



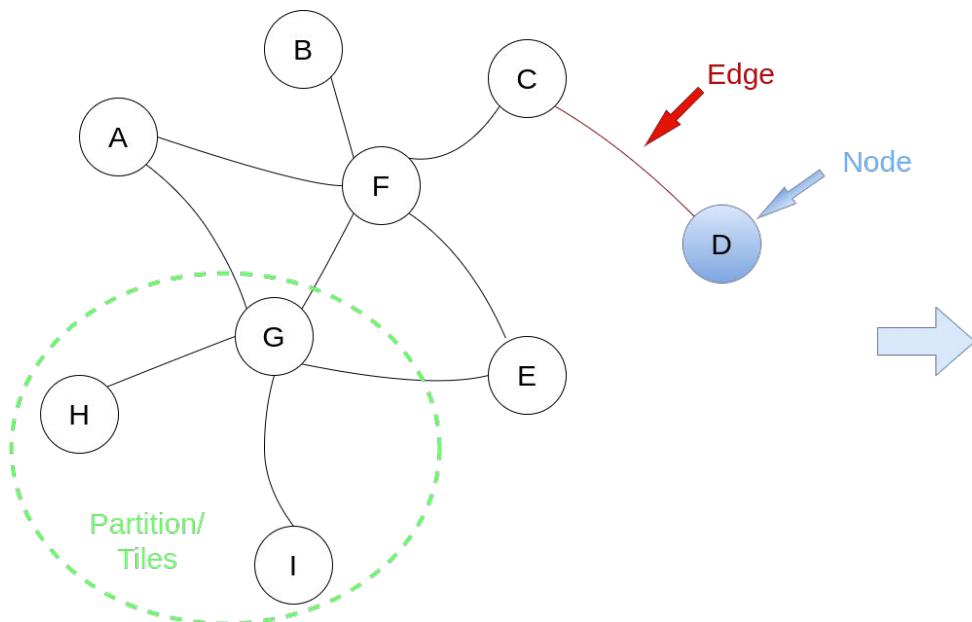
Parallel Graph Coloring in Shared Memory

Daniel Reith and Eric Kern

Outline

1. Terminology
2. Project Description
3. Implementation
 - a. Tiling
 - b. Collision Computation
4. Evaluation
 - a. Data Movement
 - b. Distance 1
 - c. Distance 2

Terminology



A	B	C	D	E	F	G	H	I	
1	0	0	0	0	1	1	0	0	A
0	1	0	0	0	1	0	0	0	B
0	0	1	1	0	0	1	0	0	C
0	0	1	1	0	0	0	0	0	D
0	0	0	0	1	1	1	0	0	E
1	1	1	0	1	1	1	0	0	F
1	0	0	0	1	1	1	1	1	G
0	0	0	0	0	0	1	1	0	H
0	0	0	0	0	0	0	1	0	I

Project Description

- Max & Total collisions for Distance 1 & Distance 2 coloring
- Different hash functions for coloring
- Performance & Quality evaluation
- Comparison with cusparseDcsrcolor() and old ASC project

Tiling

- No need to access elements in different partitions
- Permutation not needed
- Only row pointer important
- Sequential
 - Simply create partitions with constraint on maximum size

How size of partition is calculated may differ

Row_Ptr
0
5
9
13
16
20
...

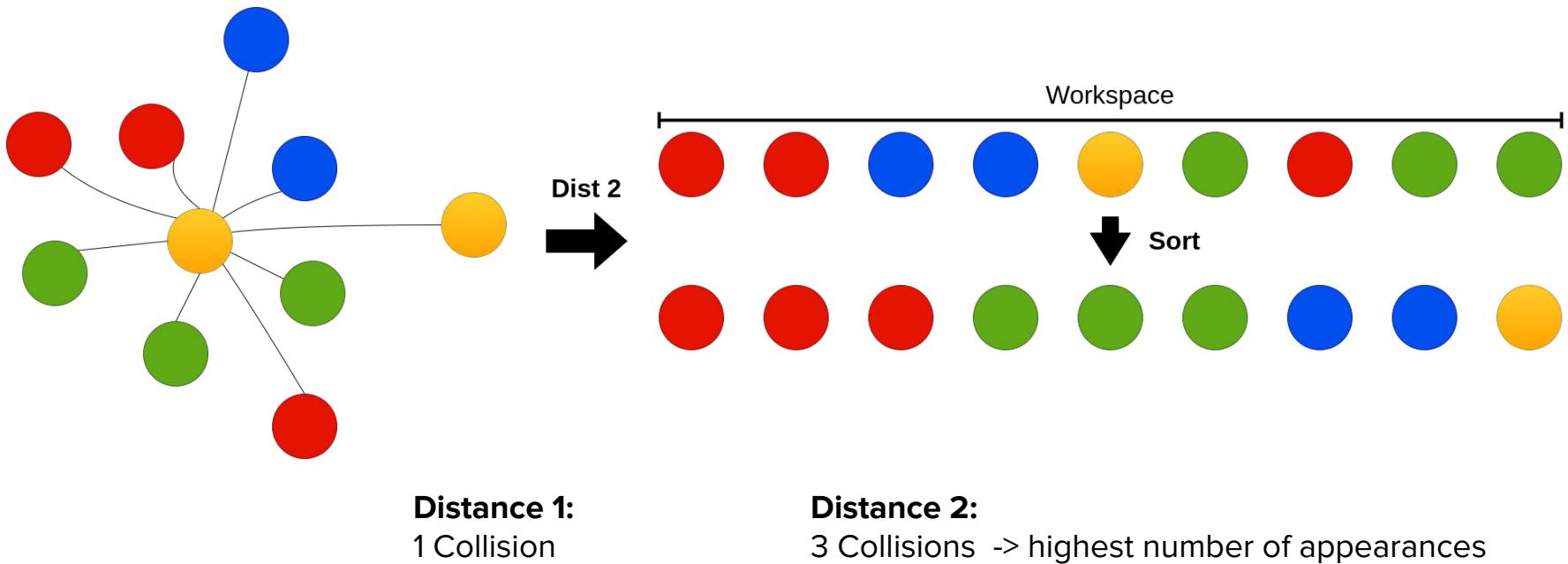
Adjacent_Difference
0
5
4
4
3
4
...

```
tile_start ← 0
tile_cols ← 0
i ← 1
while i < len(Row_Ptr) do
    tile_rows ← i - tile_start
    tile_cols ← tile_cols + Adjacent_Difference[i]

    tile_size ← foo(tile_rows, tile_cols)

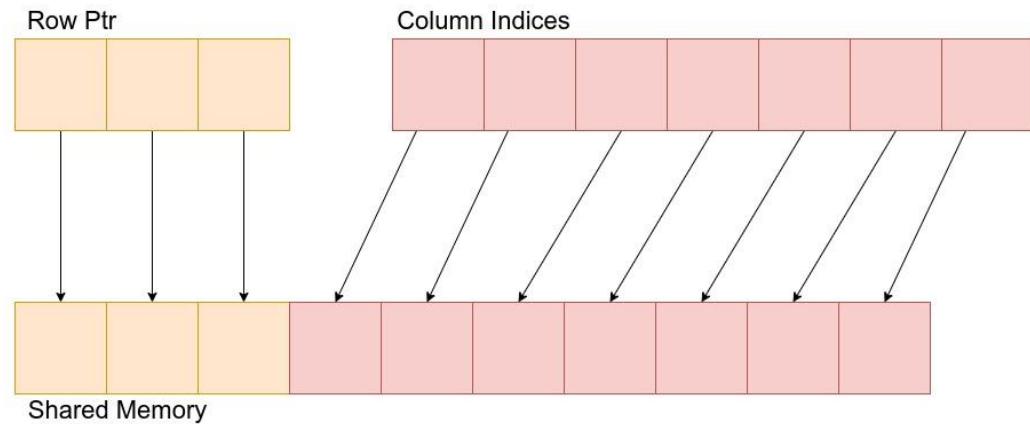
    if tile_size > shared_mem_size then
        set_tile_start(i);
        tile_cols ← 0
    end if
    i ← i + 1
end while
```

Collision Counting



Shared Memory Load

- Coalesced load of both Row Ptr and Column Indices
- Possibly unaligned
- Barrier sync. after shared memory load



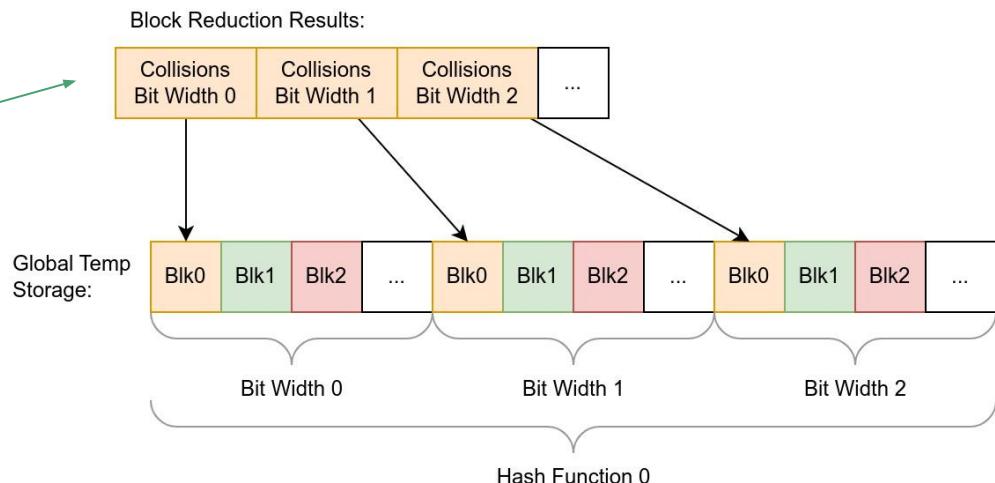
→ No overlap of load and compute inside a single thread block

Distance 1 Kernel

Algorithm 1 Distance 1 Kernel

```
PartitionToSharedMemory(tile)
for hash_functions do
    ComputeLocalCollisions(tile)
    for bit_widths do
        BlockReduceTotal(local_collisions)
        BlockReduceMax(local_max_colisions)
        if threadID == 0 then
            global_block_total <- block_total
            global_block_max <- block_max
        end if
    end for
end for
threadfence()
if Thread == 0 then
    CountTickets()
end if
if last_block then
    ReduceAllBlocks()
end if
```

$2 * \text{hash_functions} * \text{bit_widths}$ Counters
→ 300 in our case



Hardware

RTX 2080 TI

Memory Size:	11 GB
Memory Bandwidth:	616 GB/s
Max Shared Memory per SM:	64 KB
Max Register Memory per SM:	256 KB
Max Threads per SM:	1024
Streaming Multiprocessors (SM):	68
Compute Capability	7.5



Evaluation

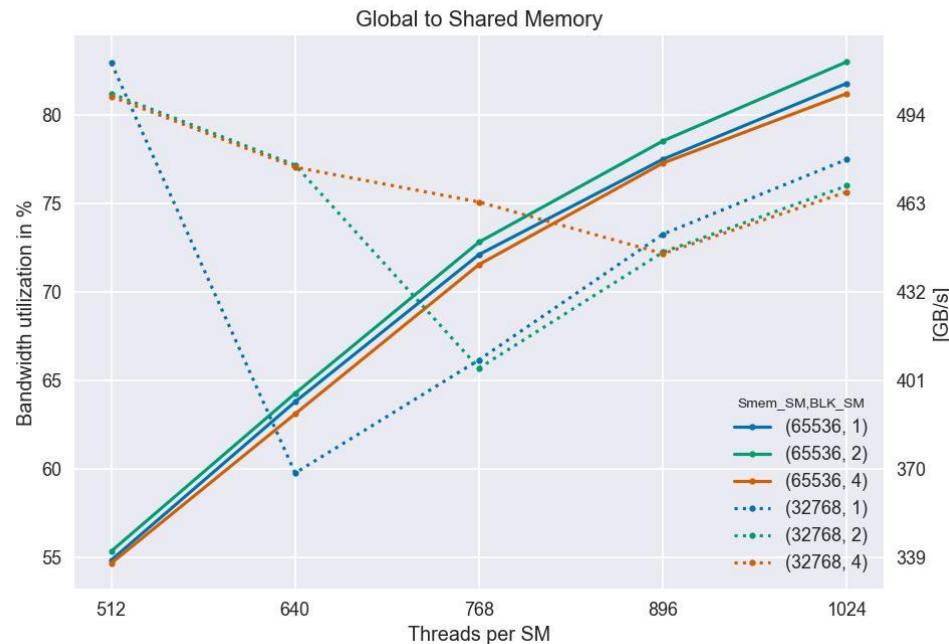
- We use bandwidth as metric
 - Reference to HW peak
 - Comparable
- Speedup in comparisons
- Performance might vary with other matrices

Janna/Cube_Coup_dt0	
Rows:	2 164 760 (8 MiB)
NonZero Entries:	127 206 144 (485 MiB)
Avg Node degree:	59
Max Node degree:	68

Global to Shared Memory

Launch Bounds Configuration:

- Blocks per SM is **desired lower limit** (can be more)
- We adjust threads per block so that desired blocks fit on SM
- More Threads are better
- 2 Blocks per SM seems best

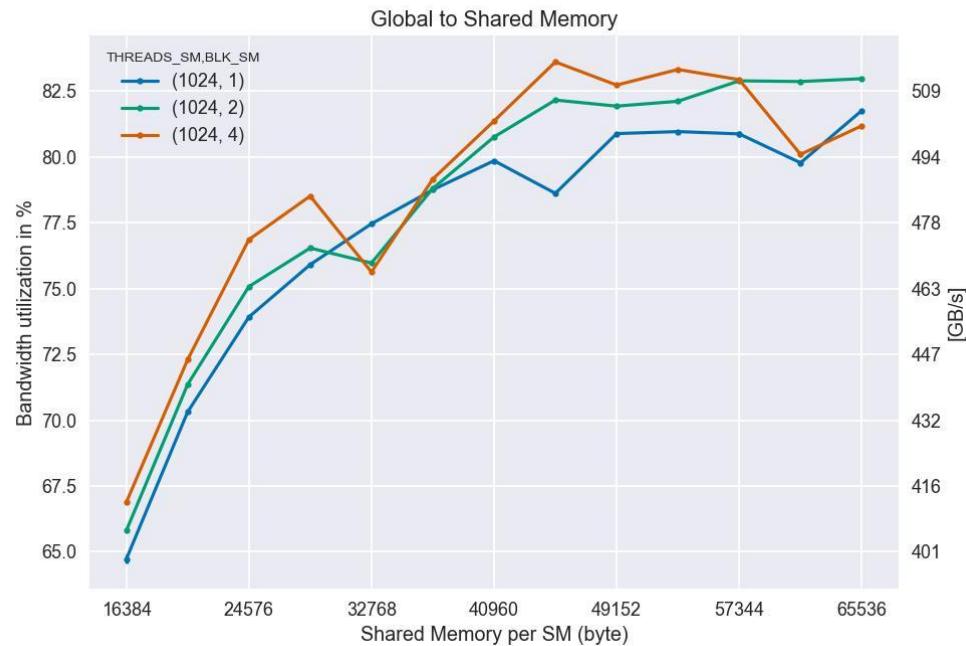


Global to Shared

Partition Size:

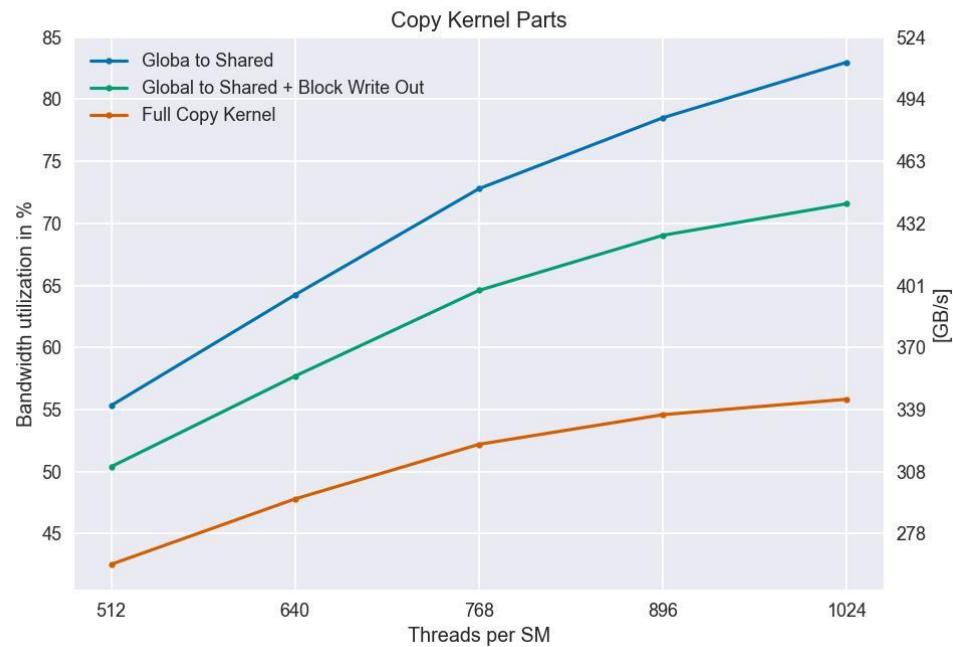
- Partitions below 48 KiB loses performance
- Above 48 KiB no big change
- Distance 2 coloring
→ only half shared memory for data

Less Shared Memory
↓
More Partitions
↓
More unaligned Memory Accesses



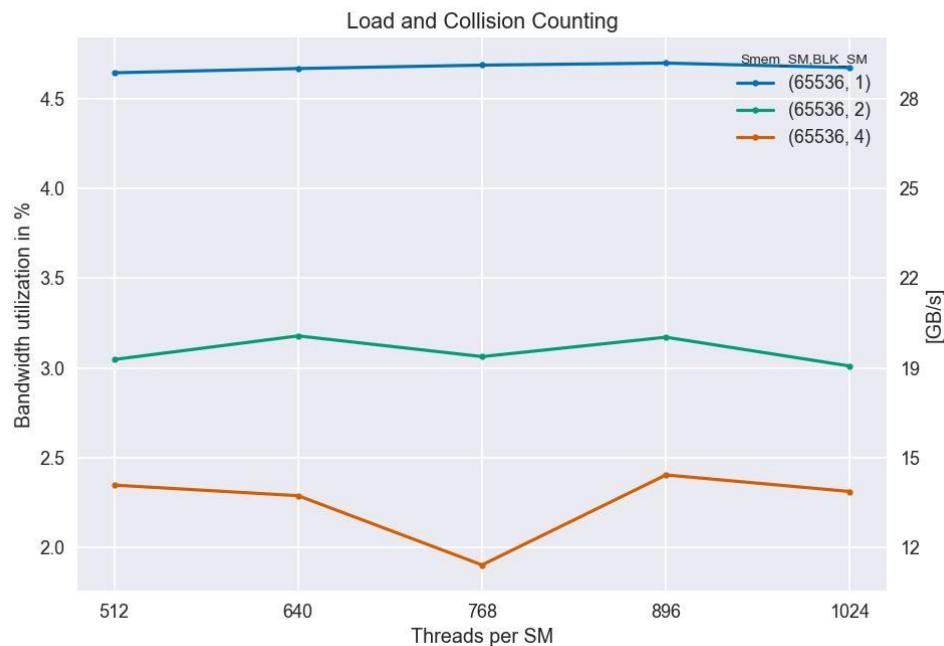
Copy Kernel Parts

We can achieve 56% of peak BW with the current global data movement



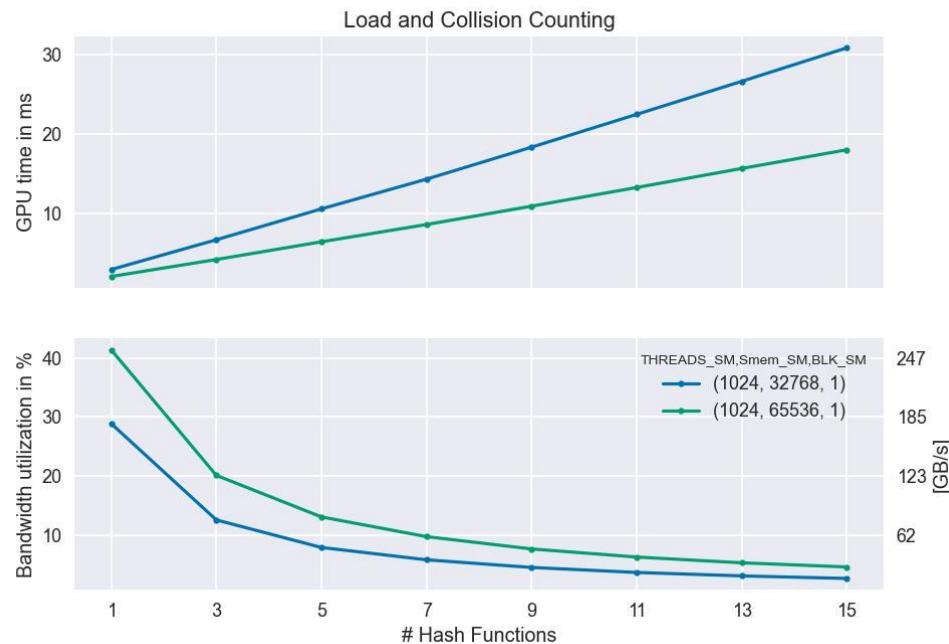
Load and Collision Counting

- Hashing significantly reduces Throughput



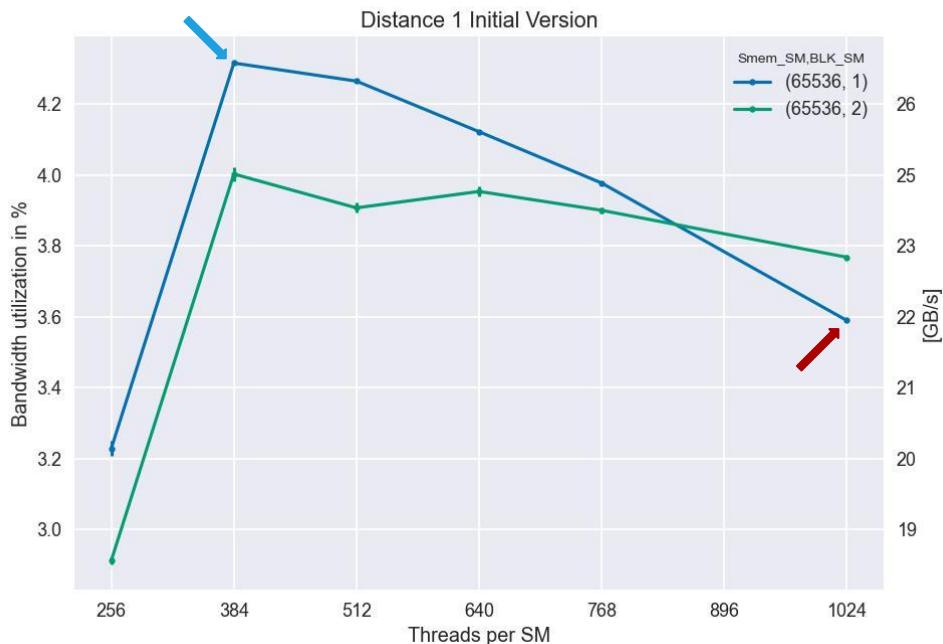
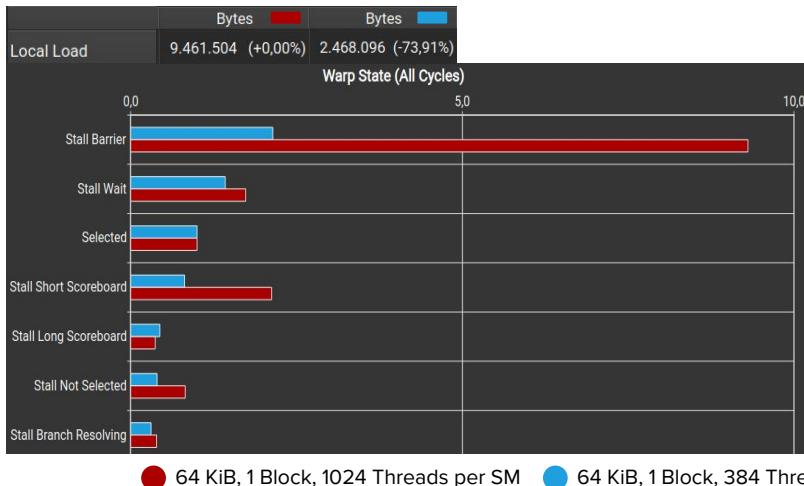
Load and Collision Counting

- We are latency-bound
- For 15 hash functions we spend only 4% of the time for loading data to shared memory



Distance 1 Initial Version

- Not enough parallelism for matrices with high node degree
- Only a bit register spilling with many threads

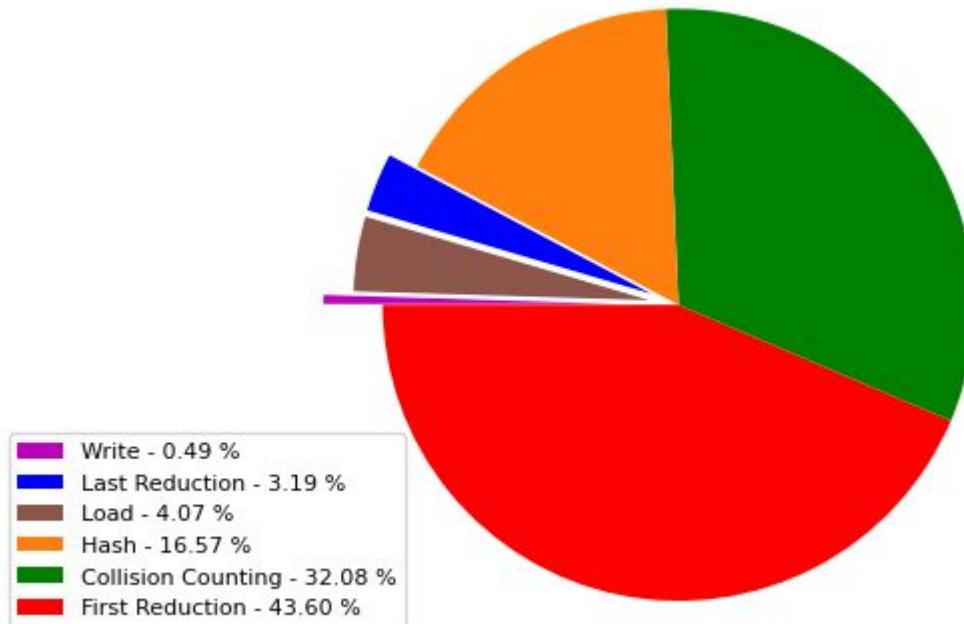


Relative Execution Time

Main contributors:

- Block reduction
- Collision computation

→ Focus of improvements

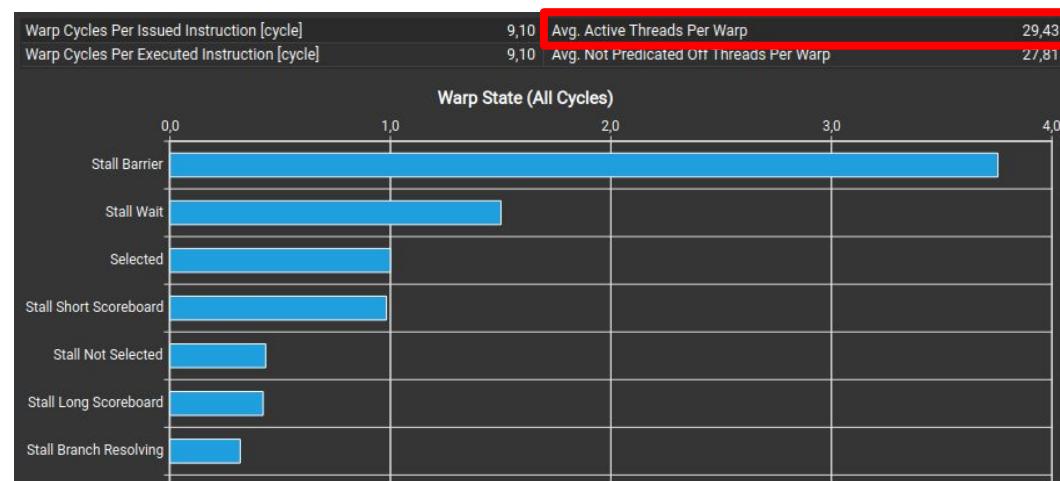


64 KiB, 1 Block, 384 Threads per SM

Constraints on Performance

Possible problems during computation:

- Unnecessary computations (break loop, unrolling)
- Bank conflicts (read before hash)
- Too many synchronizations
- Other reduce algorithms
- Branch divergence

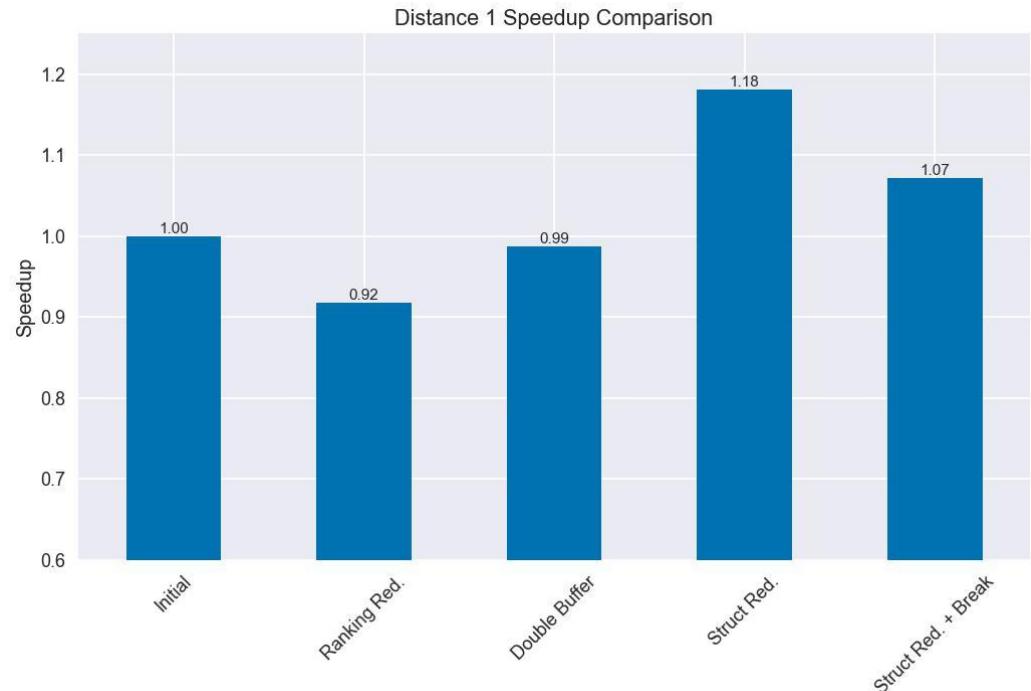


● 64 KiB, 1 Block, 384 Threads per SM

Speedup Comparison

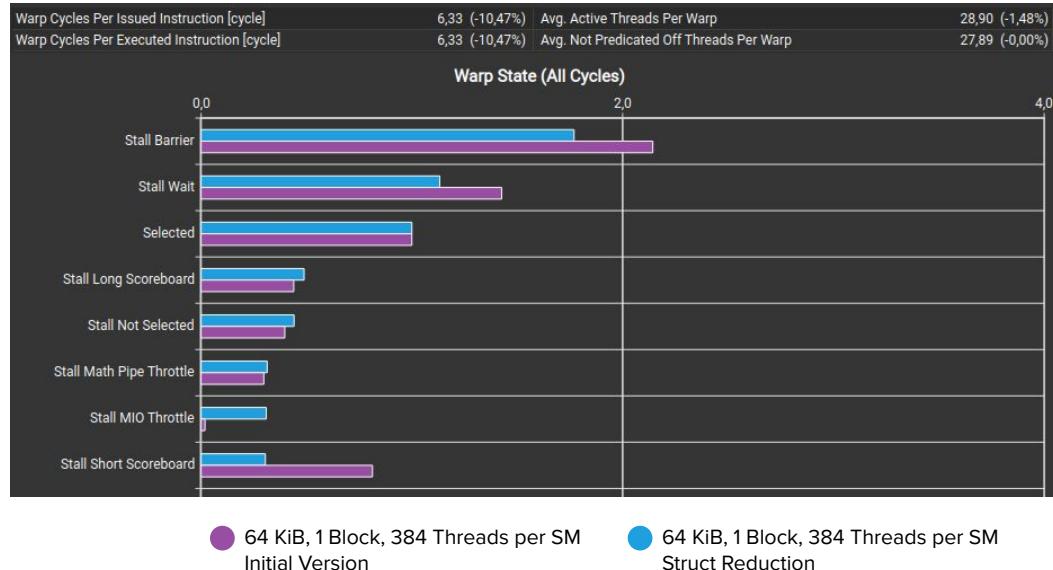
- Warp Reduction (default)
better than Ranking
Reduction Algorithm (cub)
- Struct reduction decreases
number of reduction calls
→ less synchronization
- Breaking loop causes
branch divergence

64 KiB, 1 Block, 384 Threads per SM



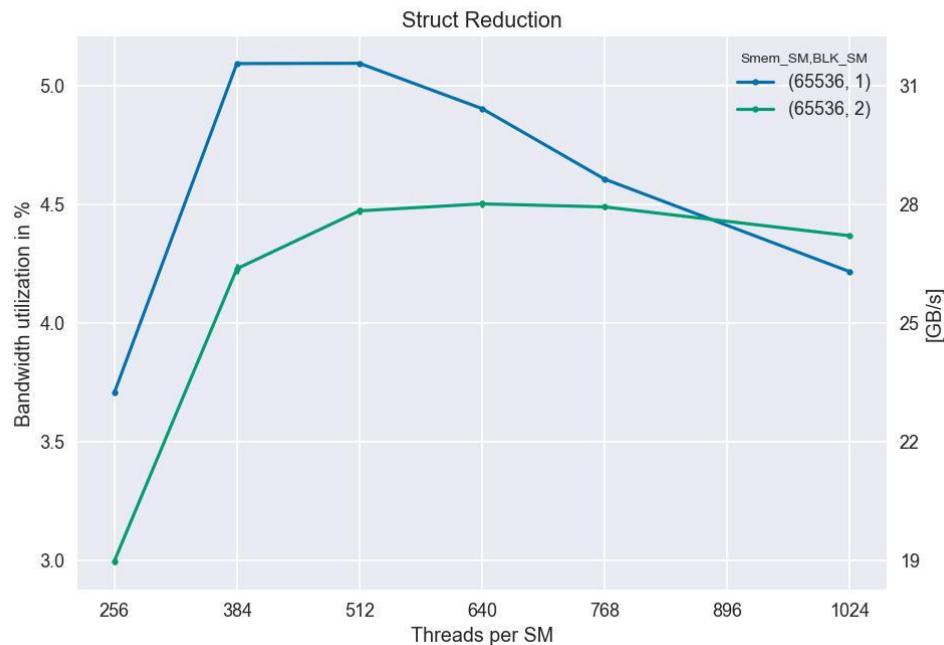
Distance 1 with Struct Reduction

- More work in single reduction
 - More shared memory in single reduction (less reuse)
- Less stalls caused by barrier



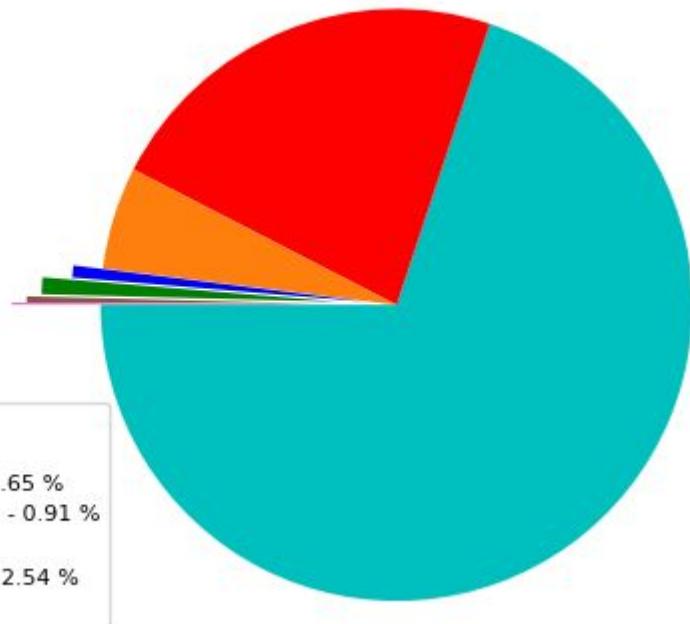
Best Version Launch Config

- launch configuration behavior similar to previous version



Relative Execution Time Distance 2

- Workspace sorting main contributor
 - main focus
- Reduction also big impact

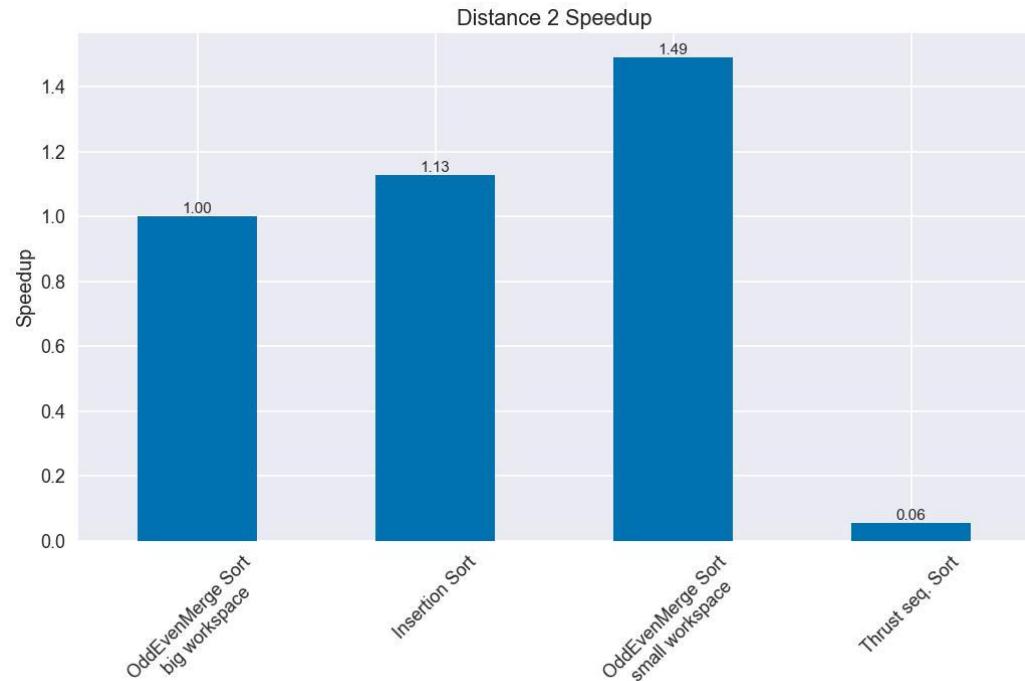


64 KiB, 2 Blocks, 384 Threads per SM

Distance 2 Speedup

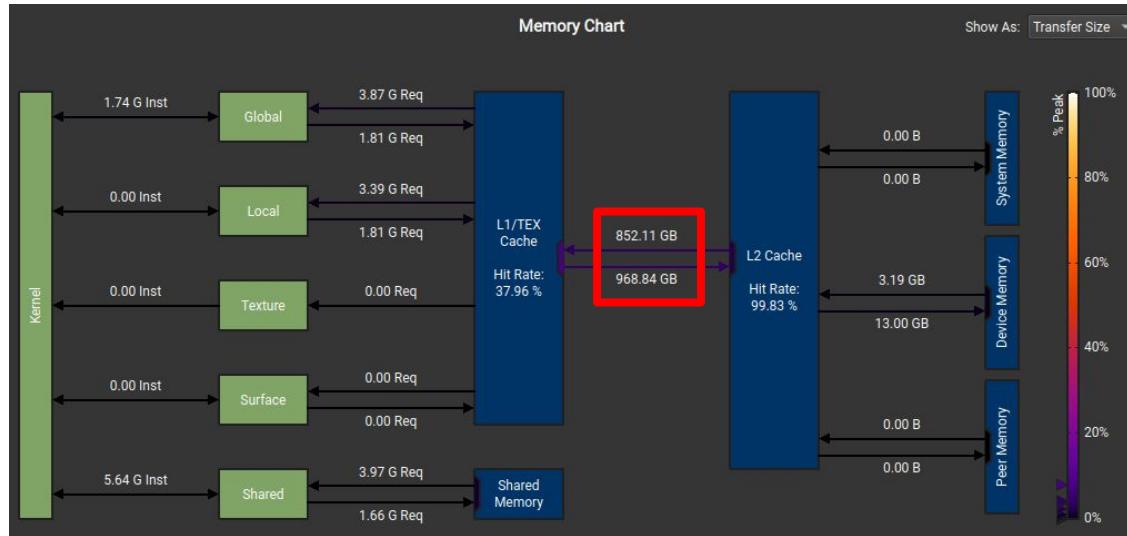
64 KiB, 1 Blocks, 384 Threads per SM

- Big Workspace:
 - round up to multiple of $32 + 1$
- But: makes partition smaller
- Sorting Network still useful
 - Depending on matrix



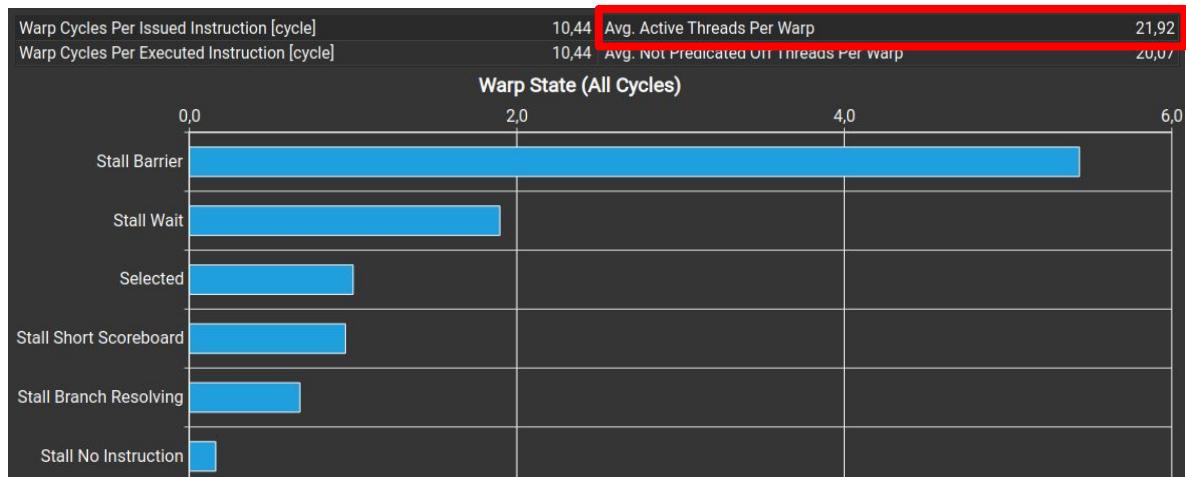
Thrust seq Sort

→ global memory is used for temporary sorting storage



Profile Distance 2 Optimized

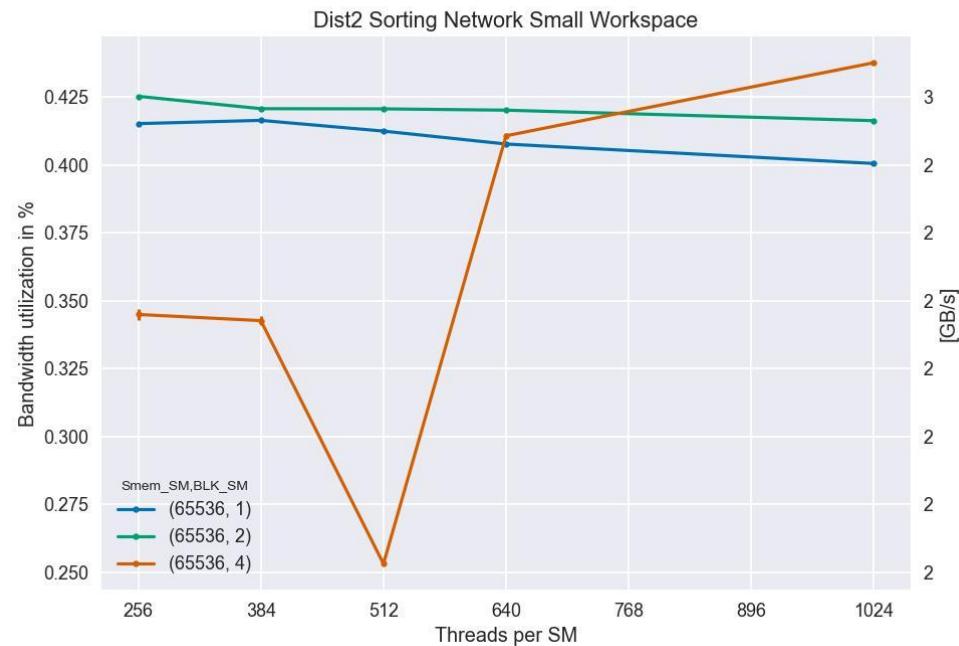
- Barrier still main stall reason
- Branch divergence



64 KiB, 1 Block, 384 Threads per SM

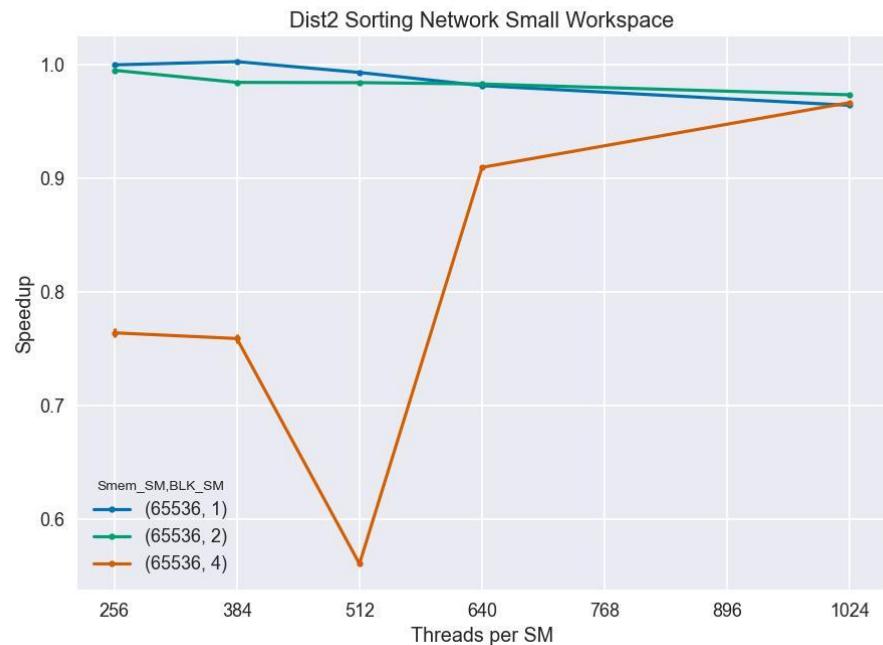
Distance 2 Launch Config

- Best bandwidth for 4 block only because of more memory access
- Actual execution time is not the best for 4 blocks



Distance 2 Launch Config

- Speedup plot with 1 block and 256 threads as reference shows actual results

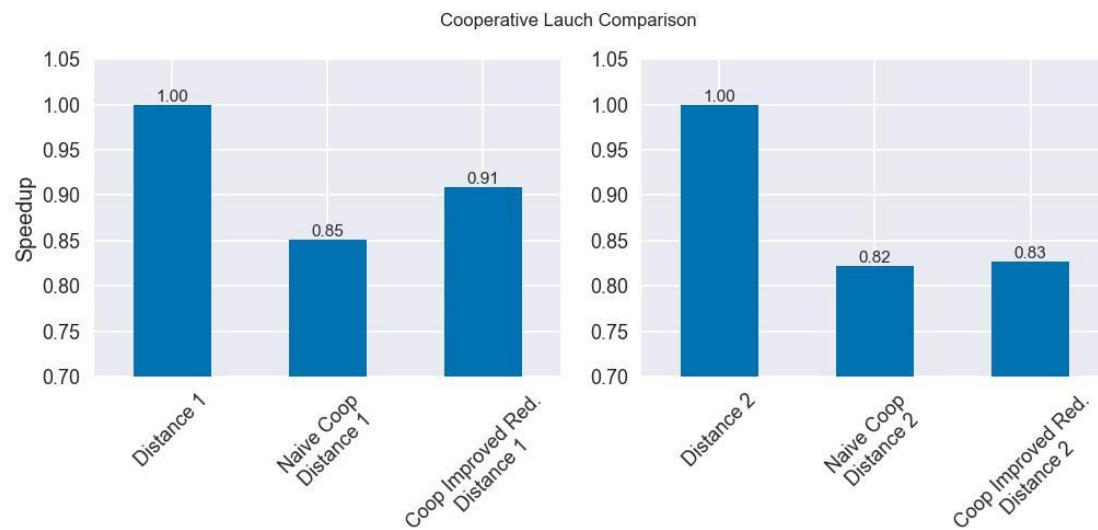


Cooperative Launch Comparison

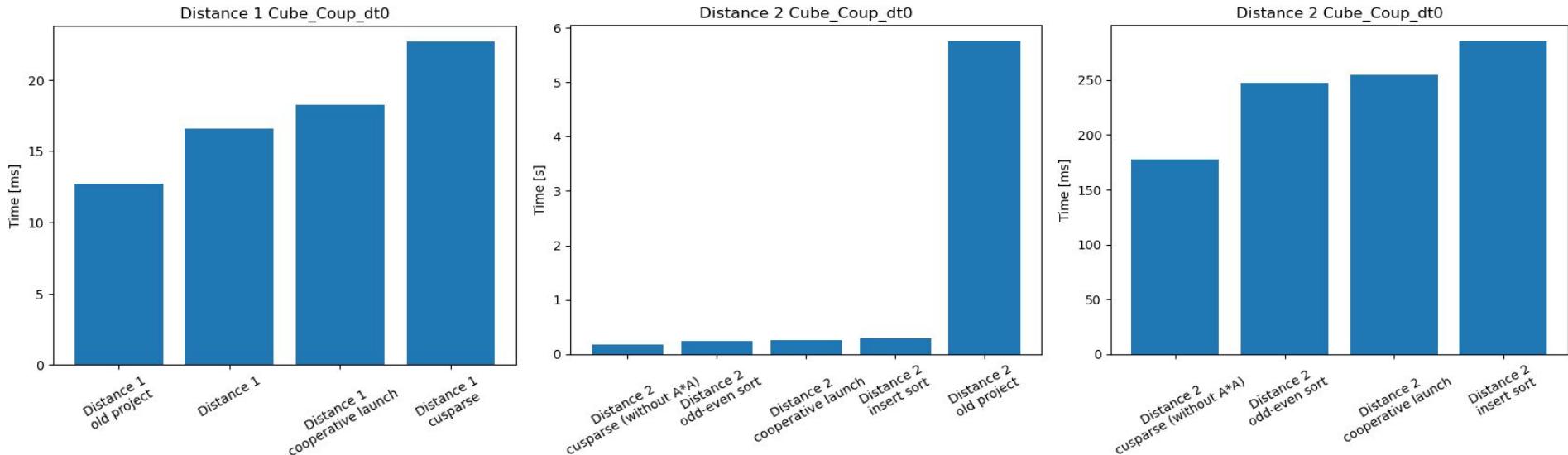
- Reduction of multiple partitions in single block can save global stores
→ Improvements for Dist 1

→ Overall performance depending on matrix

64 KiB, 1 Blocks, 384 Threads per SM

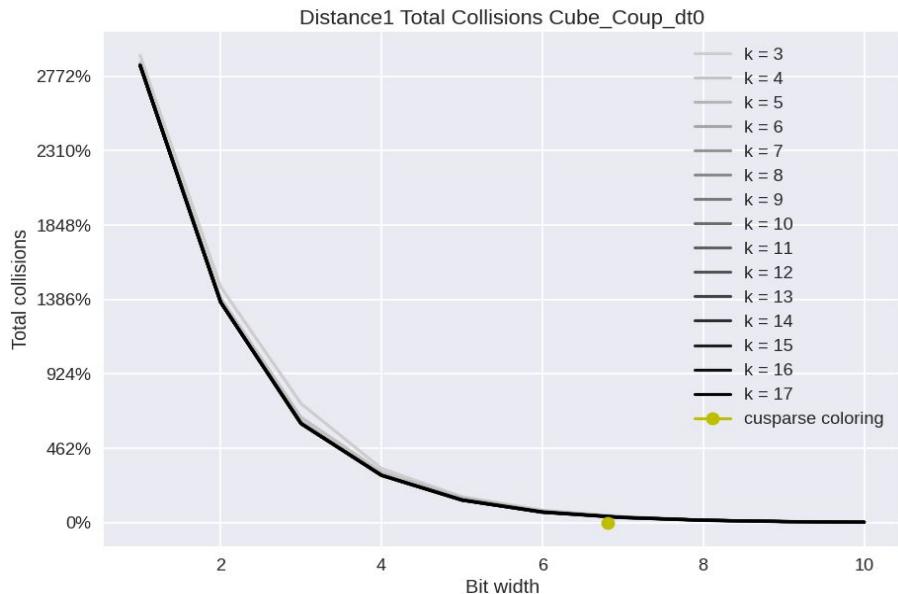


Best Version comparison

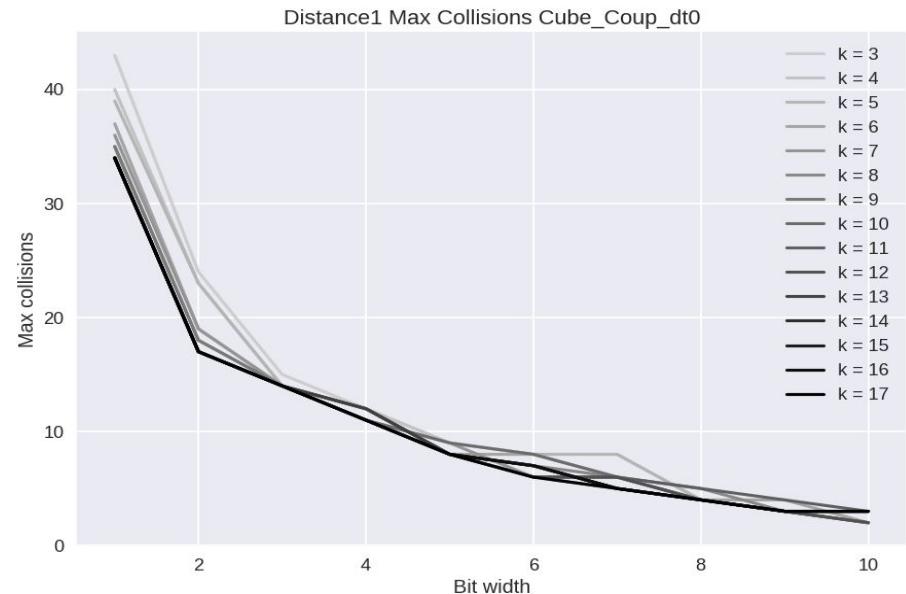


64 KiB, 1 Blocks, 384 Threads per SM

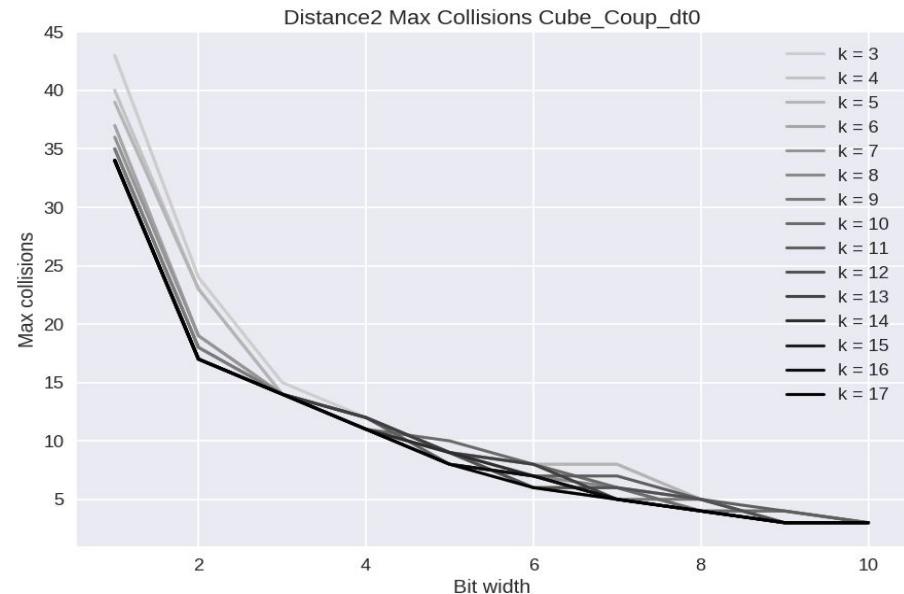
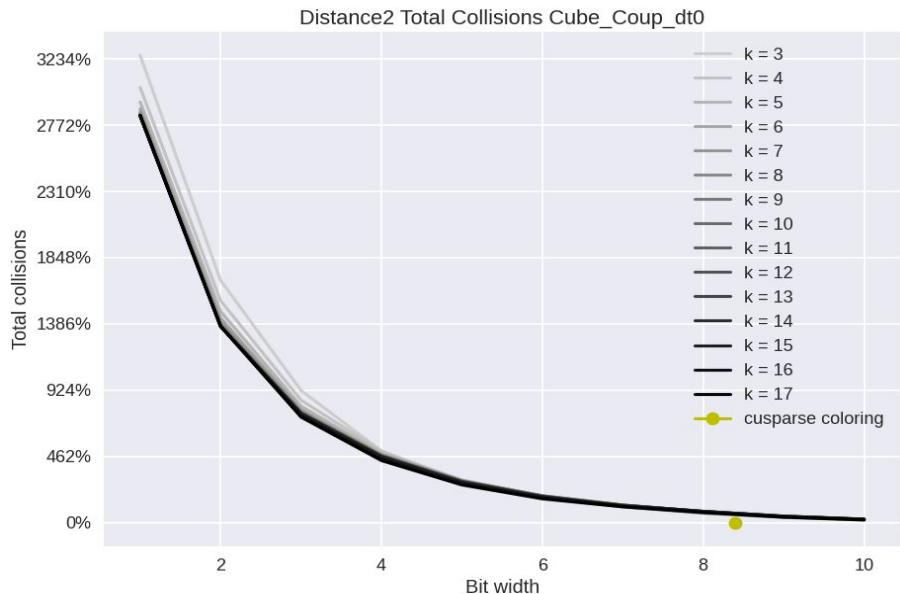
Coloring Quality Distance 1



- old project: 32768 colors ($k = 5$)



Coloring Quality Distance 2



- old project: 65536 colors ($k = 5$)

Summary

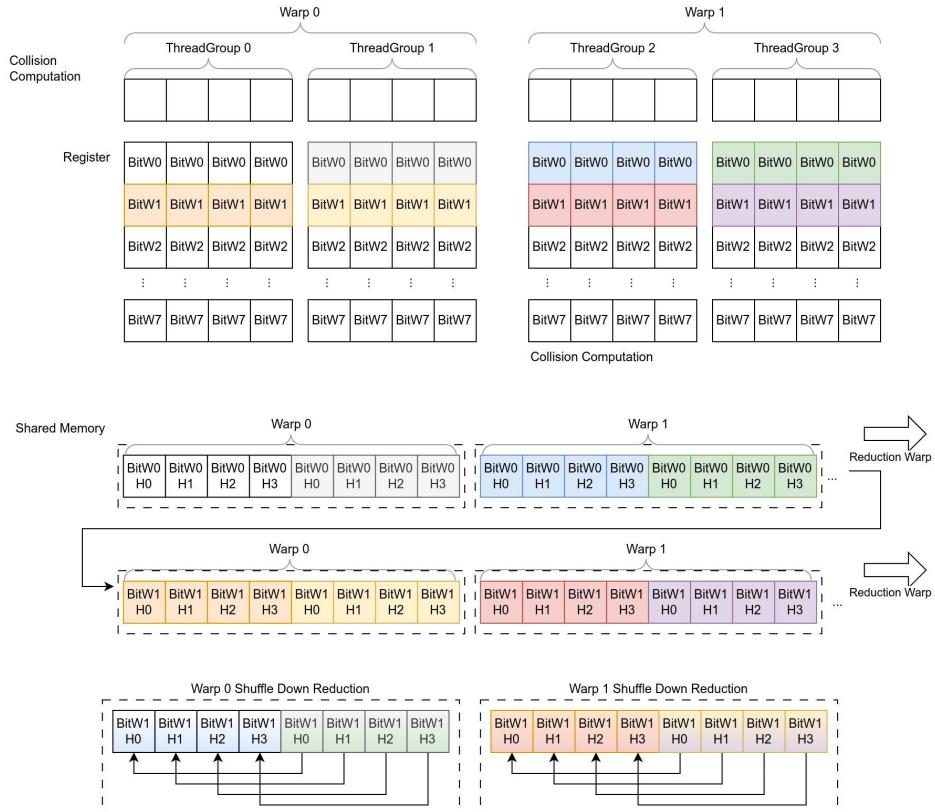
- Collision computation can happen completely in shared memory
- Contrary to our expectations: compute bound
- Time spent primarily in collision computation and reduction
- Shared Memory size can limit performance
- Bank conflicts not that problematic for sorting

Further Improvements

- Trade Computation for Communication in global Memory
 - Currently Global memory unused most of the time
- Think about using a whole Warp for a single Node
 - Too few rows in partition for graphs with high node degree
 - Might also help with branch divergence in very irregular graphs
- Use of better Hardware
 - More Shared Memory and more SMs

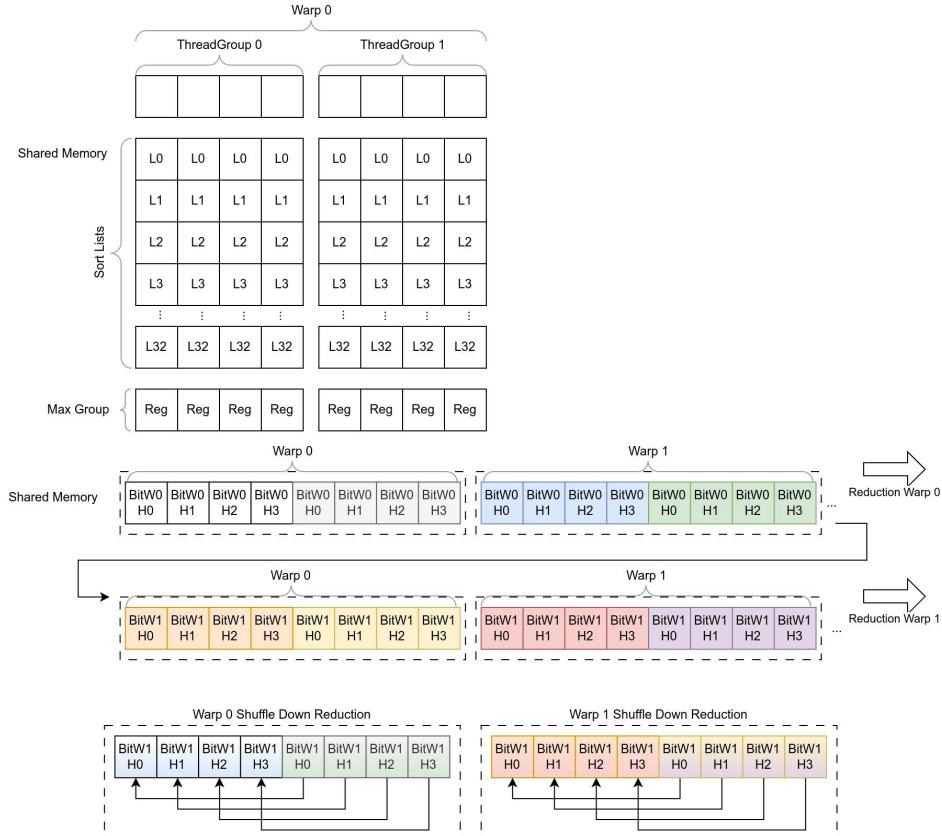
Thank you!

New Mapping of Warps/ThreadGroups to Nodes



- Thread group responsible for single node
- Each thread in thread group uses different hash function
- Reduction in shared memory with whole warp
- Finally the warp does a shuffle reduction to reduce the remaining thread group counters

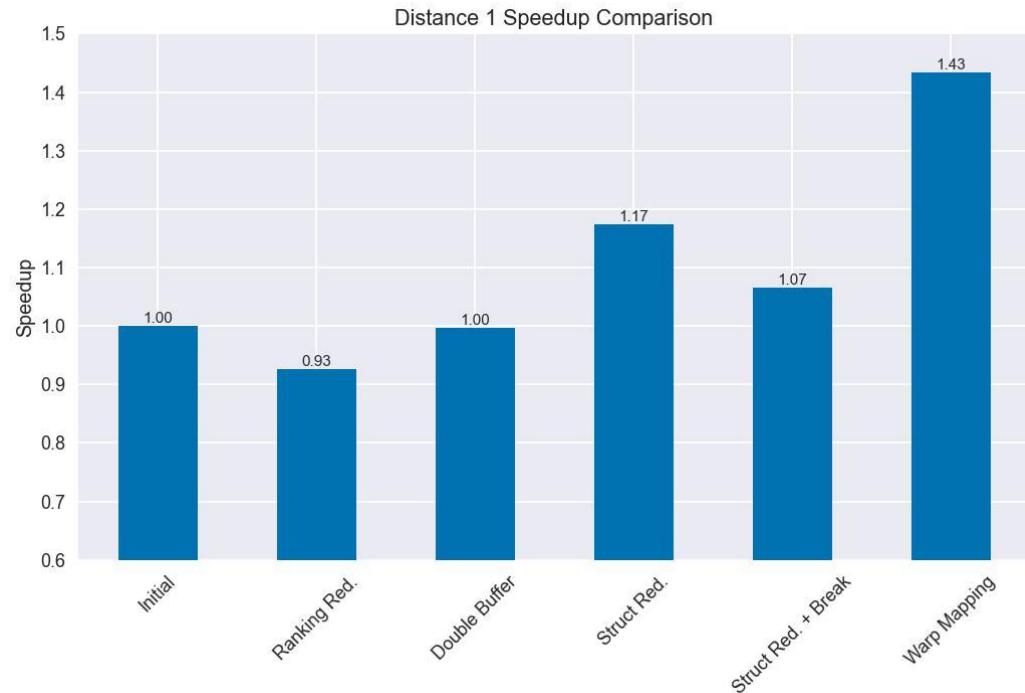
New Mapping of Warps/ThreadGroups to Nodes



- Shared memory consumption increases but allows fast parallel reduction
- Bank conflict potential increases because threads in thread groups use workspaces with exact same size
- Additional shared memory consumption will further reduce available space for partition data (potential problem for distance 2)

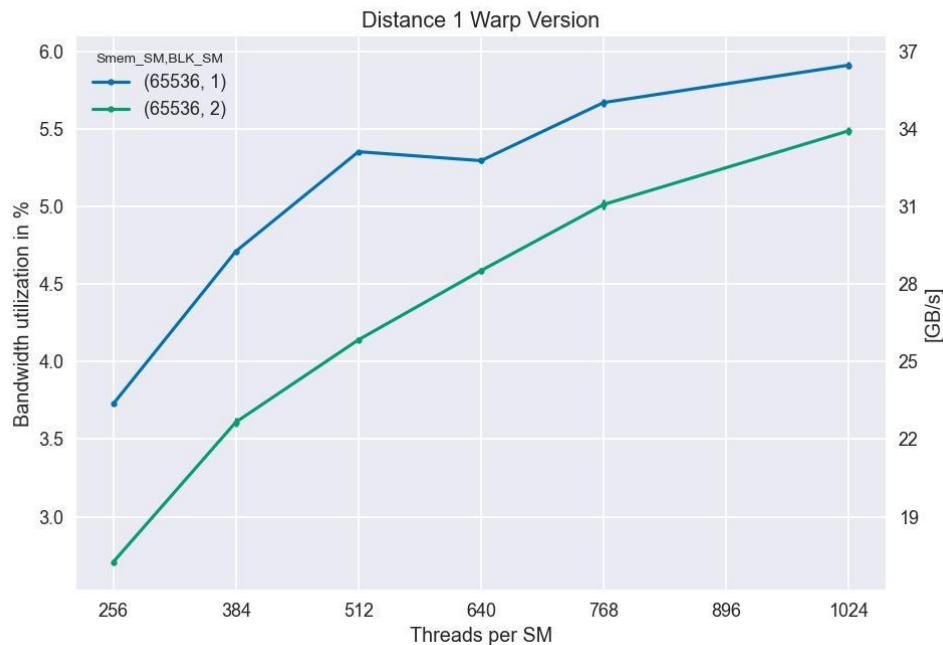
Speedup Comparison Distance 1

- Warp Version makes better use of available concurrency
- Better use of shared memory



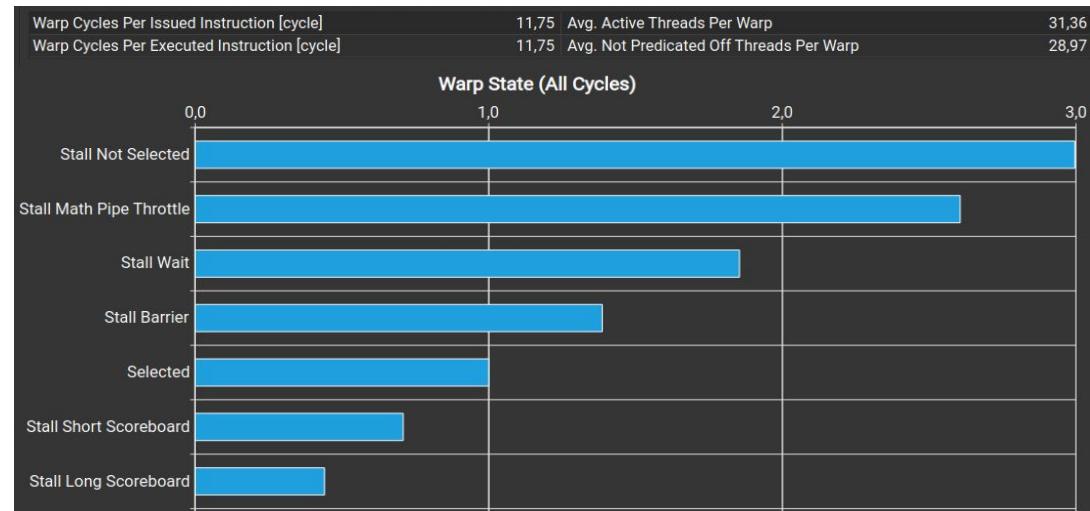
Warp Version Distance 1 Launch Config

- Now using all available threads yields the best performance
- New algorithm can make use of available compute resources

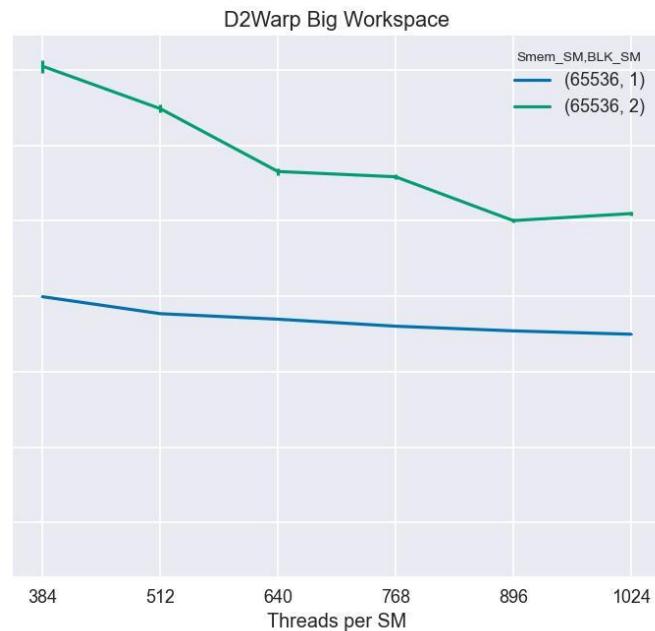
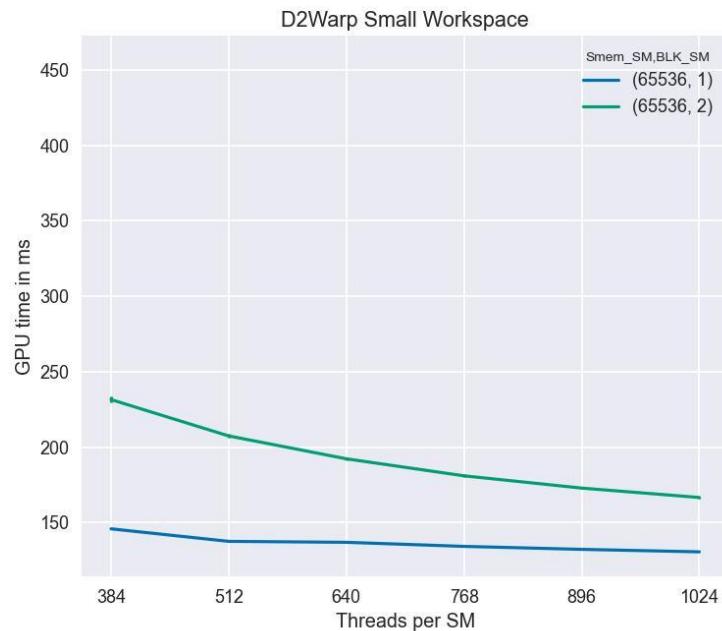


Distance 1 Warp Version Profil

- New kernel is now actually compute bound
- Primary contributor to “Stall Wait” is sequential instruction dependency in hash computation



Distance 2 Warp Launch Config



Distance 2 Warp Version Speedup

- It is actually worth to trade memory space for bank conflicts during sort

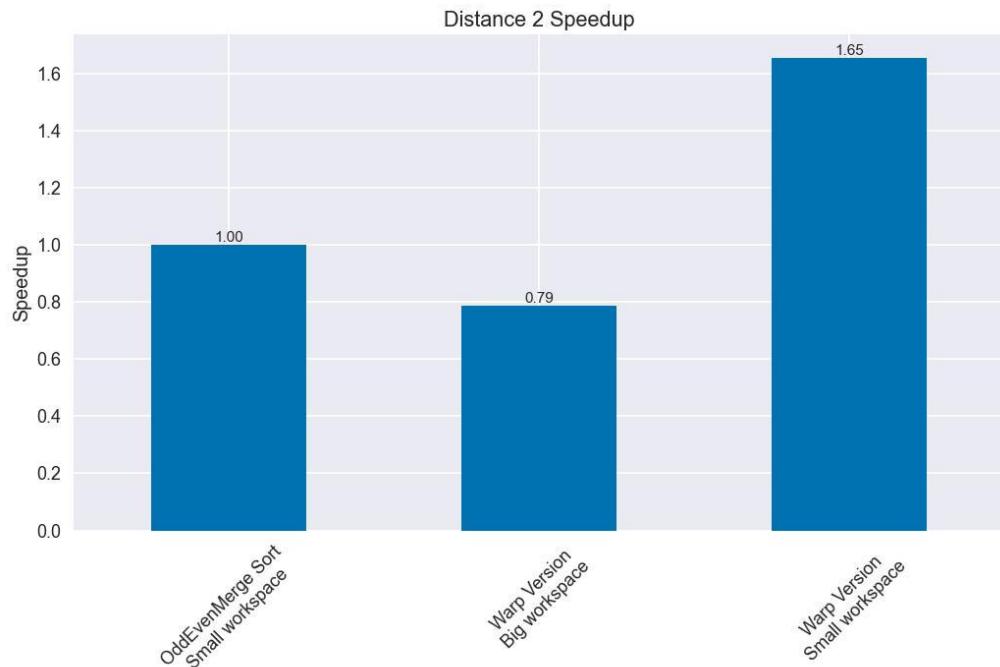
Reminder

Big workspace:

Bank conflict free. Workspace is multiple of $32 + 1$

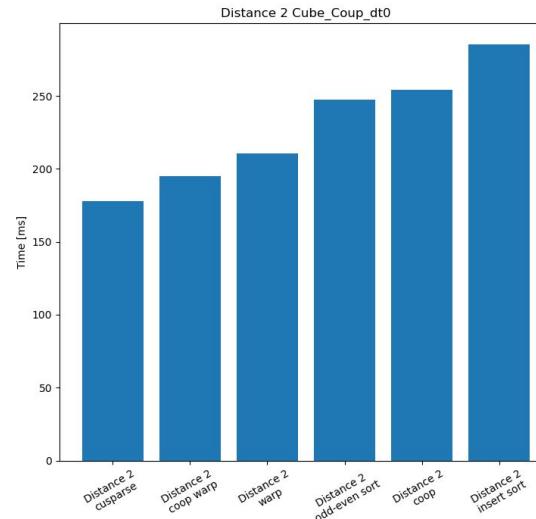
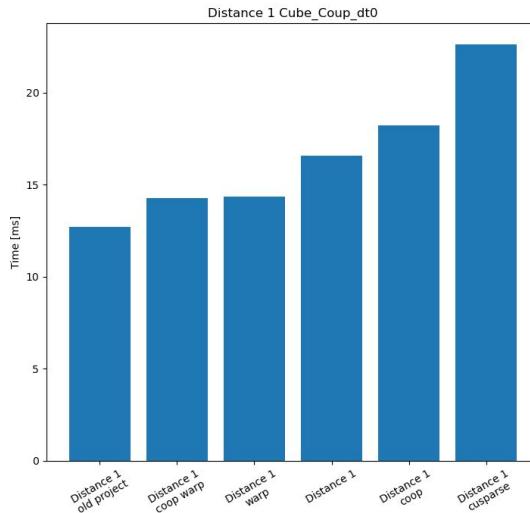
Small work space:

No round up to 32



(Warp versions also use OddEvenMerge SortNet)

Cooperative Launch Comparison



- With new algorithm and contiguous partitions mapped to the same block cooperative launch is faster

Shared Memory Distribution Warp Versions

Accepting Bank Conflicts

	D1Warp		D2Warp		D2Warp Small Workspace	
Matrix	CurlCurl_4	Cube_Coup_dt0	CurlCurl_4	Cube_Coup_dt0	CurlCurl_4	Cube_Coup_dt0
Collision Counter	45.4 kB	20.8 kB	3.5 kB	1.1 kB	8.5 kB	3.1 kB
Workspace	-	-	58.0 kB	54.6 kB	51.1 kB	57.4 kB
Tile Rows + Col Idx	17.4 kB	42.1kB	1.3 kB	2.4 kB	3.2 kB	3.6 kB
(Rows + Col Idx) / MaxSmem	27%	66%	2%	4%	5%	6%
Tiles	6 542	12 396	85 019	240 529	35 649	142 001

Shared Memory Distribution Initial Versions

Accepting Bank Conflicts

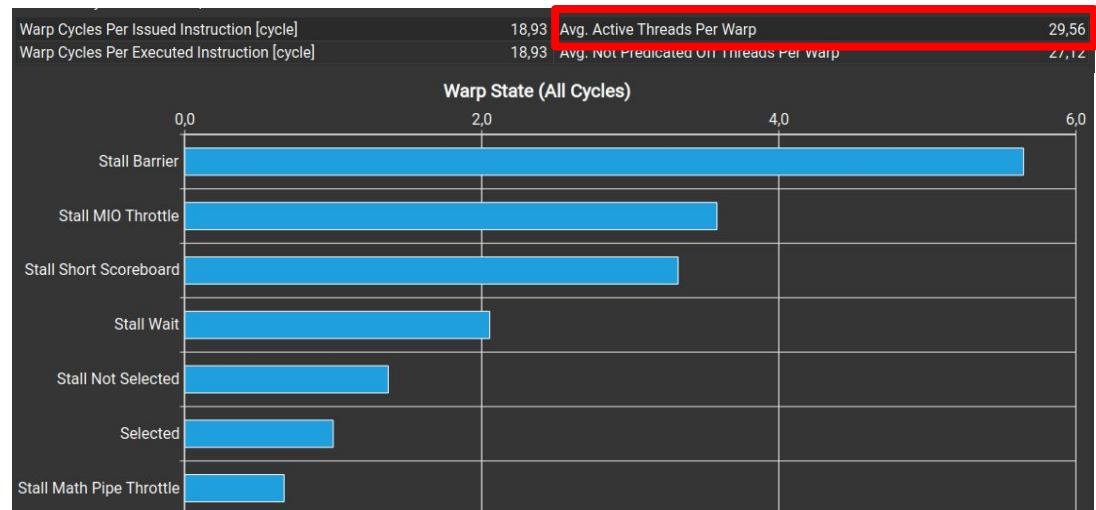


	D1 Struct Reduce		D2 Big Workspace		D2 Small Workspace	
Matrix	CurlCurl_4	Cube_Coup_dt0	CurlCurl_4	Cube_Coup_dt0	CurlCurl_4	Cube_Coup_dt0
Collision Counter	0.9 kB	0.9 kB	0.9 kB	0.9 kB	0.9 kB	0.9 kB
Workspace	-	-	45.1 kB	36.8 kB	31.2 kB	31.2 kB
Tile Rows + Col Idx	62 kB	62 kB	16.8 kB	24.6 kB	31.2 kB	31.2 kB
(Rows + Col Idx) / MaxSmem	97%	97%	26%	38%	49%	49%
Tiles	1 824	8 178	6 785	21 503	3 621	16 215

Distance 2 Warp Version Profile

First version only 22 Thread active on average

- Bank conflicts now cause warp stalls (MIO Throttle and Short Scorboard)
- Biggest contributor to barrier stall is barrier after collision counting, before reduction in shared memory
→ big time difference for collision computation among threads
(worse for more irregular matrices)



Tail Effect

	D1Warp	D2Warp	D2Warp Small Workspace
Matrix	Cube_Coup_dt0		
Tiles	12 396	240 529	142 001
Wave/TotalTiles	0.5%	0.2%	0.5%

Time measure for cooperative launch measured with clock64()

$\frac{max - mean}{mean}$	6.5%	2.8%	25.8%
$\frac{min - mean}{mean}$	-10.1%	-5.3%	-17.5%

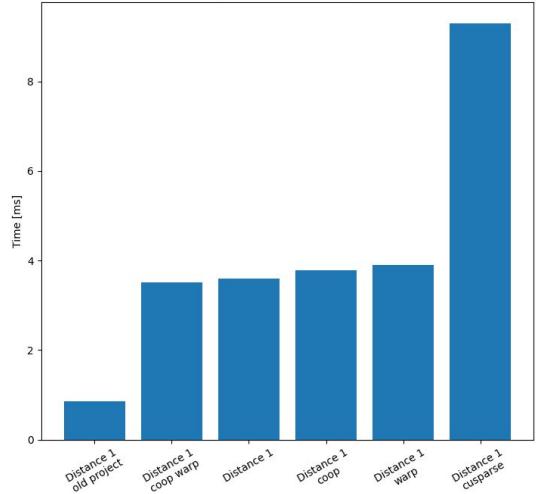
New Summary

- New problem mapping performs better
 - Distance1 much better use of shared memory
 - Distance2 better as well but with even more limitations on remaining shared memory size for actual input data (More shared memory most likely even more effective)
- Mapping of consecutive tiles to same SM in cooperative launch yields even better results than normal version (without cooperative launch)
- Tail effect is actually not a problem

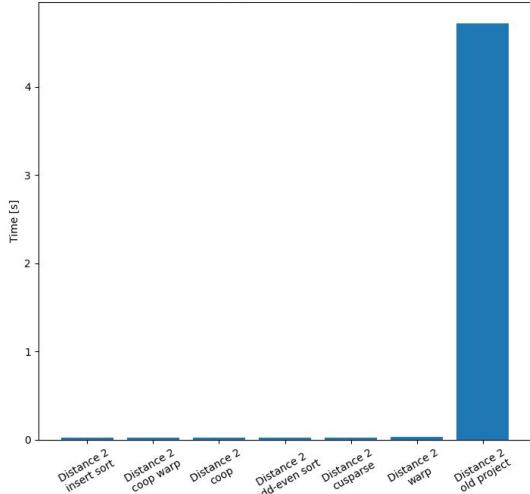
Remark on upcoming plots

- all following plots created with best performing setup
 - 1 Thread Block per SM & 384 Threads per Block for initial versions
 - 1 Thread Block per SM & 1024 Threads per Block for new warp versions
 - 64 kB Shared Memory for all versions
 - For each Matrix a comparison of execution time and coloring quality for distance 1 & 2 with `cusparseDcsrcolor()` and the old project is made
- Since the matrix multiplication required for Cusparse Distance 2 coloring was performed on the CPU, we did not include this in our measurements

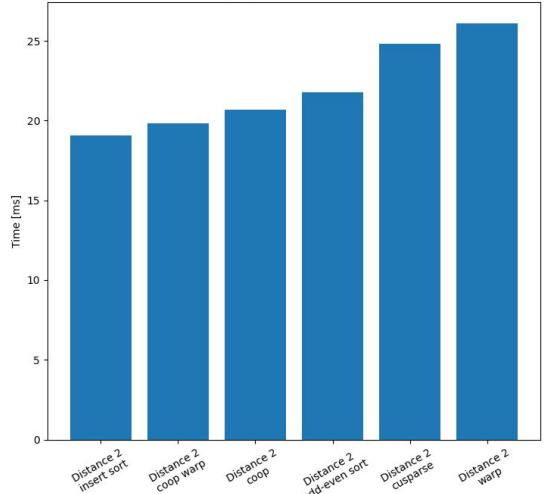
Distance 1 CurlCurl_4

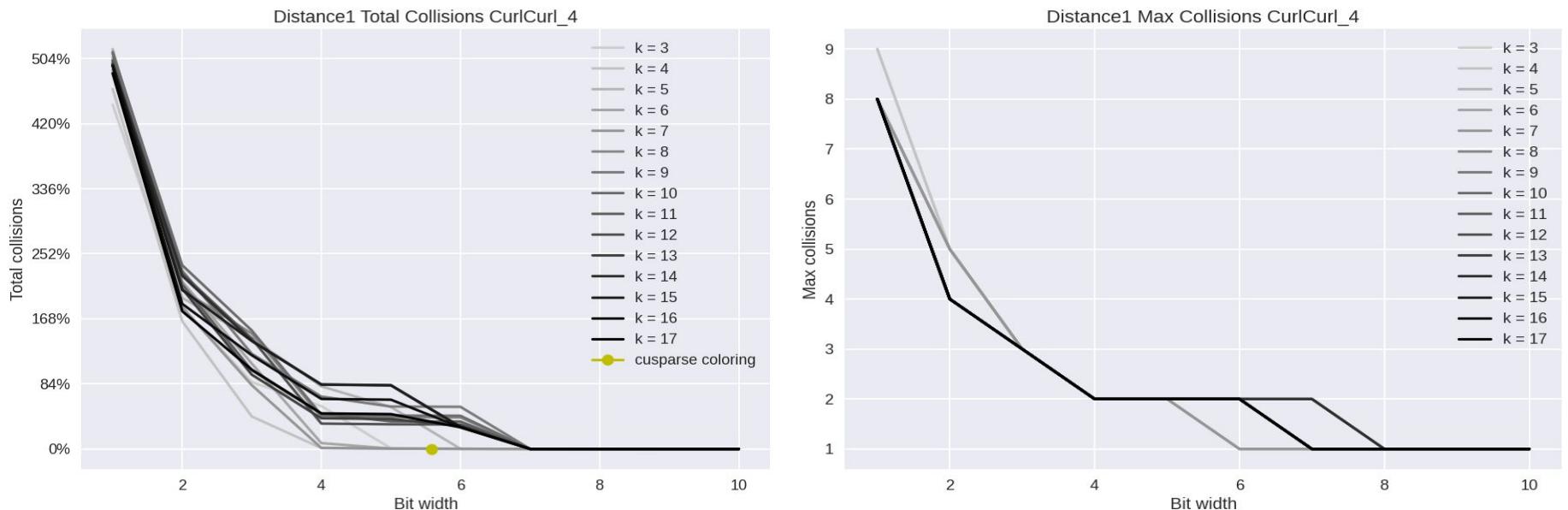


Distance 2 CurlCurl_4



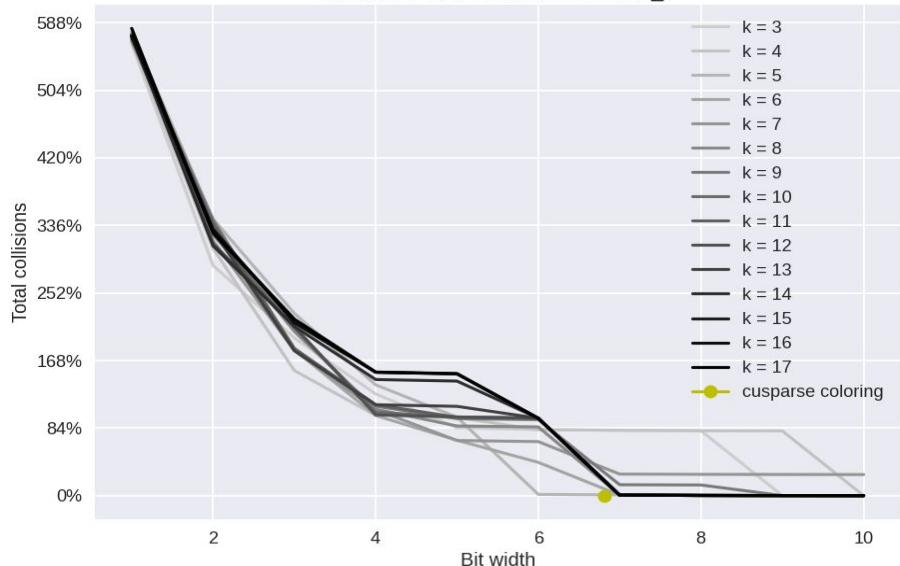
Distance 2 CurlCurl_4



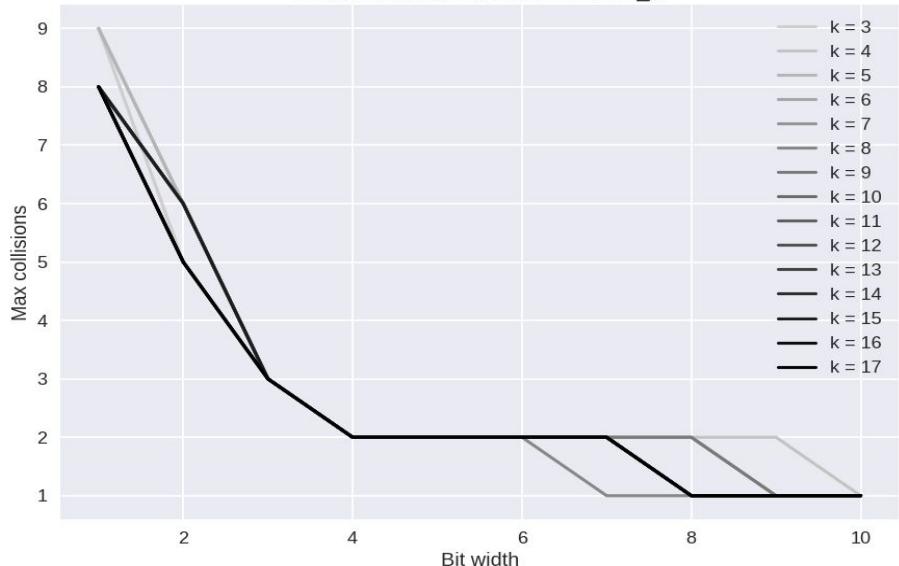


- old project: 65536 colors ($k = 3$)

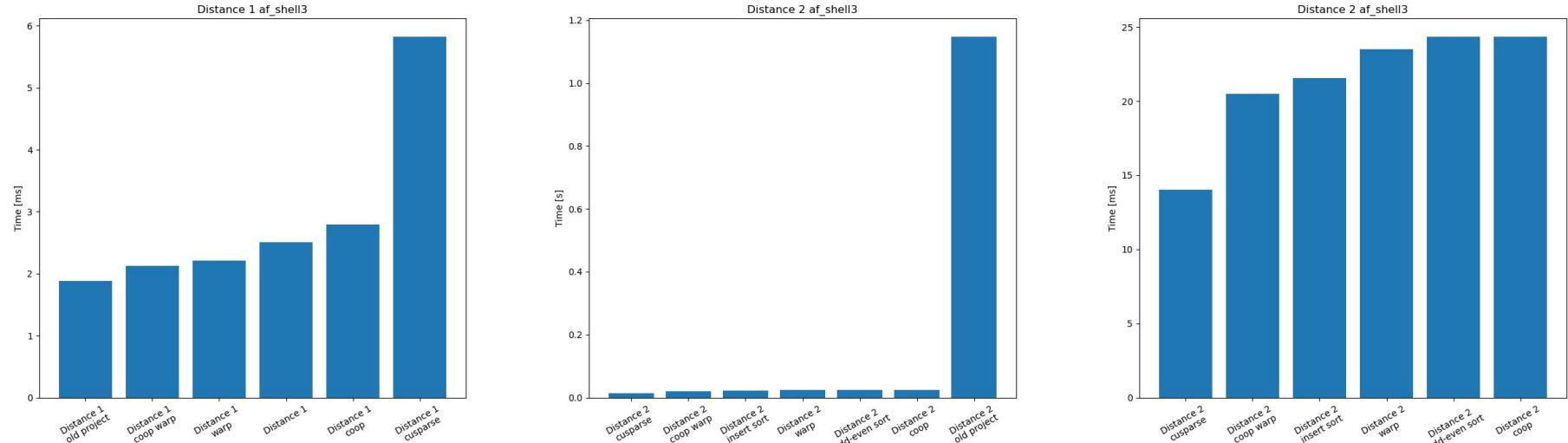
Distance2 Total Collisions CurlCurl_4



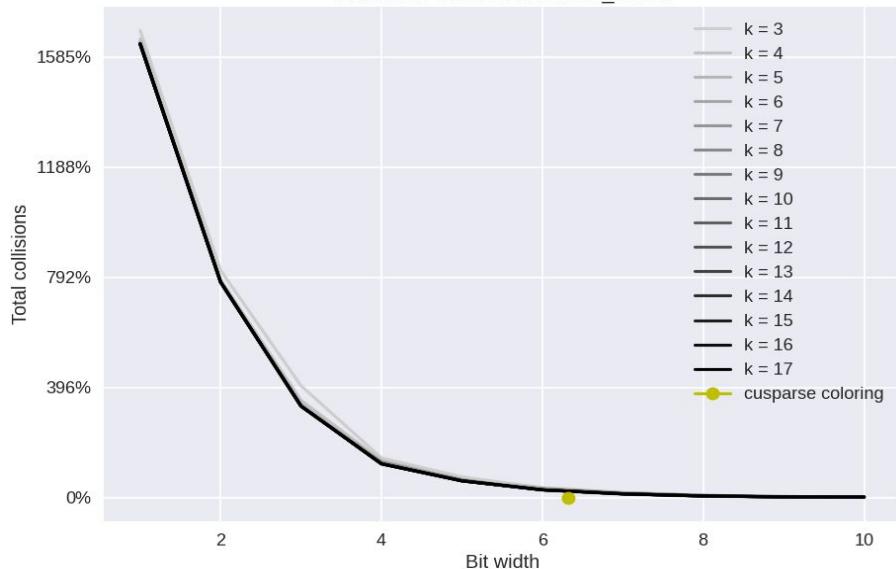
Distance2 Max Collisions CurlCurl_4



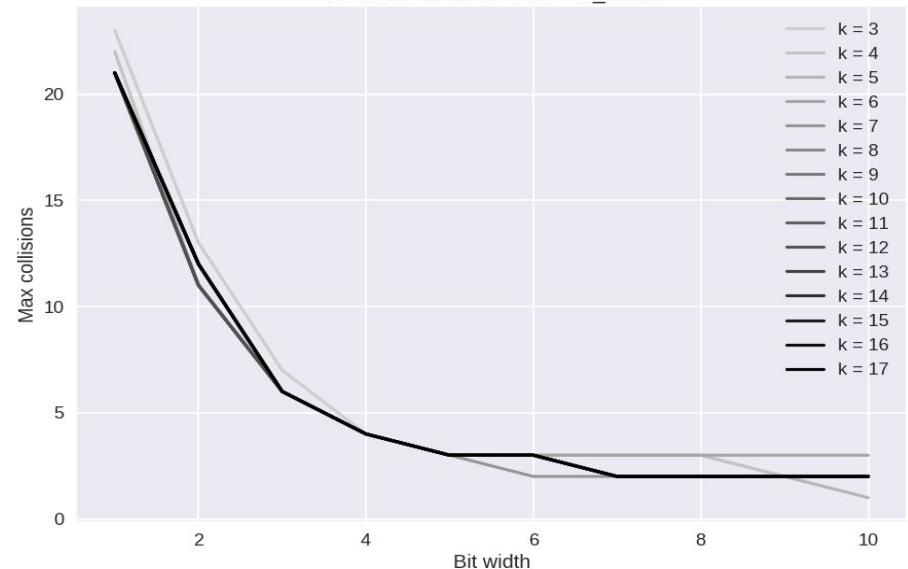
- old project: 65536 colors ($k = 3$)



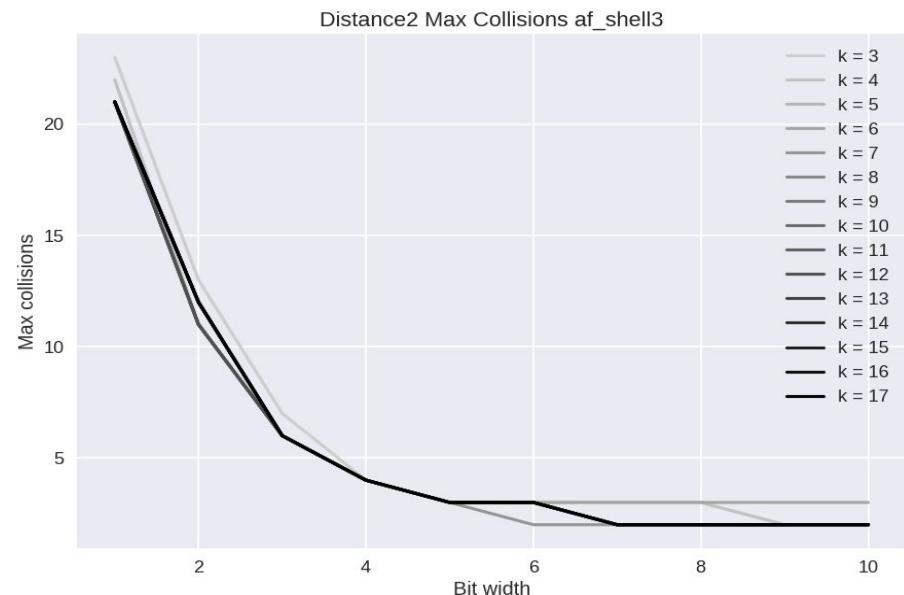
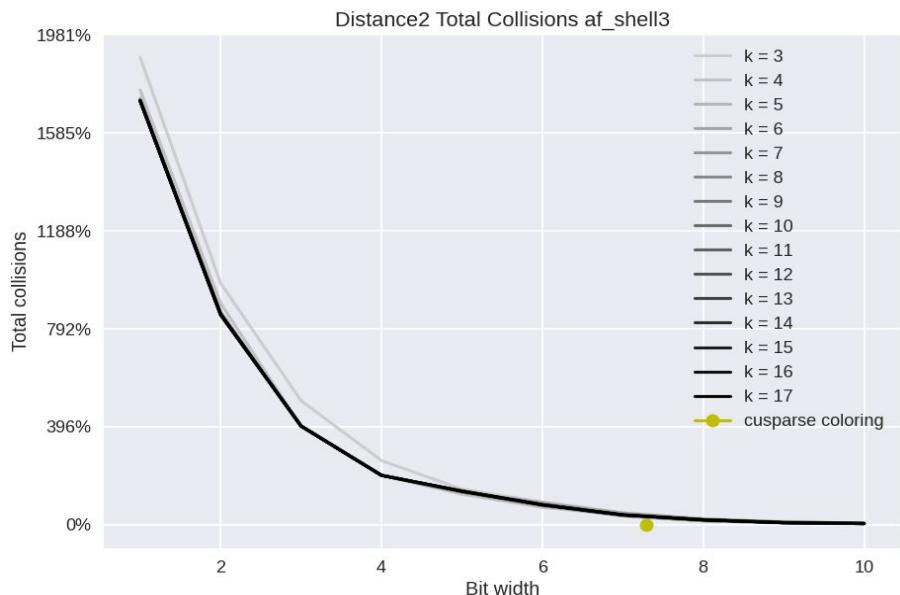
Distance1 Total Collisions af_shell3



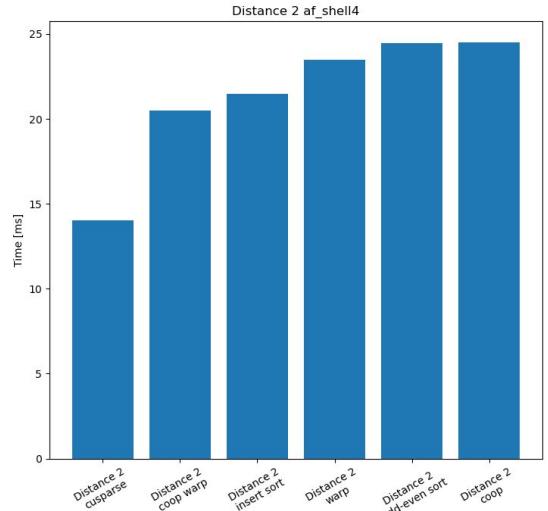
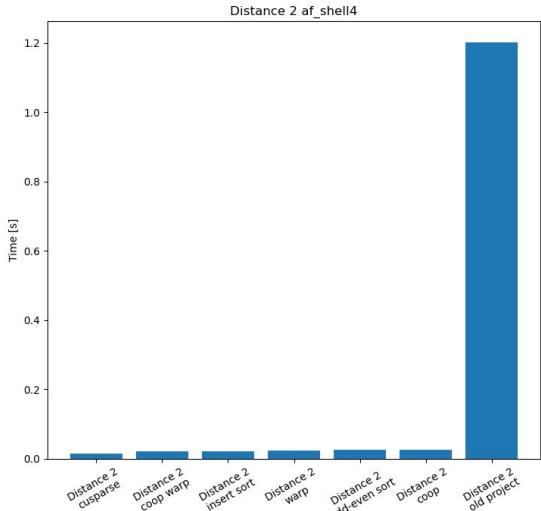
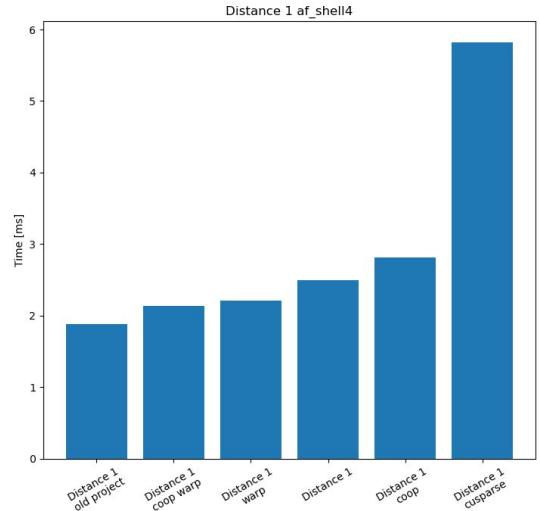
Distance1 Max Collisions af_shell3

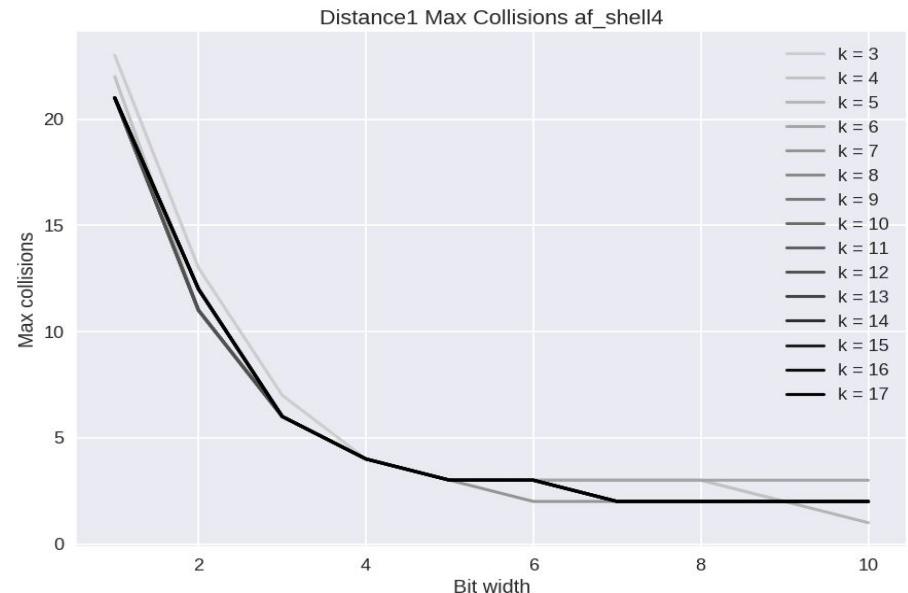
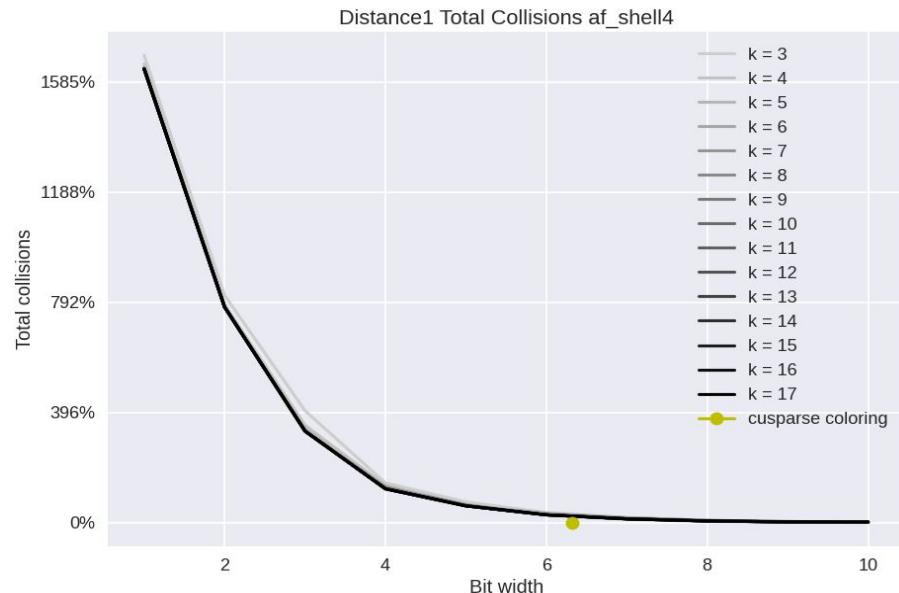


- old project: 4096 colors (k = 3)

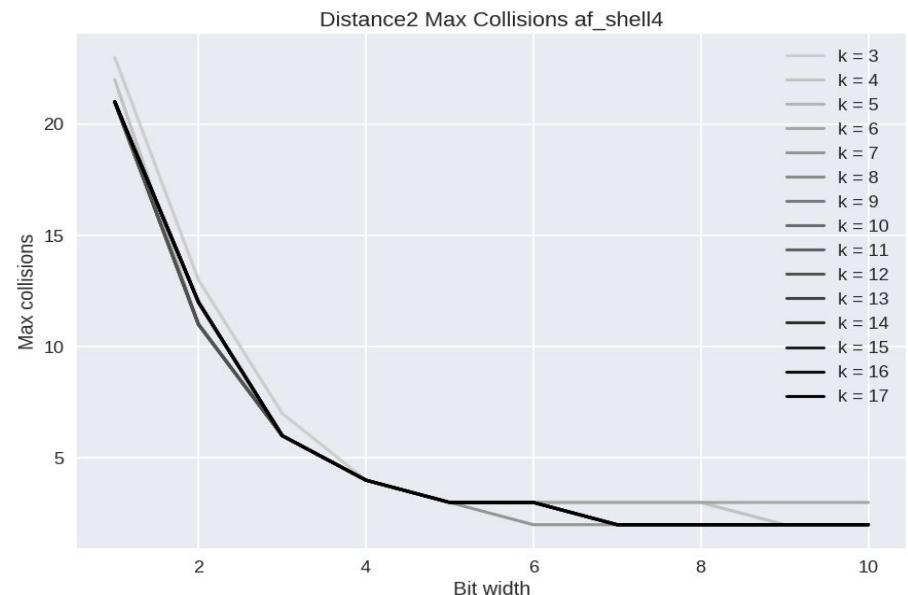
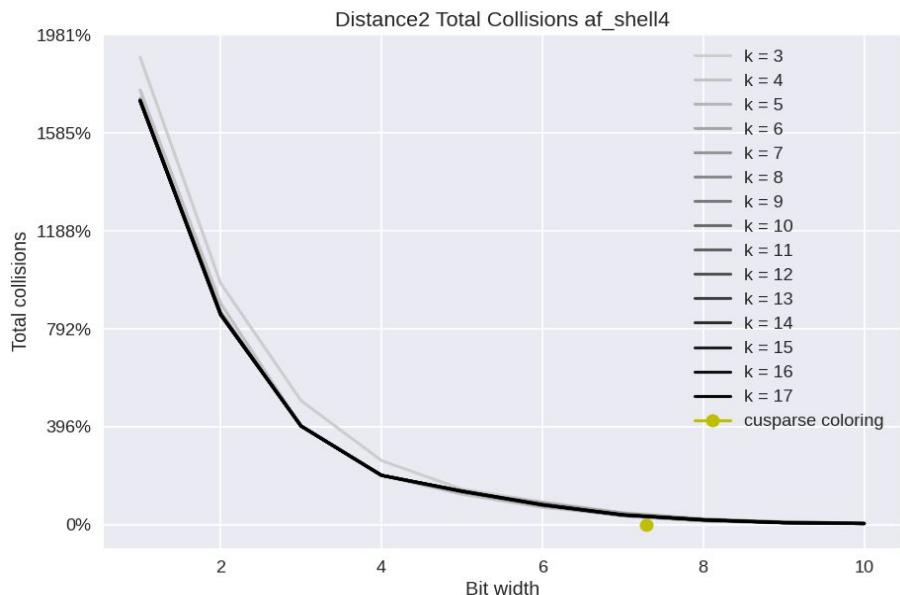


- old project: 8192 colors ($k = 4$)

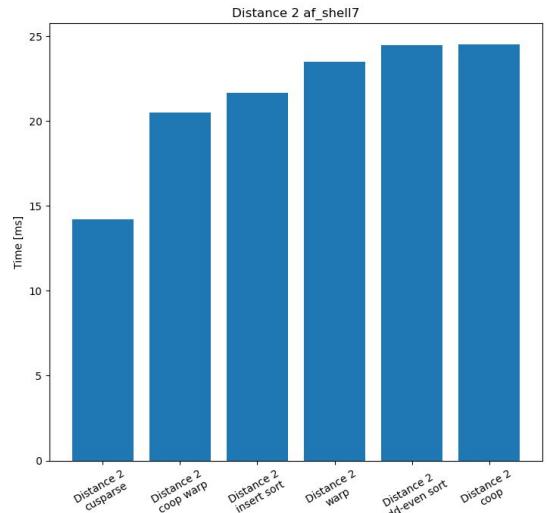
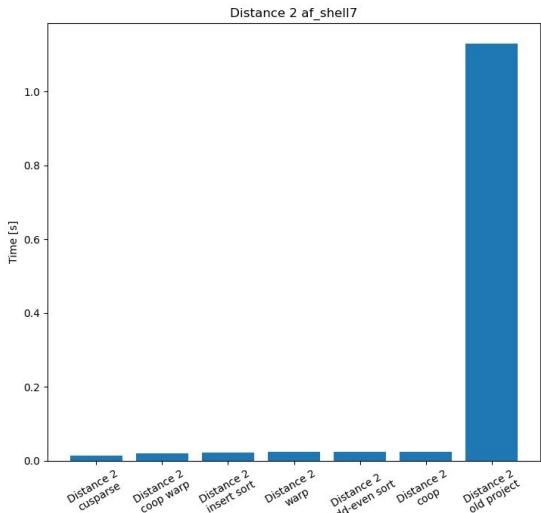
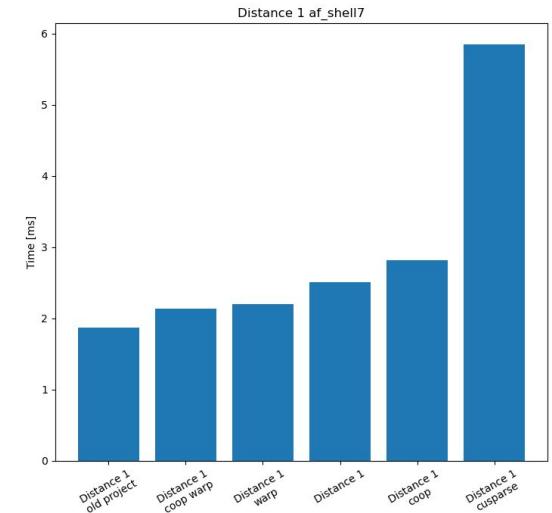




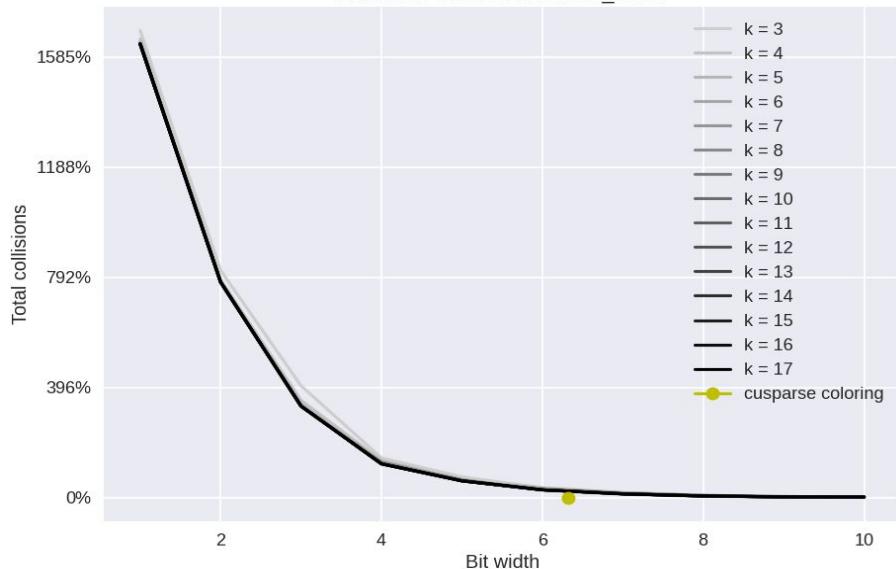
- old project: 4096 colors (k = 3)



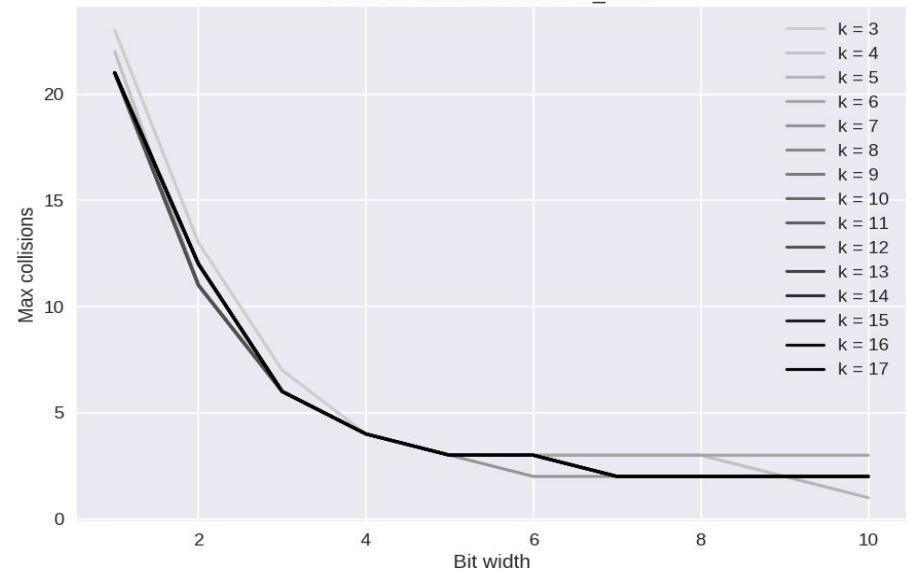
- old project: 8192 colors ($k = 4$)



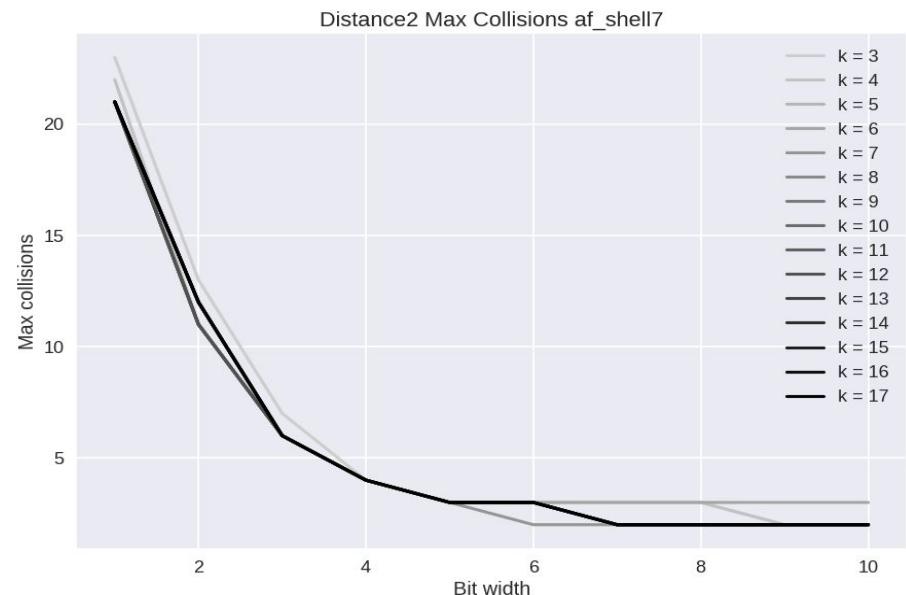
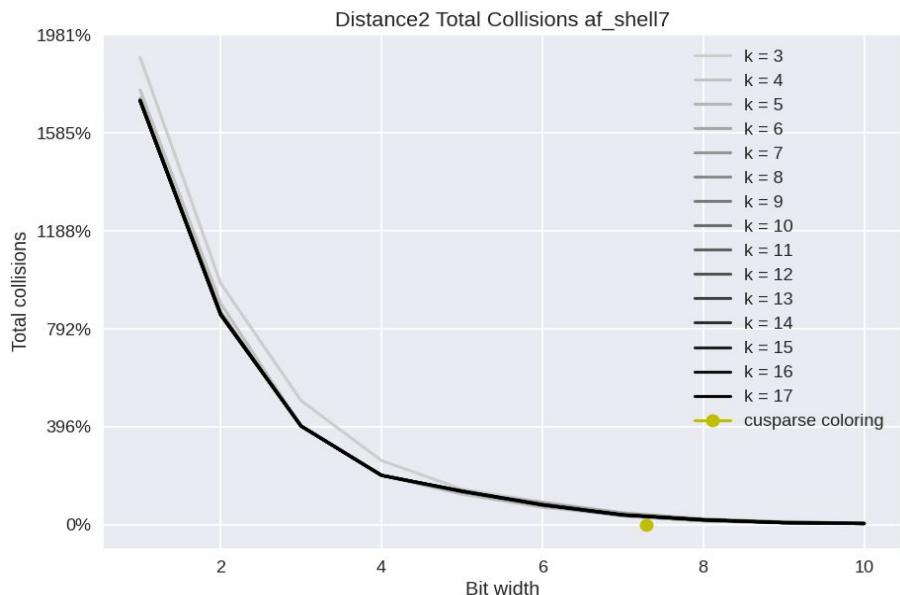
Distance1 Total Collisions af_shell7



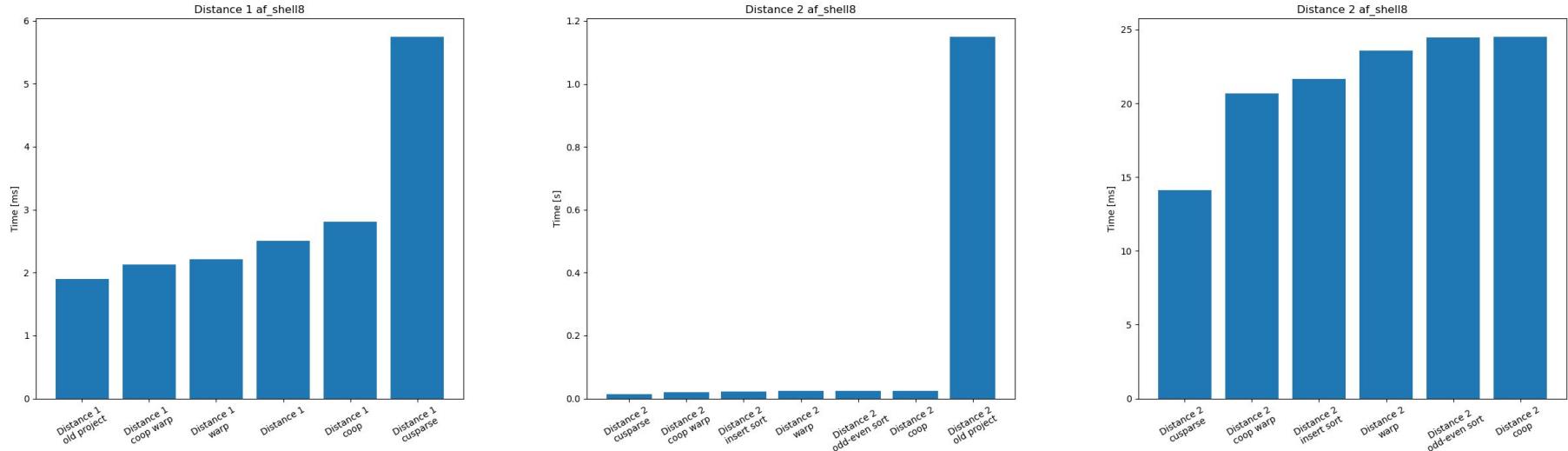
Distance1 Max Collisions af_shell7



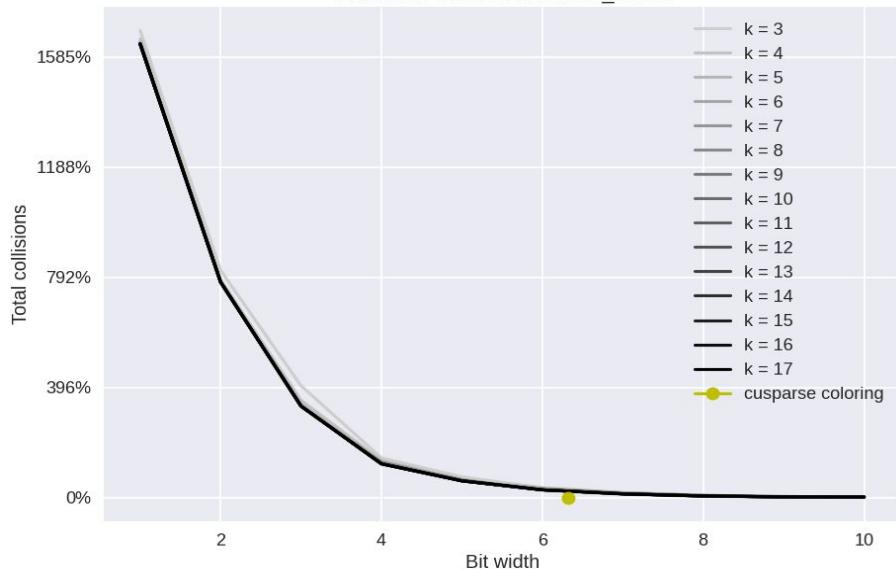
- old project: 4096 colors (k = 3)



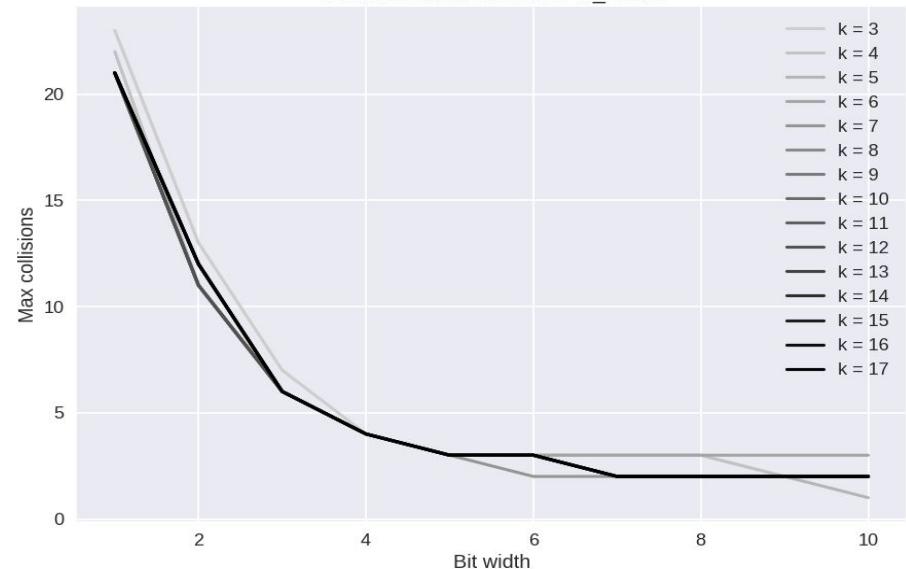
- old project: 8192 colors (k = 4)



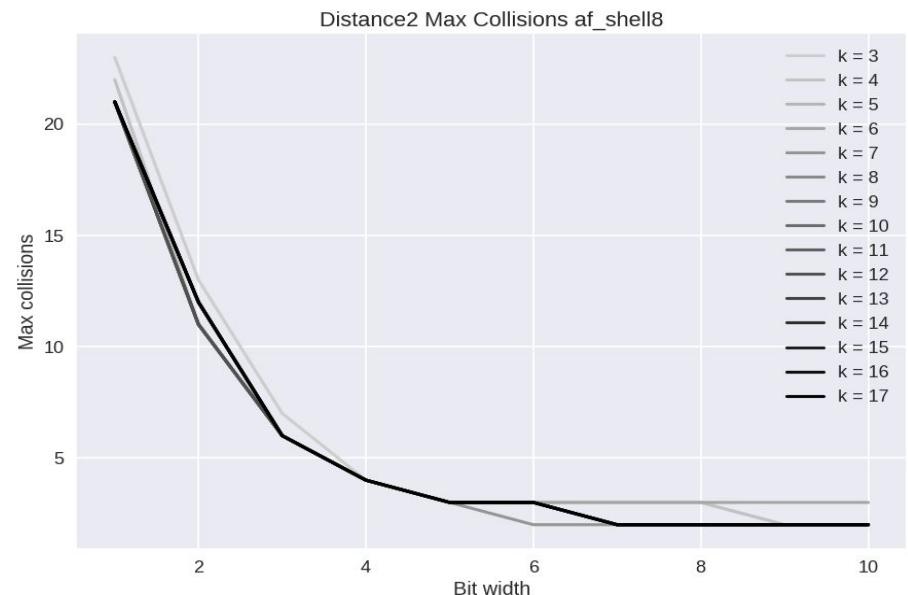
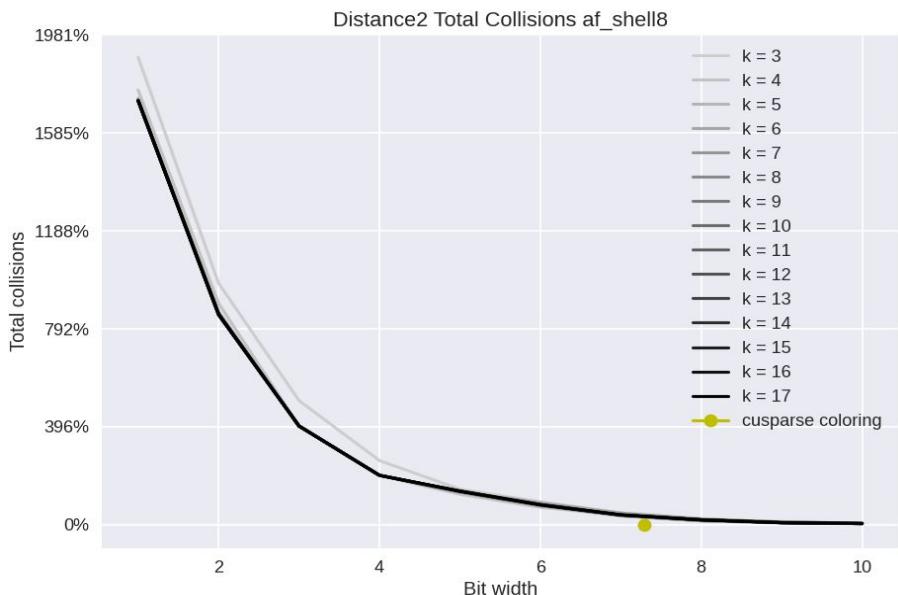
Distance1 Total Collisions af_shell8



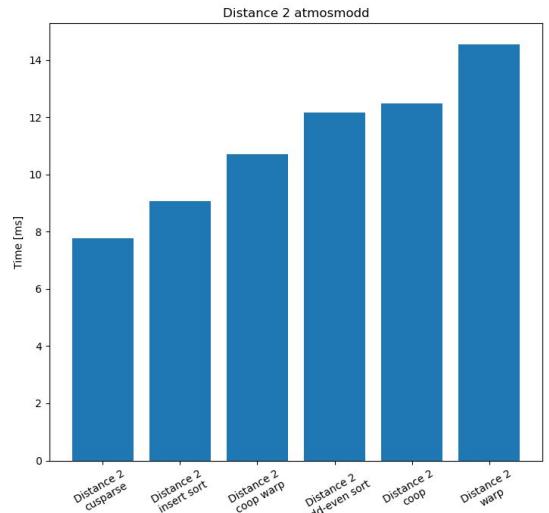
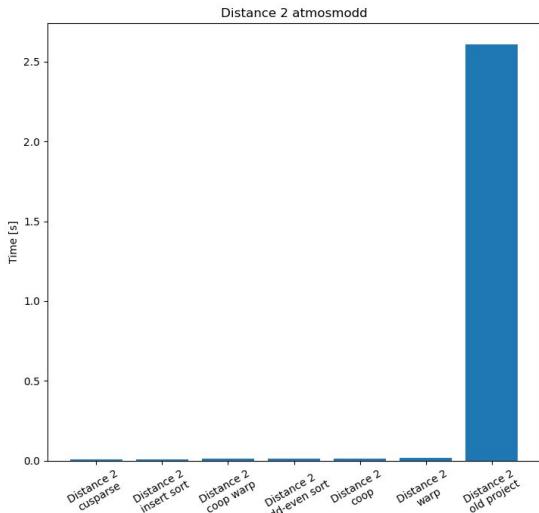
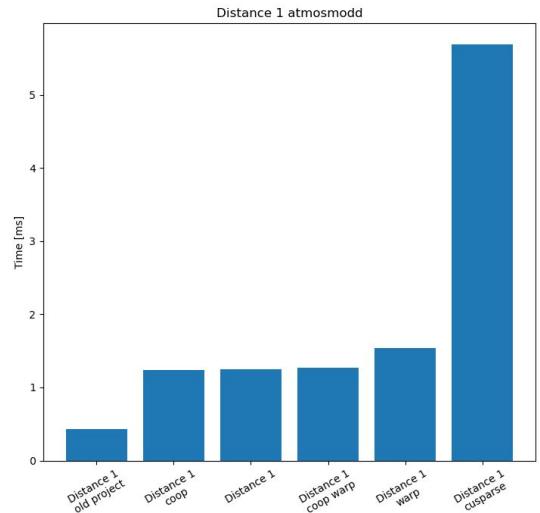
Distance1 Max Collisions af_shell8



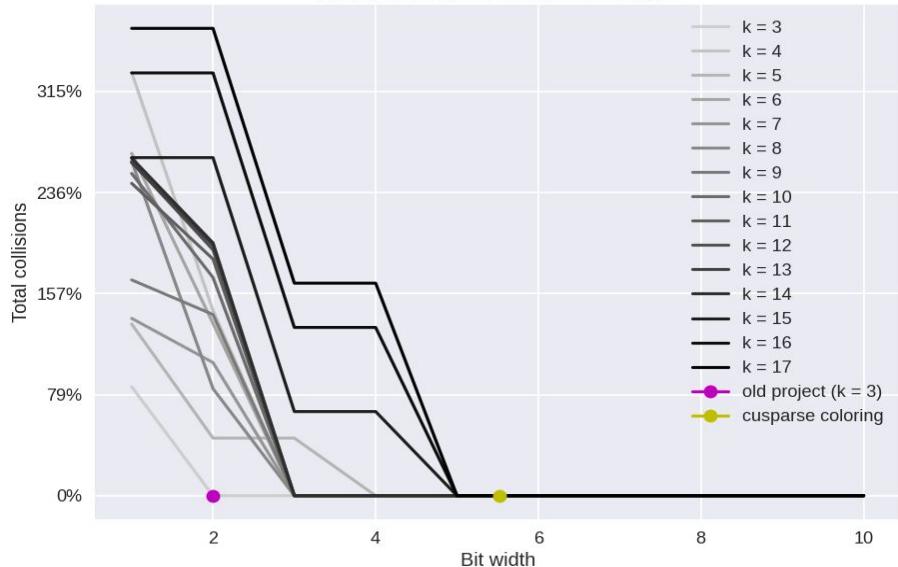
- old project: 4096 colors ($k = 3$)



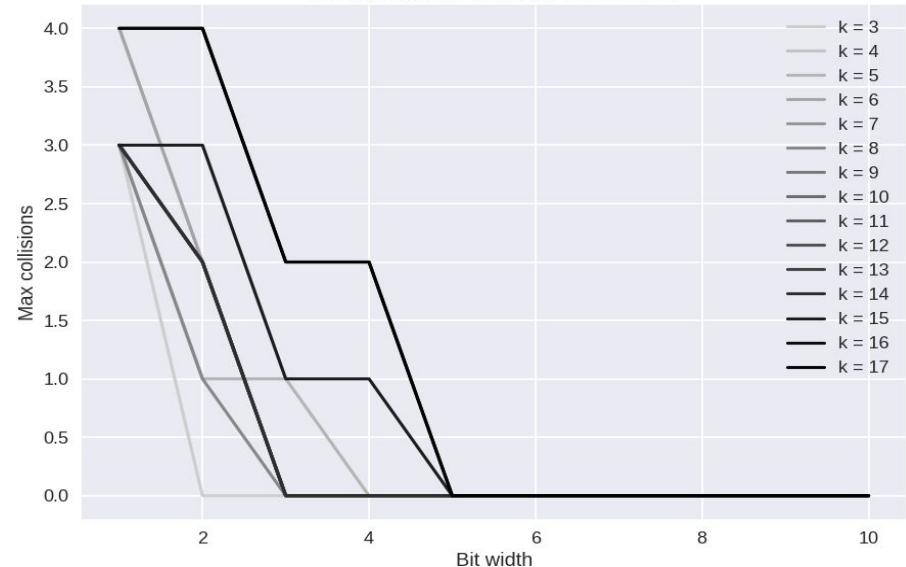
- old project: 8192 colors ($k = 4$)



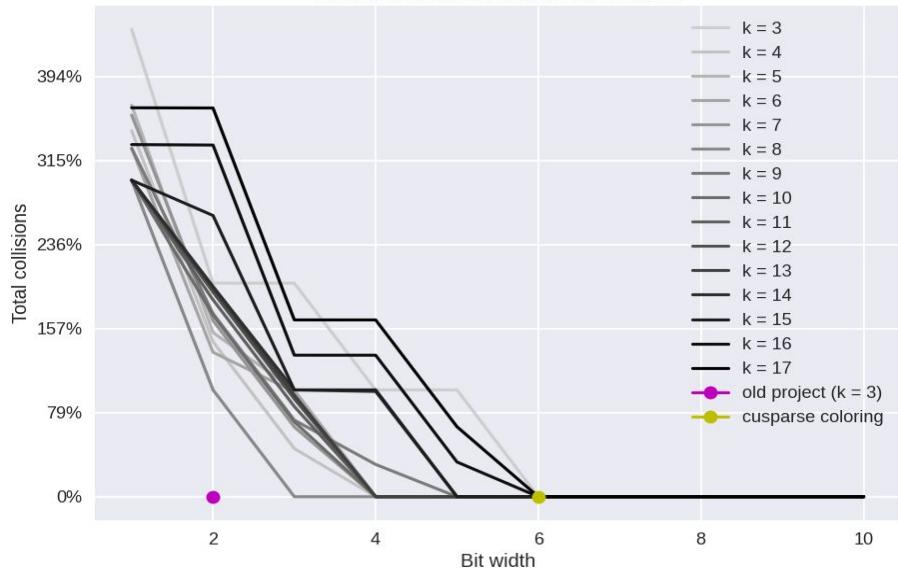
Distance1 Total Collisions atmosmodd



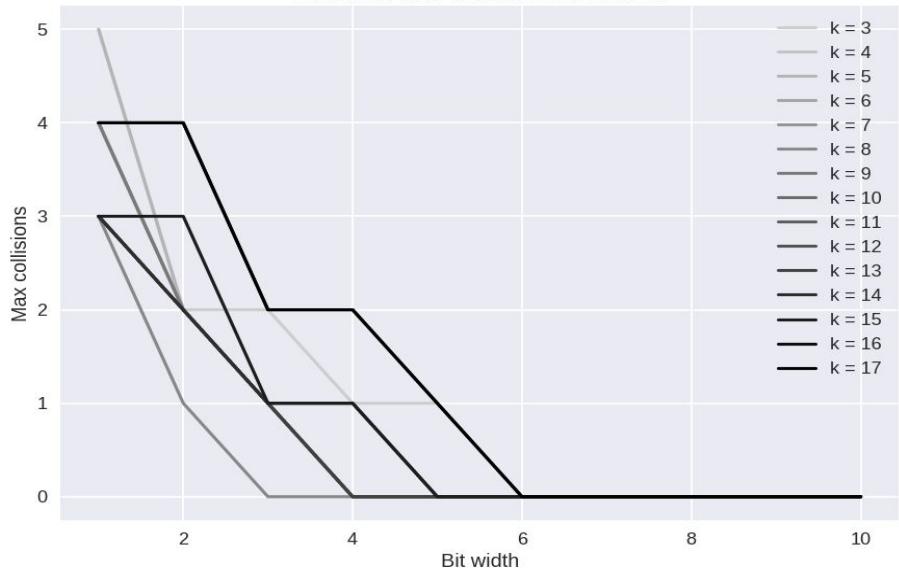
Distance1 Max Collisions atmosmodd



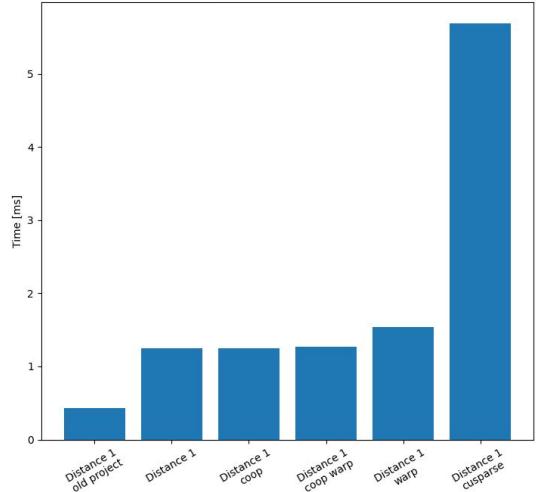
Distance2 Total Collisions atmosmodd



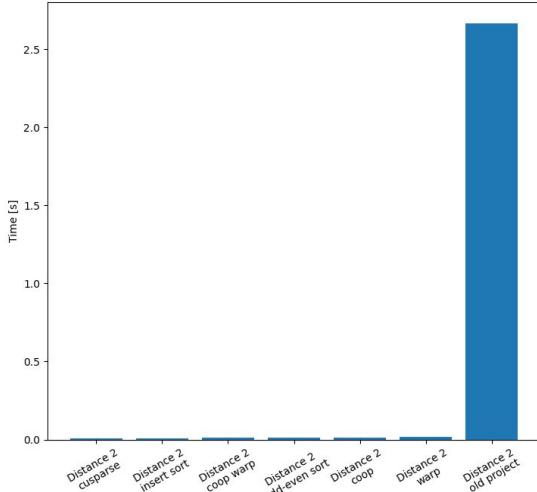
Distance2 Max Collisions atmosmodd



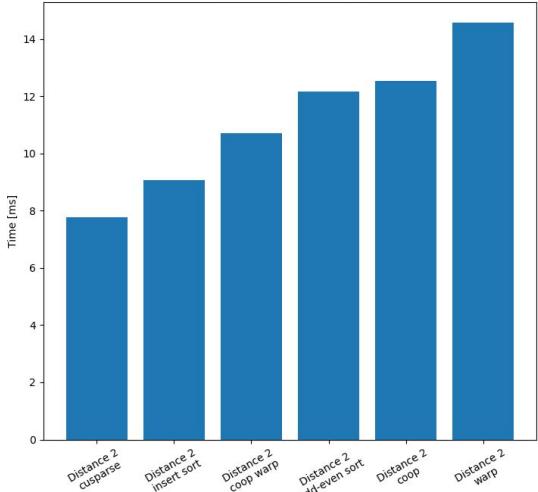
Distance 1 atmosmodj



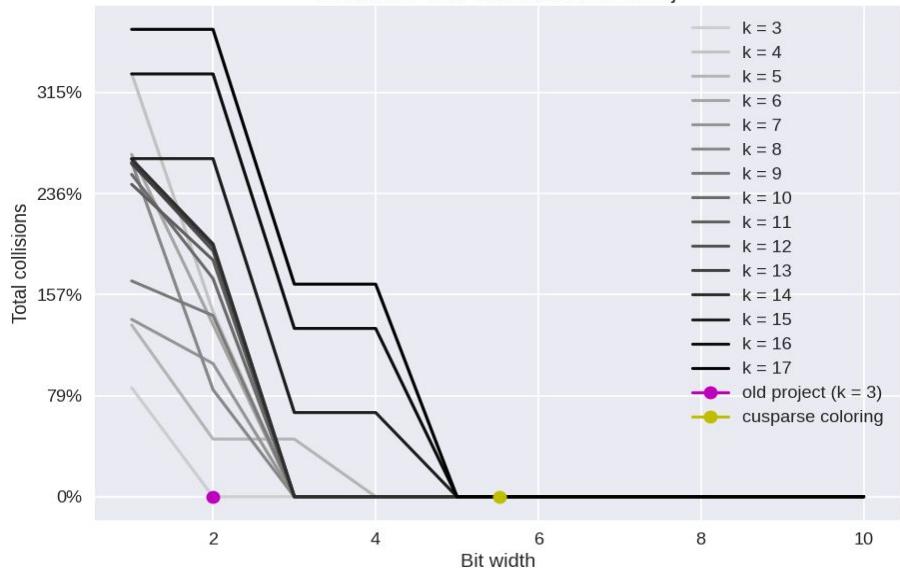
Distance 2 atmosmodj



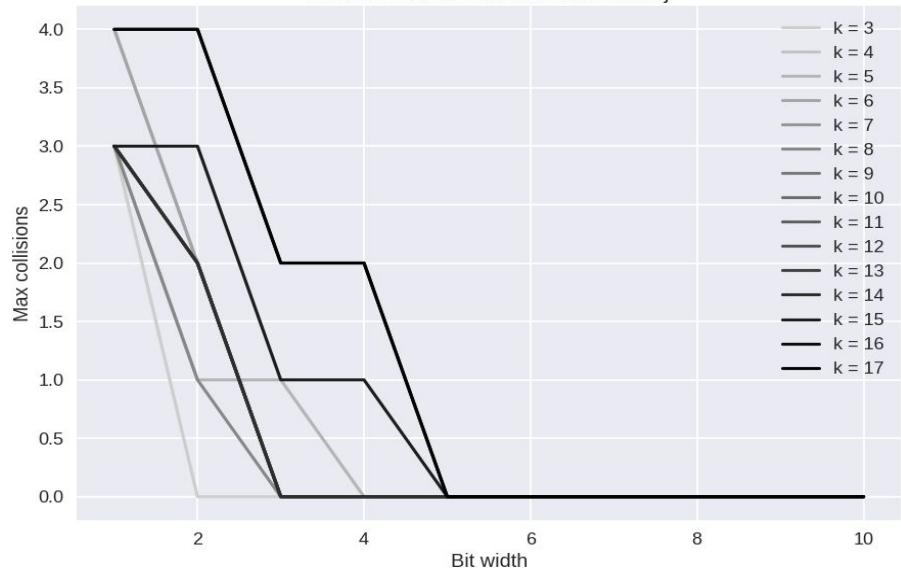
Distance 2 atmosmodj



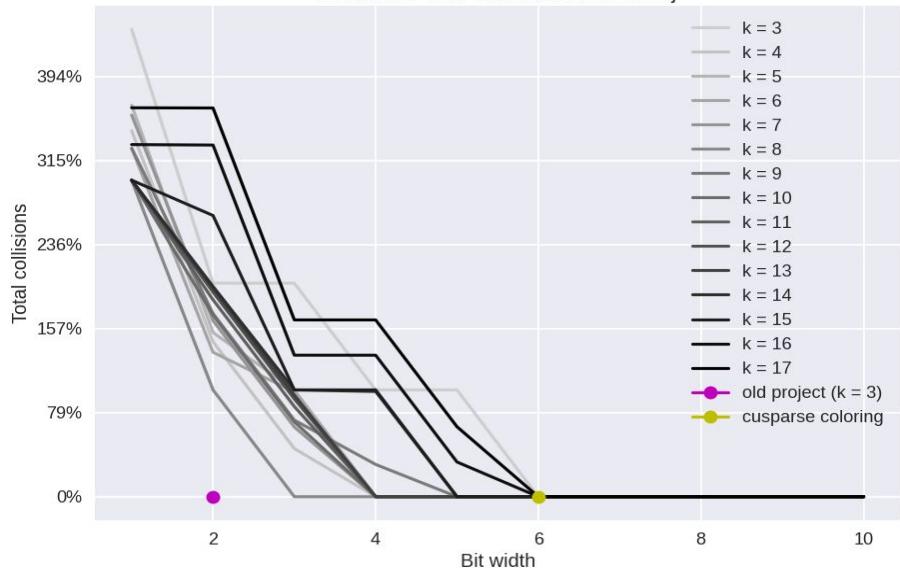
Distance1 Total Collisions atmosmodj



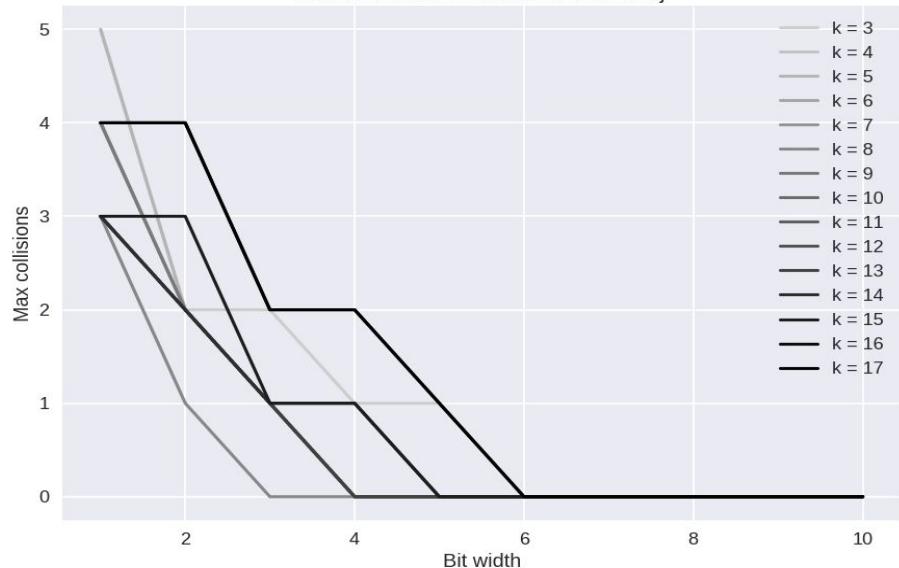
Distance1 Max Collisions atmosmodj



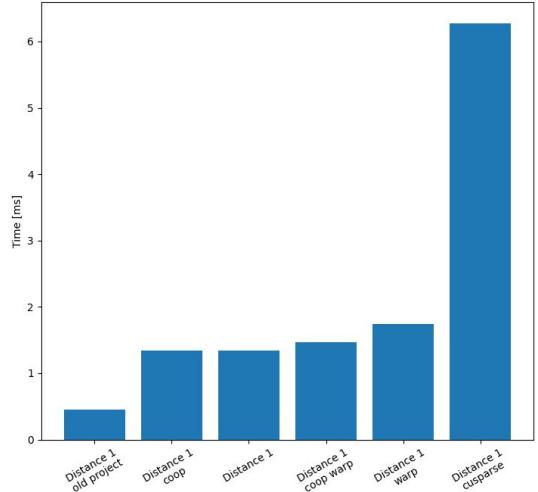
Distance2 Total Collisions atmosmodj



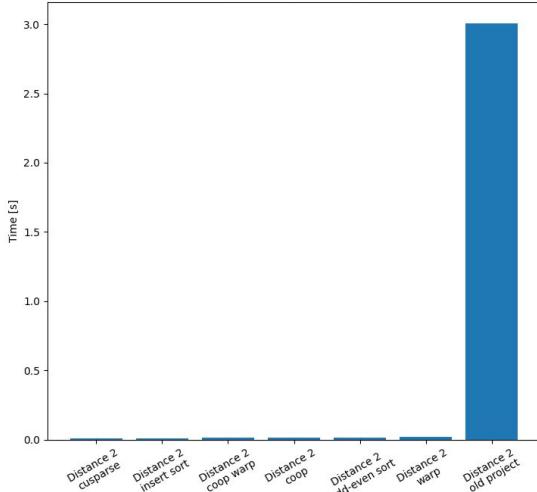
Distance2 Max Collisions atmosmodj



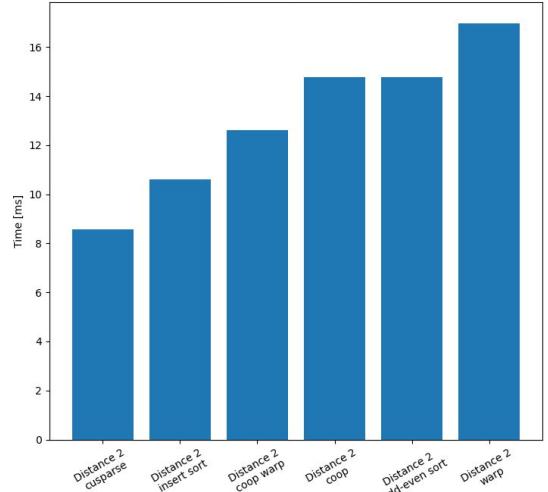
Distance 1 atmosmodl



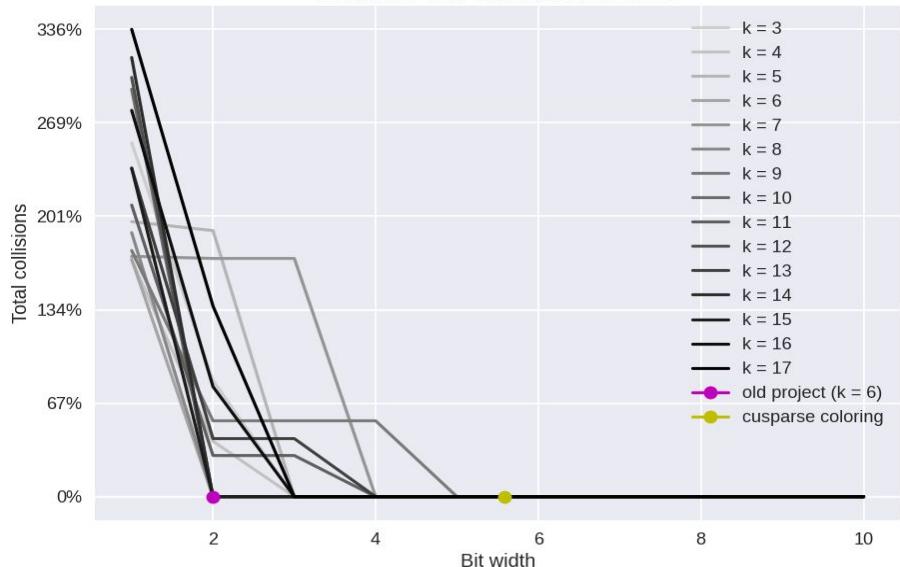
Distance 2 atmosmodl



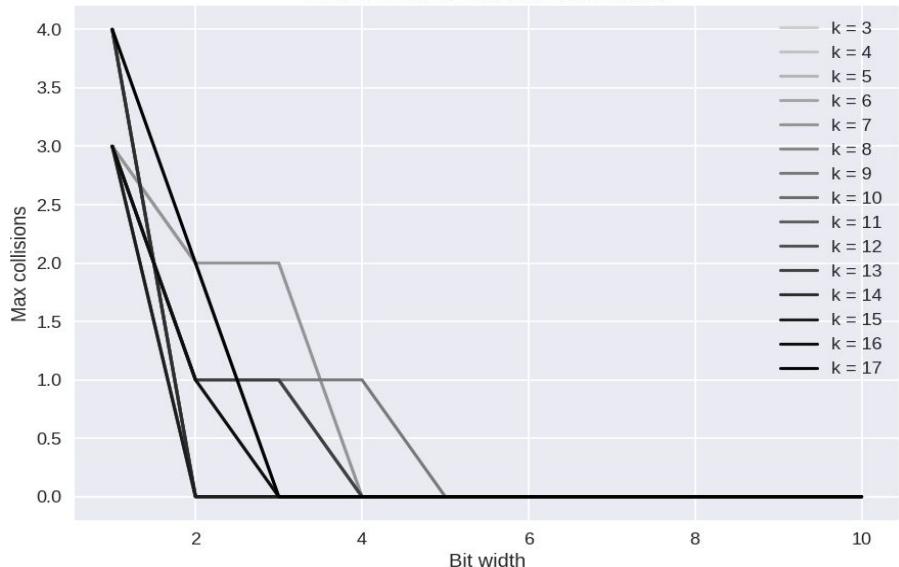
Distance 2 atmosmodl



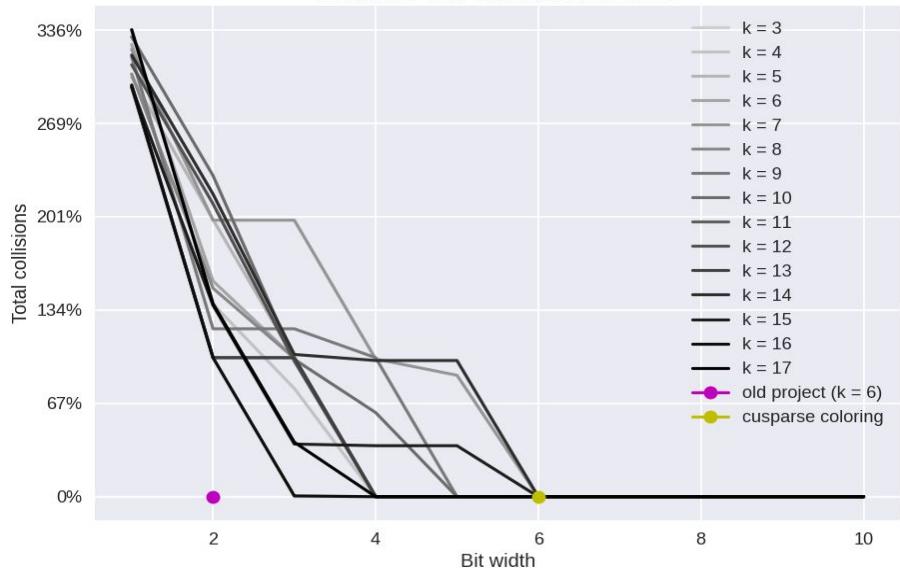
Distance1 Total Collisions atmosmodl



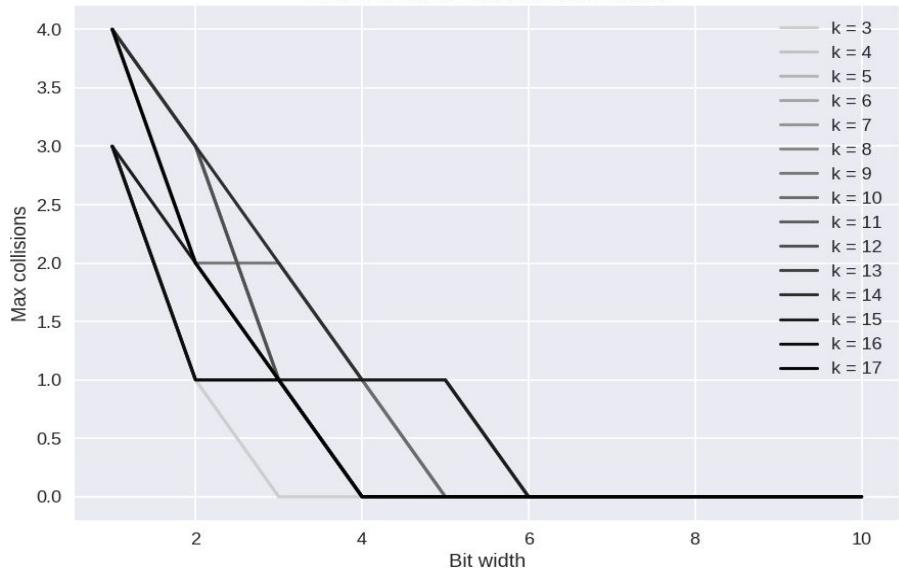
Distance1 Max Collisions atmosmodl



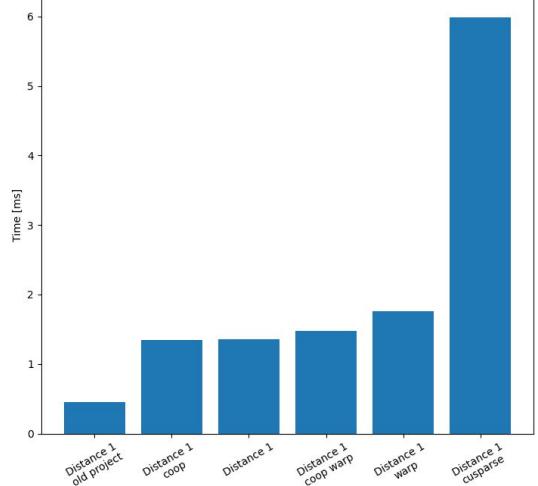
Distance2 Total Collisions atmosmodl



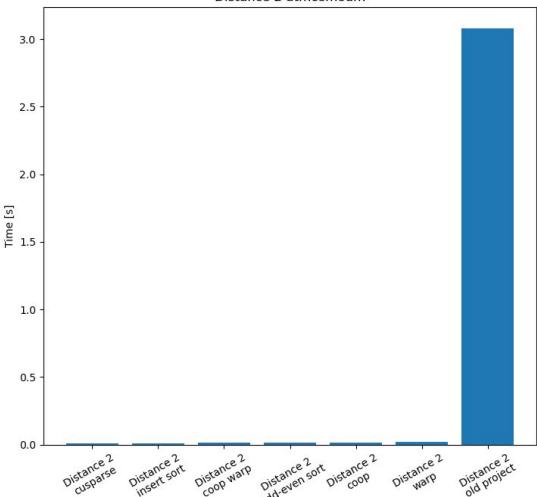
Distance2 Max Collisions atmosmodl



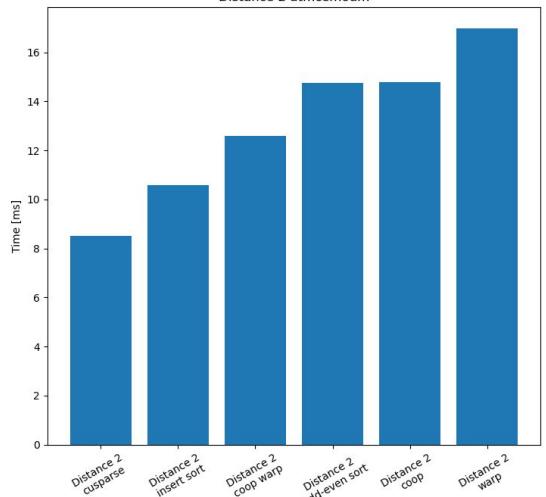
Distance 1 atmosmodm



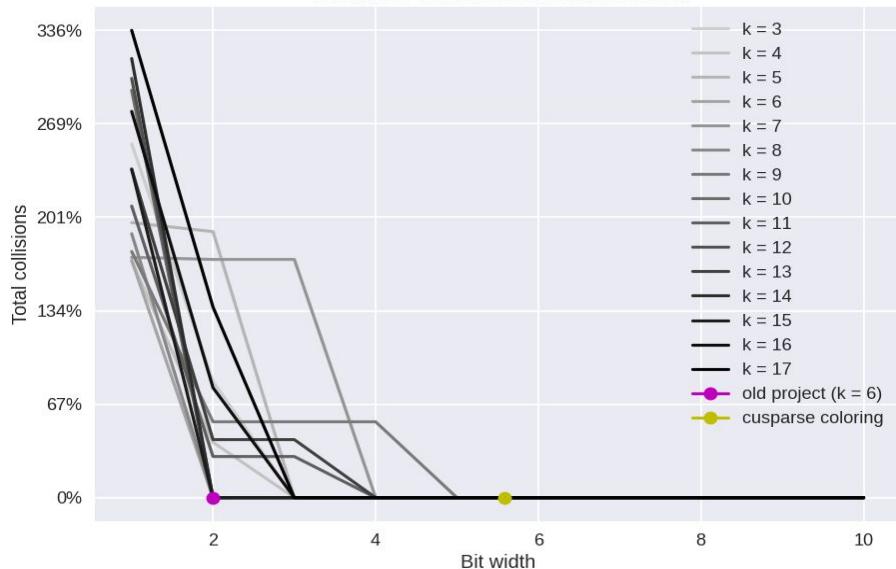
Distance 2 atmosmodm



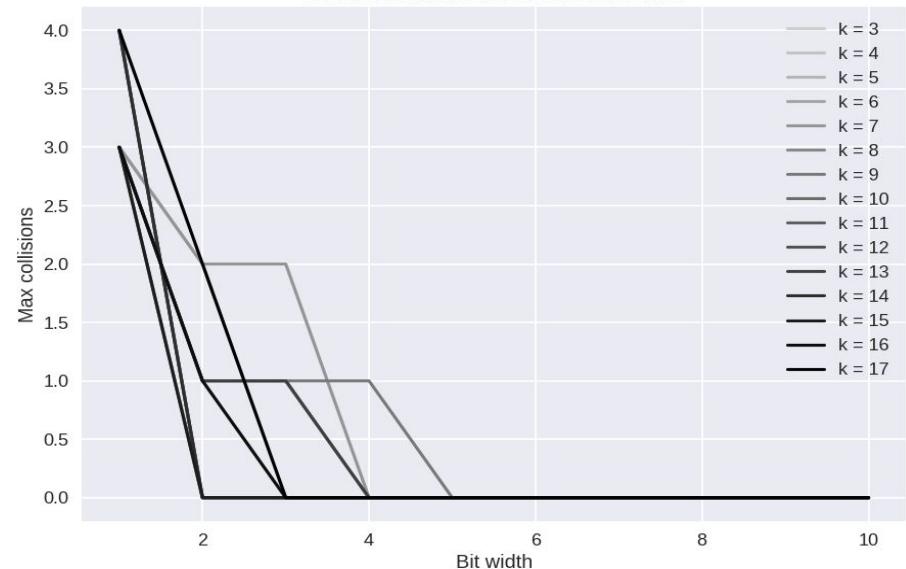
Distance 2 atmosmodm



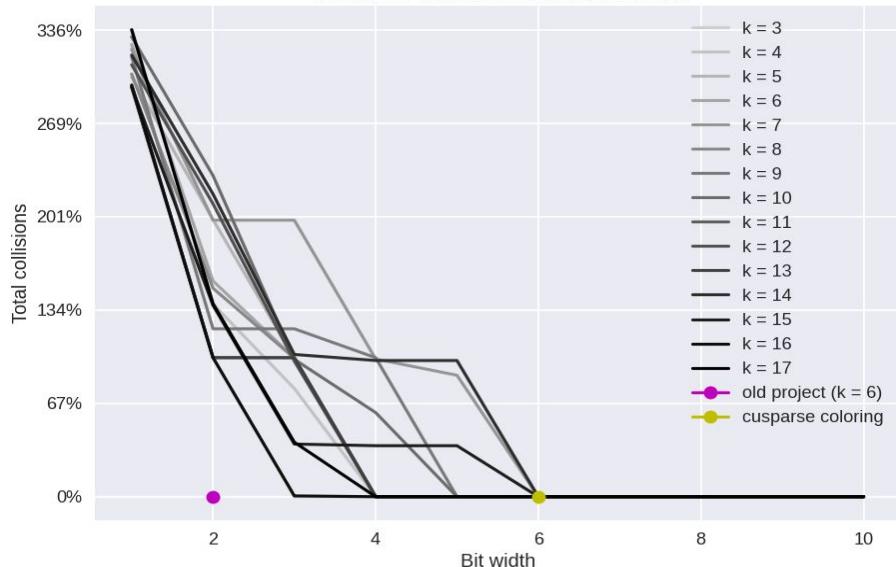
Distance1 Total Collisions atmosmodm



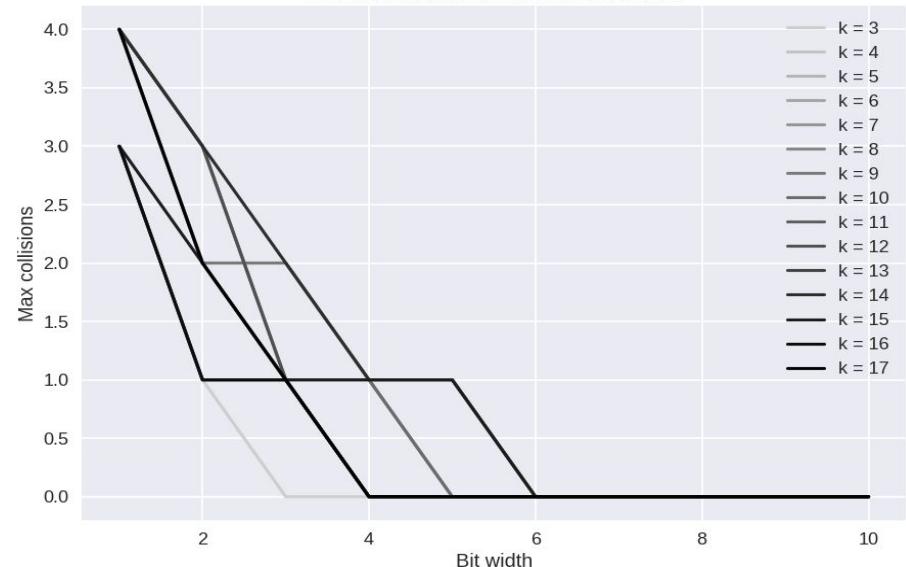
Distance1 Max Collisions atmosmodm

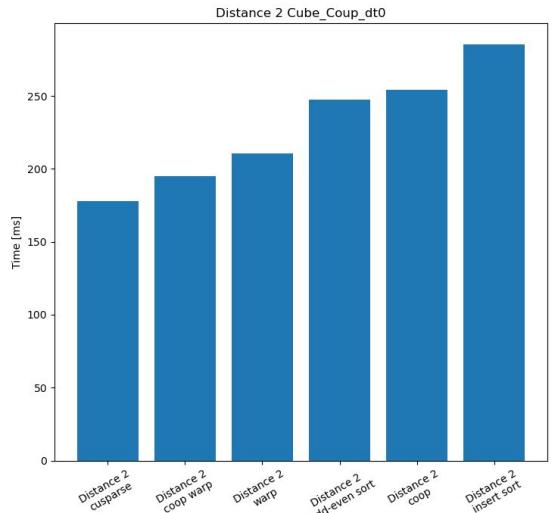
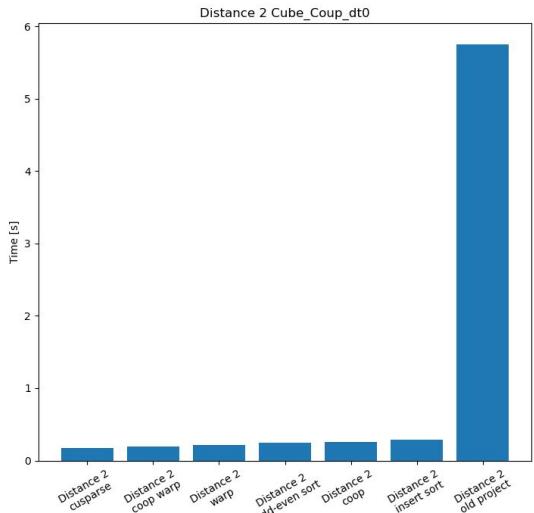
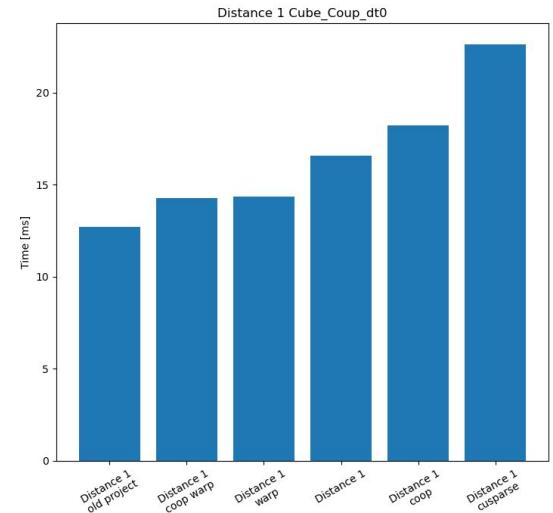


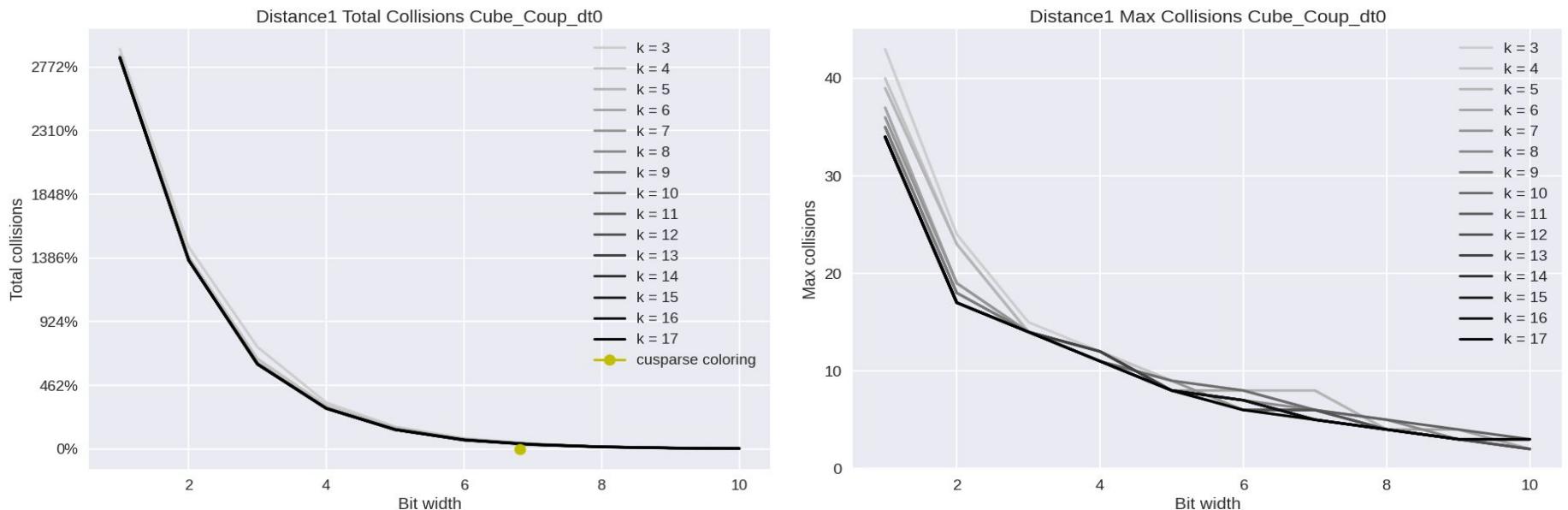
Distance2 Total Collisions atmosmodm



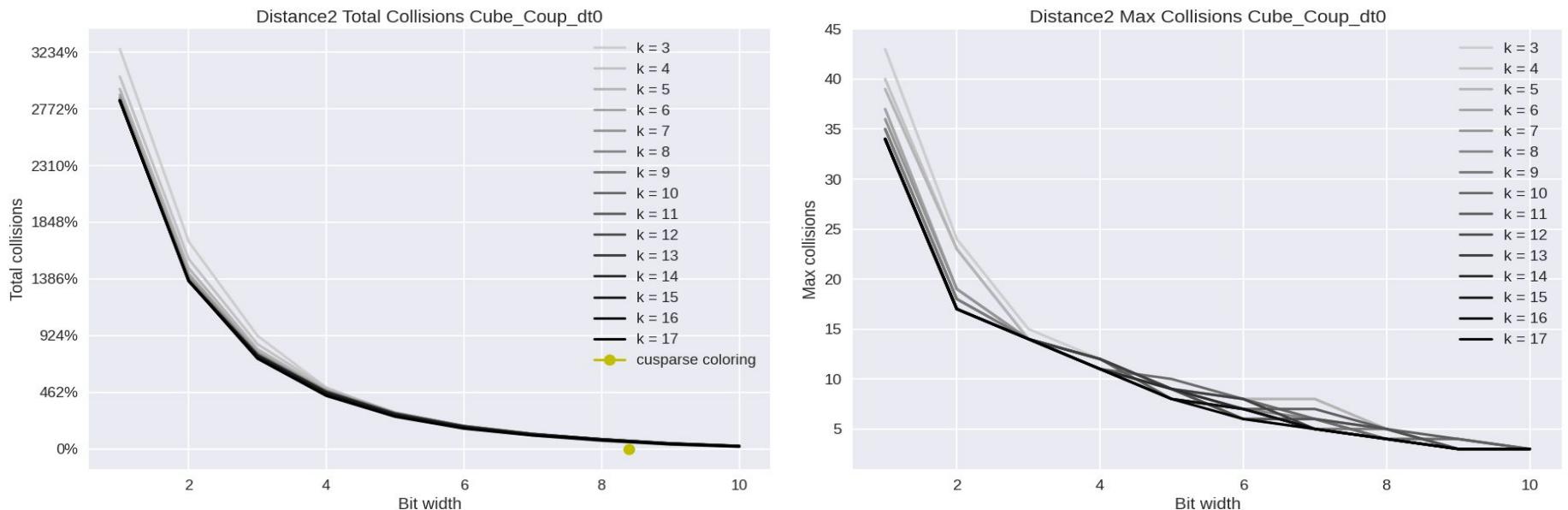
Distance2 Max Collisions atmosmodm



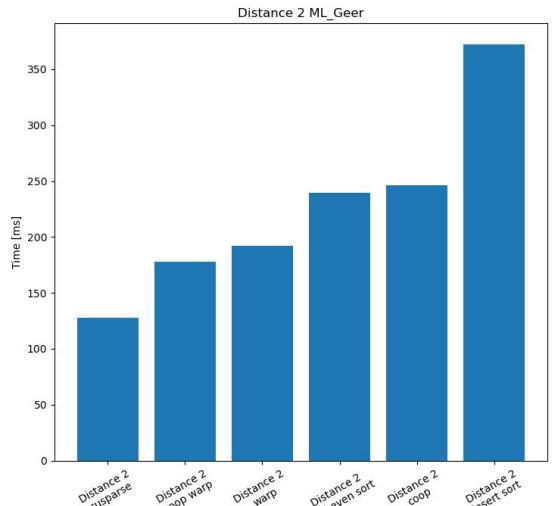
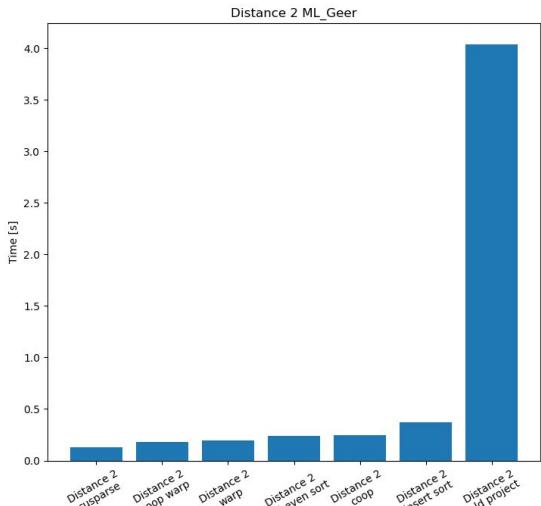
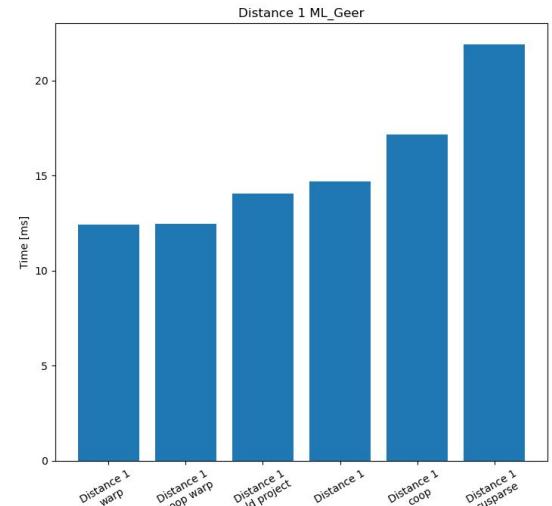


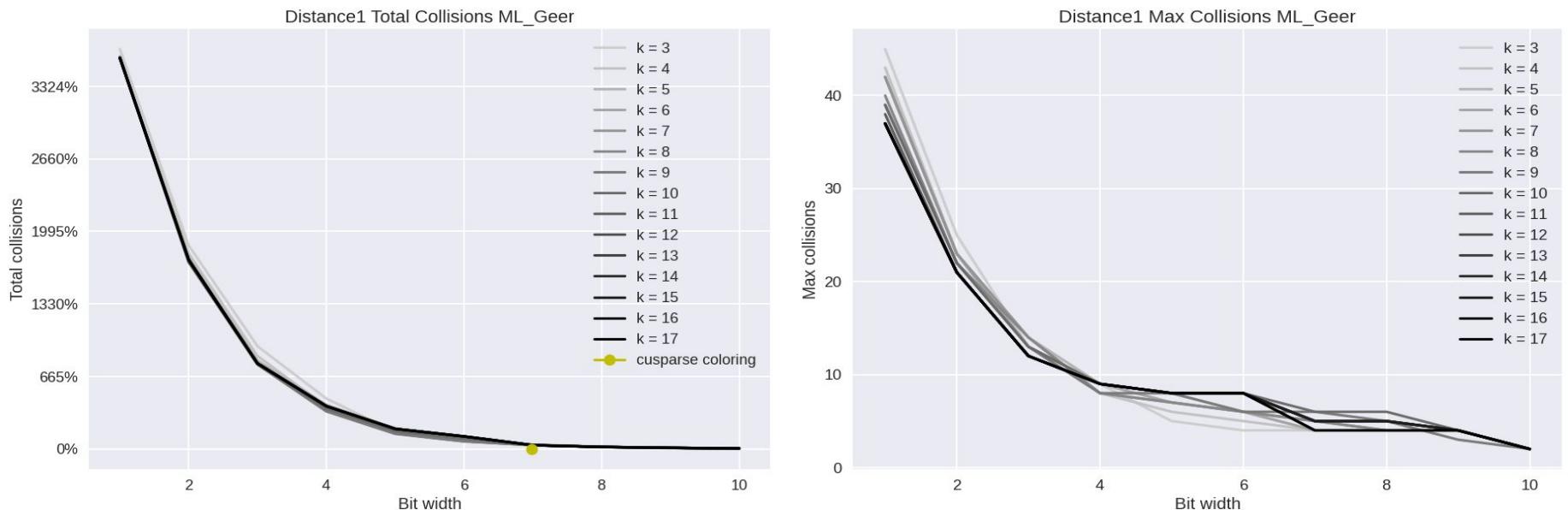


- old project: 32768 colors ($k = 5$)

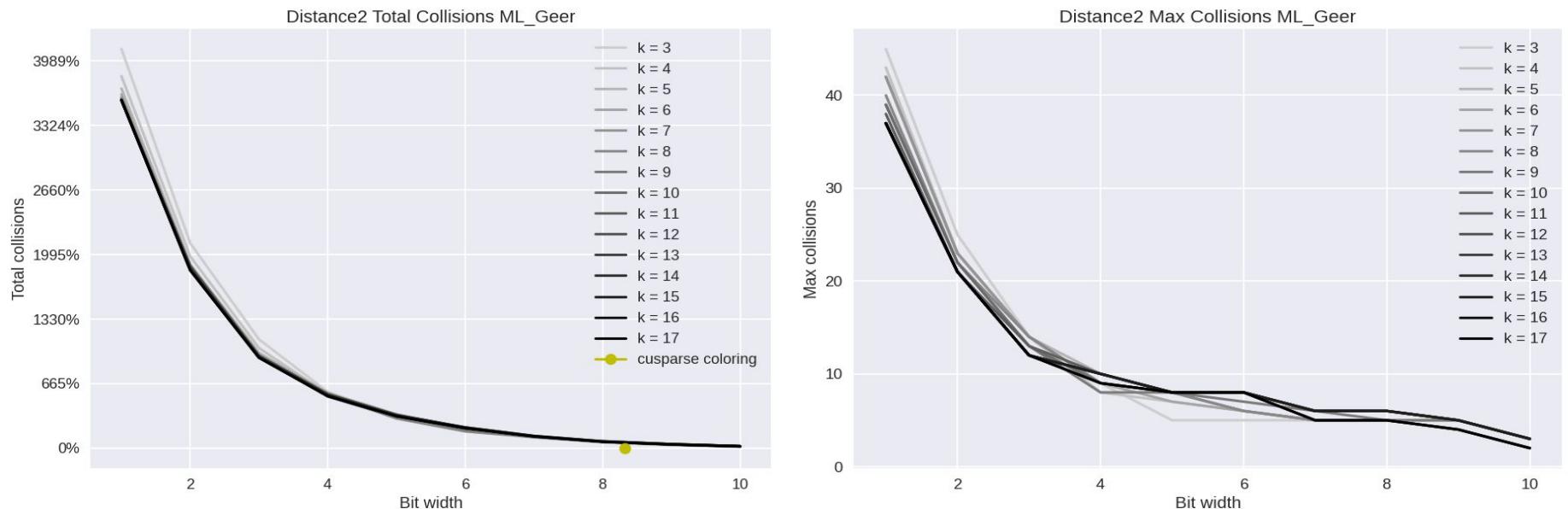


- old project: 65536 colors ($k = 5$)





- old project: 8192 colors (k = 3)



- old project: 16384 colors ($k = 3$)