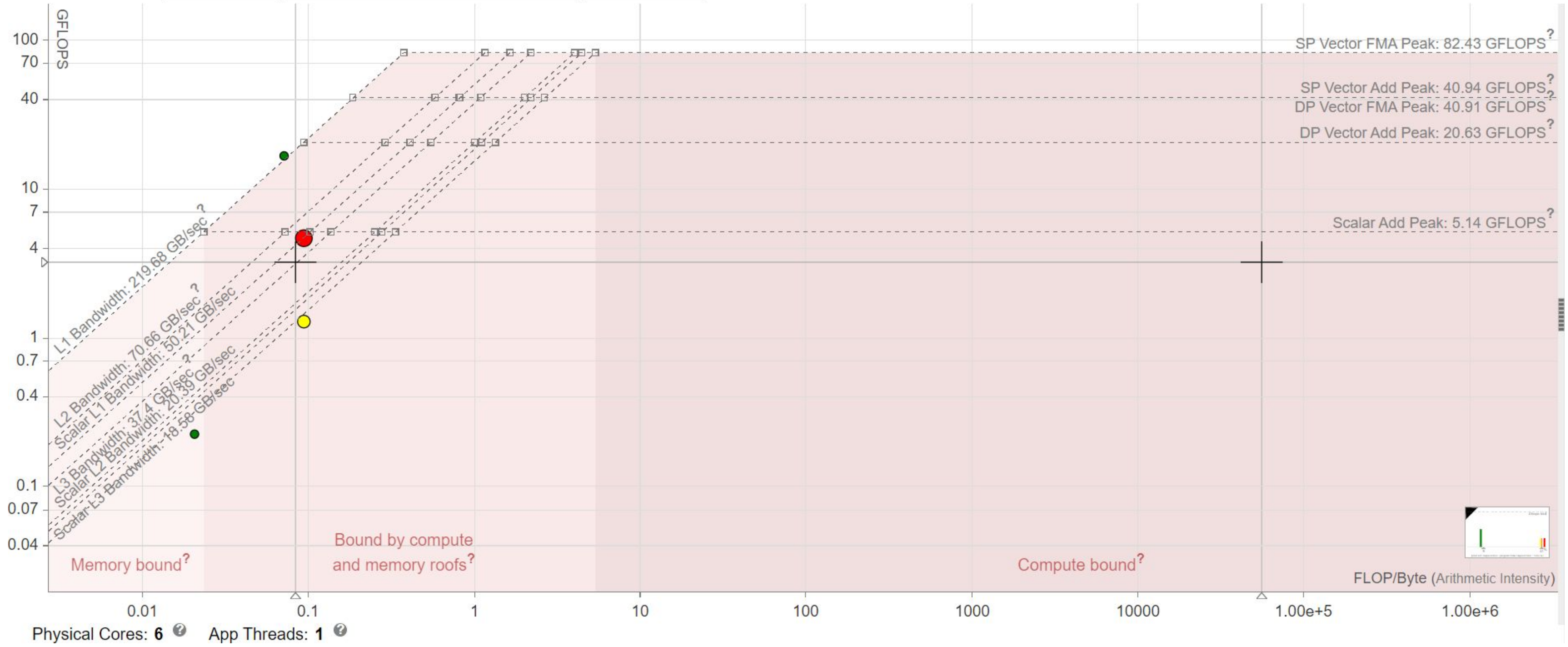


Vectorization

Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
L1d cache: 256 KiB
L1i cache: 128 KiB
L2 cache: 1.5 MiB
L3 cache: 12 MiB
Compiler: GCC 13.1.1 Flags: -O3 -ffast-math -mfma



Roofline plot



Thank you! Questions?