CLICKHOUSE OPTIMIZATIONS FOR ARM

Daniel Kutenin

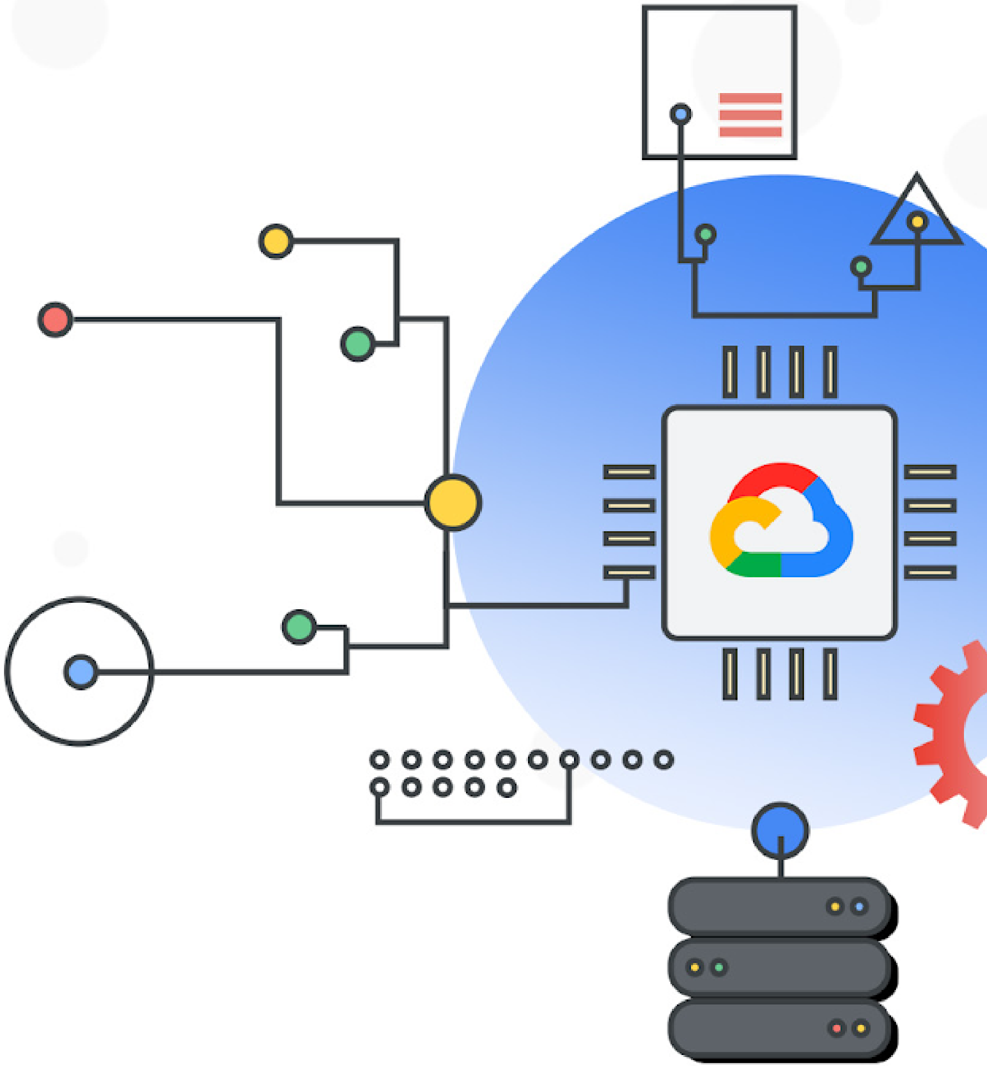
~~Google~~

WHO AM I?

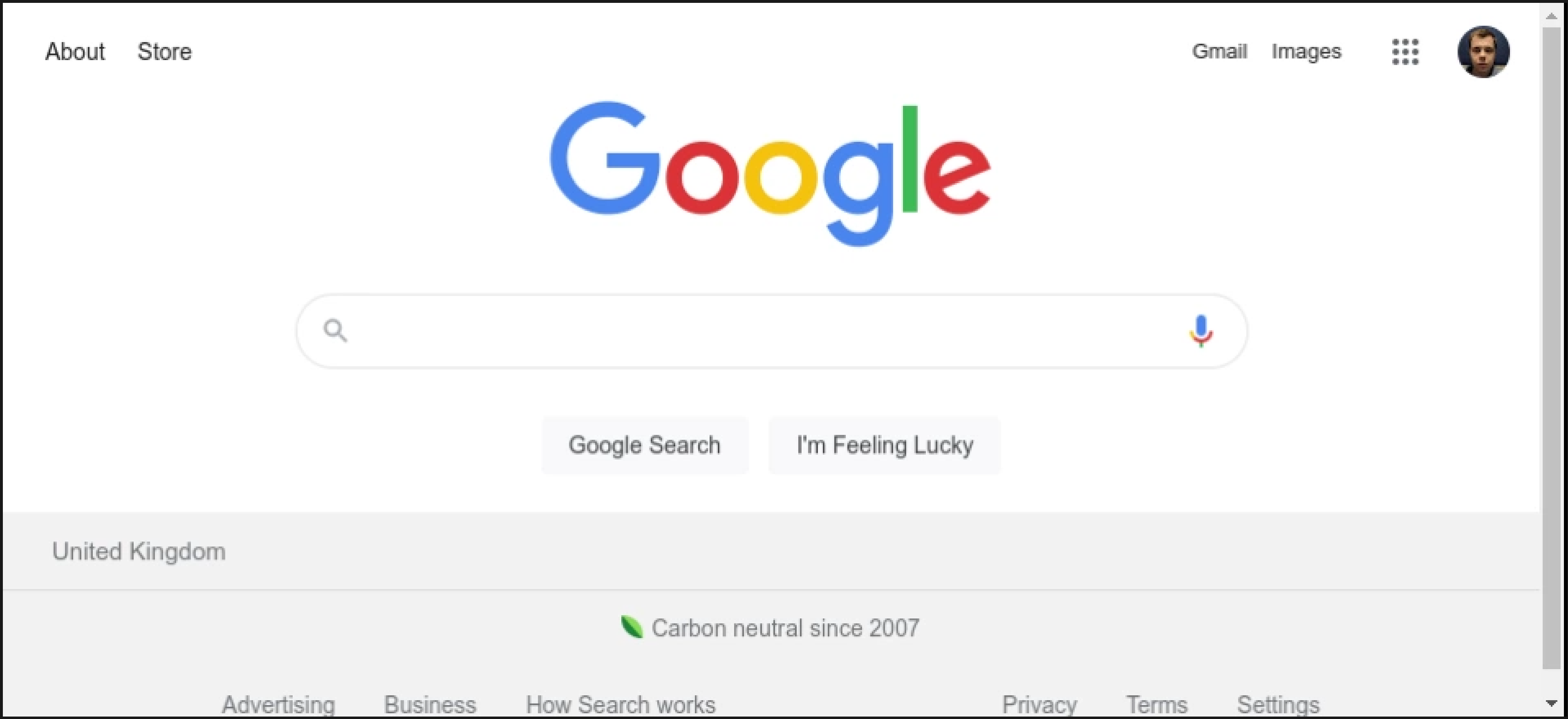
- Senior Software Engineer at Google Cloud
- ClickHouse infra and efficiency contributor
- C++ library and compiler contributor
- C++ teacher in universities

BUT

Tau T2A is first ompute Engine VM to run on Arm



Google Cloud



https://cloud.google.com/blog/products/compute/tau-t2a-is-first-compute-engine-vm-on-an-arm-chip

WHO AM I?

- Senior Software Engineer at Google Cloud
- ClickHouse infra and efficiency contributor
- C++ library and compiler contributor
- C++ teacher in universities

- ClickHouse infra and efficiency contributor

- efficiency

WHY ARM?

💰 15-20%* cost reduction of perf/$

☁️ Cloud providers finally "believed" in it

🧰 AWS, Azure, GCloud, Oracle, Alibaba, etc

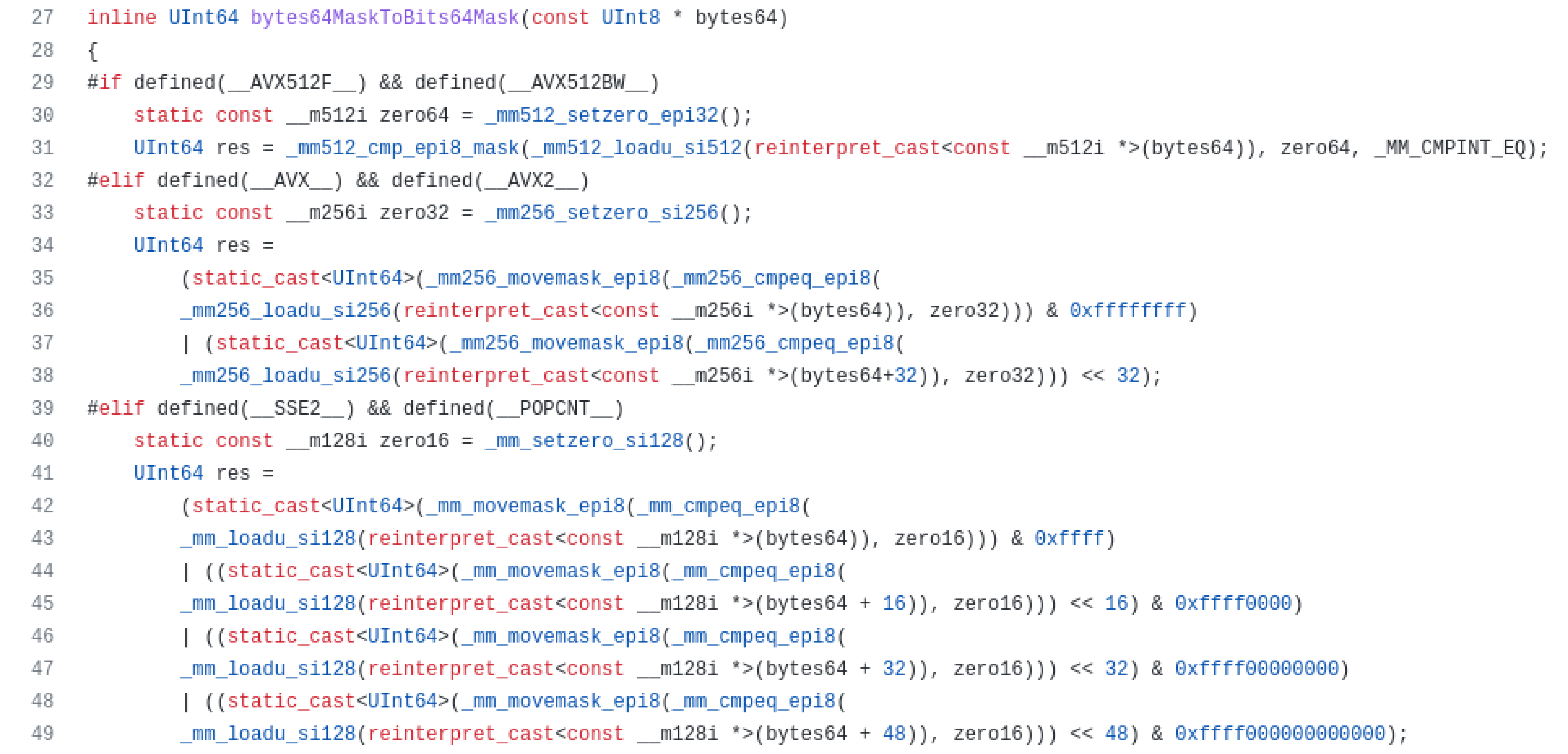
🏢 Corporations (Apple, Google, Amazon, Microsoft)

⌚ Arm managed to make competition to Intel/AMD

TECHNICAL REASONS (PROS)

1. Easier to develop (committee vs corp)
   - Proposal are open (SVE, memory tagging, etc)
   - Google got instructions for memcpy into Armv8.8
2. Less legacy (this will end)
   - 4 byte instructions (decoder is easier)
   - More registers, less moves
   - Easier architecture
3. **Software has gaps**

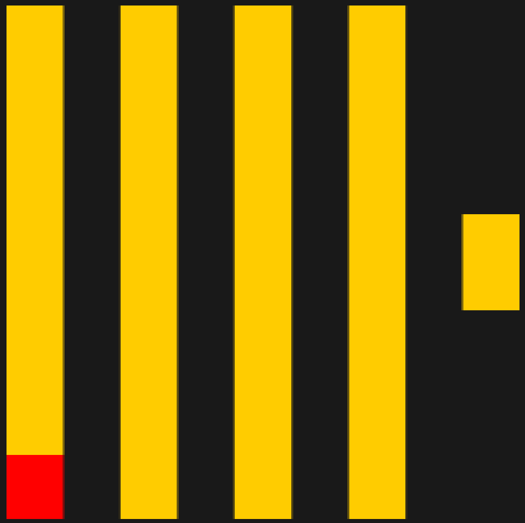
CLICKHOUSE IS AWFUL



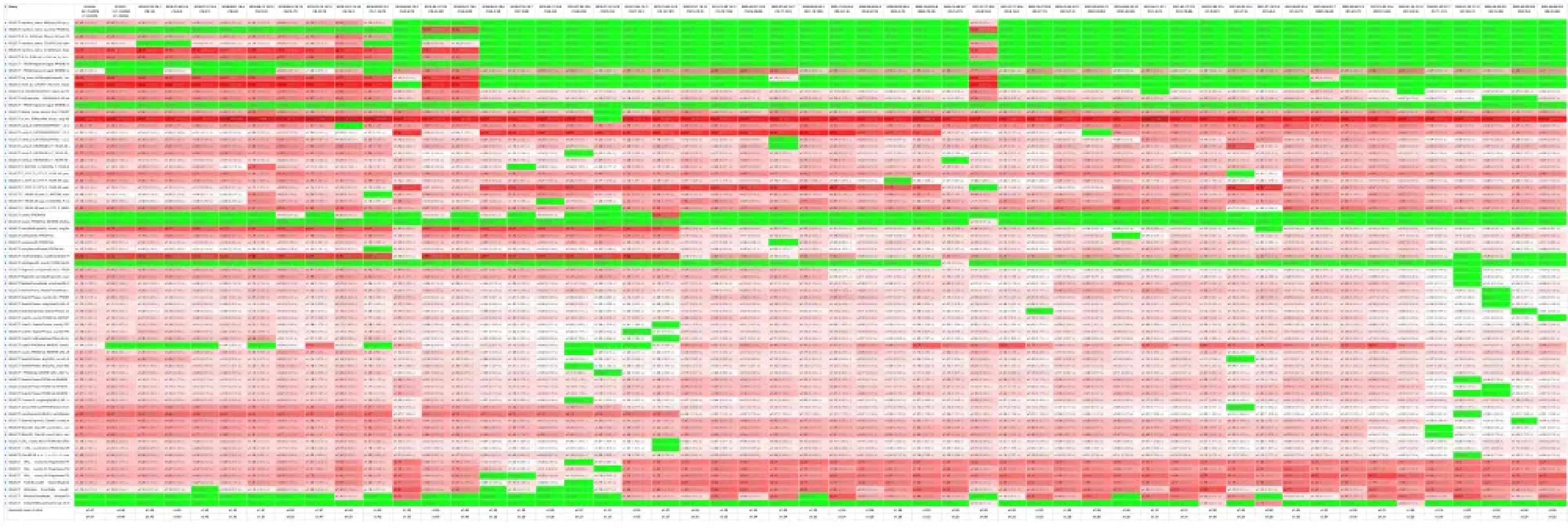
```
inline UInt64 bytes64MaskToBits64Mask(const UInt8 * bytes64)
{
#if defined(__AVX512F__) && defined(__AVX512BW__)
    static const __m512i zero64 = _mm512_setzero_epi32();
    UInt64 res = _mm512_cmp_epi8_mask(_mm512_loadu_si512(reinterpret_cast<const __m512i *>(bytes64)), zero64, _MM_CMPINT_EQ);
#elif defined(__AVX__) && defined(__AVX2__)
    static const __m256i zero32 = _mm256_setzero_si256();
    UInt64 res =
        (static_cast<UInt64>(_mm256_movemask_epi8(_mm256_cmpeq_epi8(
        _mm256_loadu_si256(reinterpret_cast<const __m256i *>(bytes64)), zero32))) & 0xffffffff)
        | (static_cast<UInt64>(_mm256_movemask_epi8(_mm256_cmpeq_epi8(
        _mm256_loadu_si256(reinterpret_cast<const __m256i *>(bytes64+32)), zero32))) << 32);
#elif defined(__SSE2__) && defined(__POPCNT__)
    static const __m128i zero16 = _mm_setzero_si128();
    UInt64 res =
        (static_cast<UInt64>(_mm_movemask_epi8(_mm_cmpeq_epi8(
        _mm_loadu_si128(reinterpret_cast<const __m128i *>(bytes64)), zero16))) & 0xffff)
        | ((static_cast<UInt64>(_mm_movemask_epi8(_mm_cmpeq_epi8(
        _mm_loadu_si128(reinterpret_cast<const __m128i *>(bytes64 + 16)), zero16))) << 16) & 0xffff0000)
        | ((static_cast<UInt64>(_mm_movemask_epi8(_mm_cmpeq_epi8(
        _mm_loadu_si128(reinterpret_cast<const __m128i *>(bytes64 + 32)), zero16))) << 32) & 0xffff00000000)
        | ((static_cast<UInt64>(_mm_movemask_epi8(_mm_cmpeq_epi8(
        _mm_loadu_si128(reinterpret_cast<const __m128i *>(bytes64 + 48)), zero16))) << 48) & 0xffff000000000000);
```

I am one of the authors for much code of this sort

ClickHouse is column based, it's basically a huge array of bytes. SIMD is great



CLICKHOUSE IS 28% FASTER OVER 4 YEARS



https://clickhouse.com/blog/clickhouse-over-the-years-with-benchmarks

IT'S HARD TO LEARN SIMD

- Over a decade Intel was publishing SIMD guides
- Arm did nothing

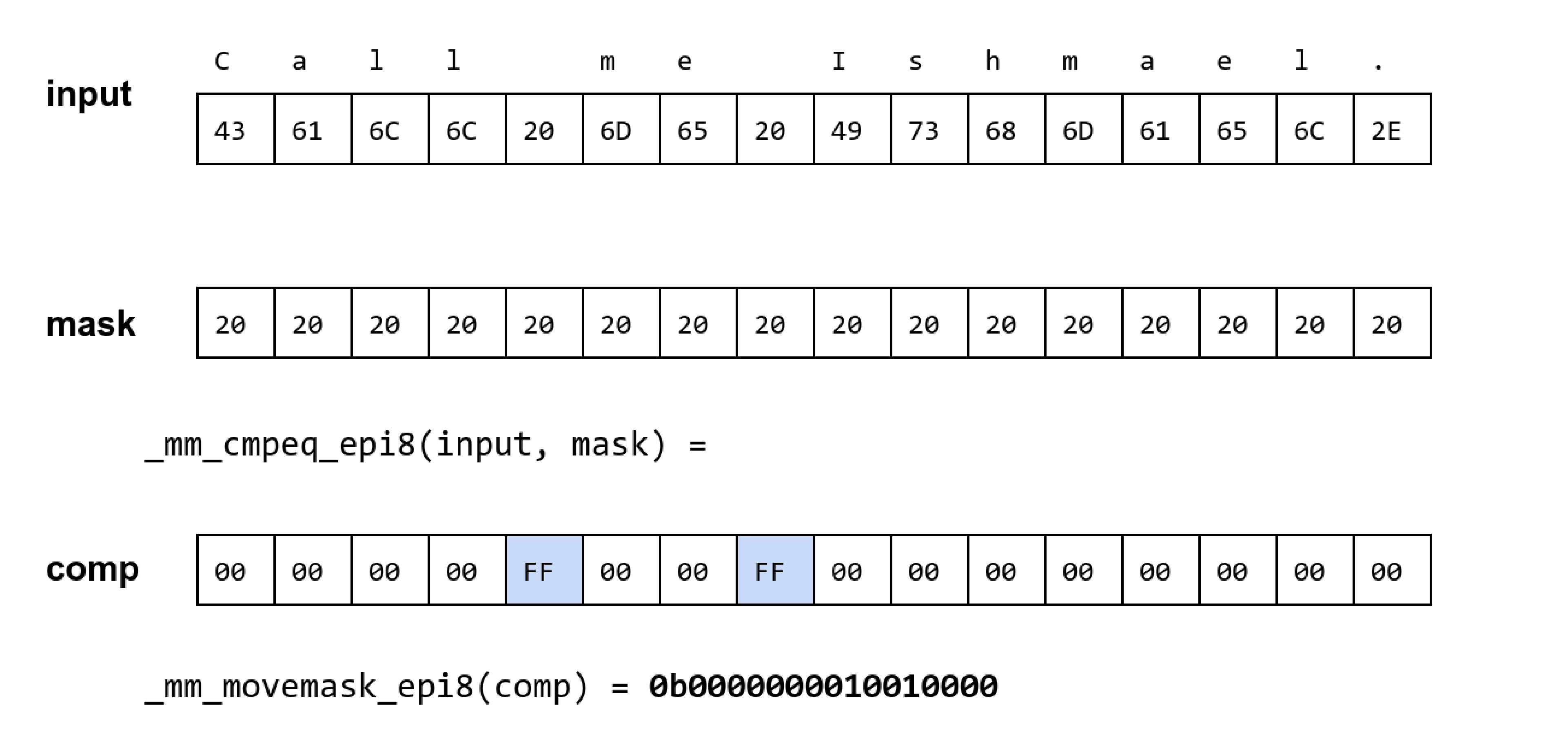
WE FIXED THE GLITCH!

- We learned Arm NEON SIMD by heart
- Will publish some software guides soon

SITUATION IS APPALLING

SITUATION IS APPALLING

READY?



	C	a	l	l		m	e		I	s	h	m	a	e	l	.
input	43	61	6C	6C	20	6D	65	20	49	73	68	6D	61	65	6C	2E
mask	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20

```
_mm_cmpeq_epi8(input, mask) =
```

comp	00	00	00	00	FF	00	00	FF	00	00	00	00	00	00	00	00

```
_mm_movemask_epi8(comp) = 0b0000000010010000
```

PMOVMSKB is an x86 instruction to move from vector to scalar. 1 cycle

ARM DOES NOT HAVE ANYTHING LIKE THAT

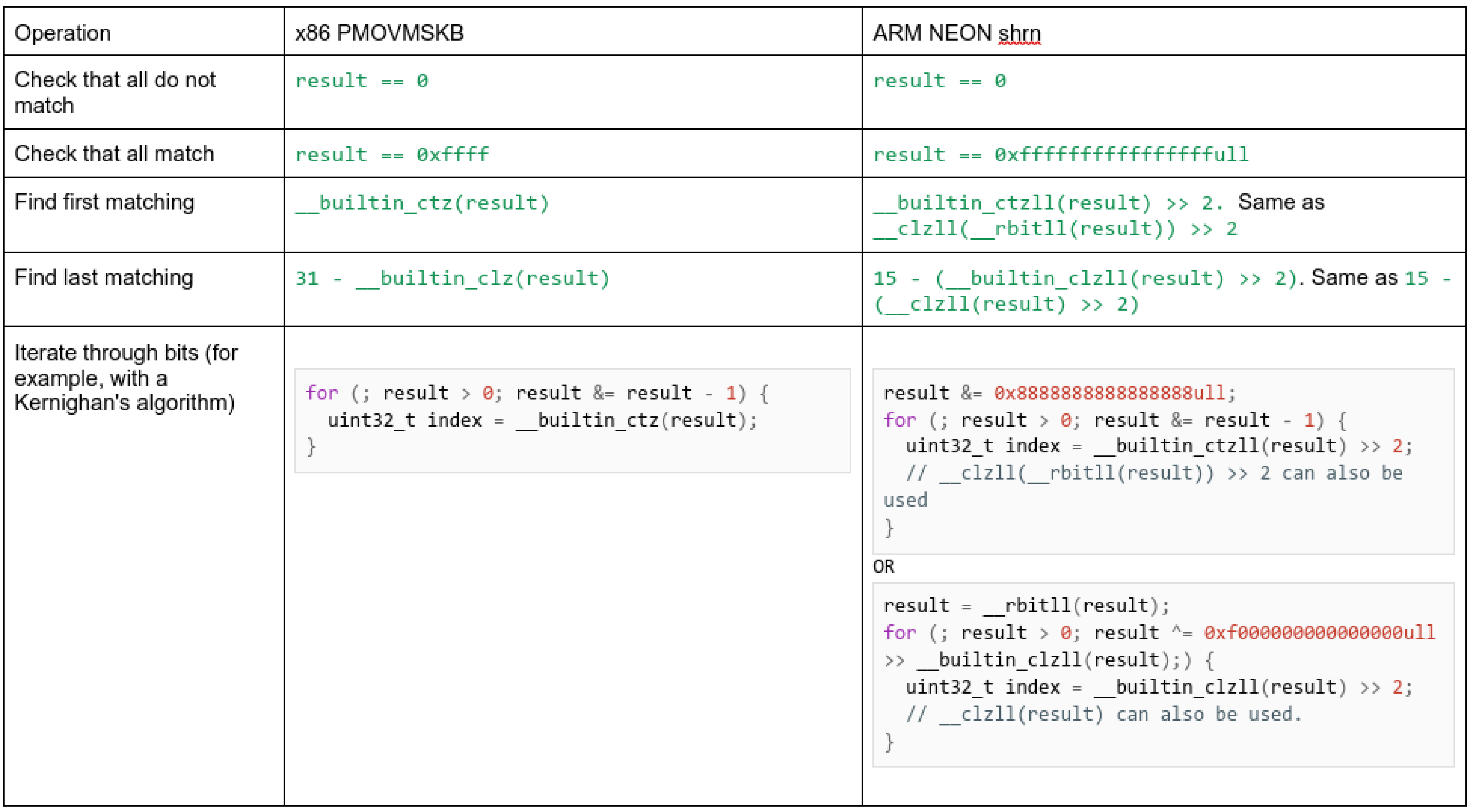
Migration emulation takes 12 cycles!

```
int _mm_movemask_epi8(__m128i a) {
    uint8x16_t input = vreinterpretq_u8_m128i(a);
    uint16x8_t high_bits =
        vreinterpretq_u16_u8(vshrq_n_u8(input, 7));
    uint32x4_t paired16 =
        vreinterpretq_u32_u16(
            vsraq_n_u16(high_bits, high_bits, 7));
    uint64x2_t paired32 =
        vreinterpretq_u64_u32(
            vsraq_n_u32(paired16, paired16, 14));
    uint8x16_t paired64 =
        vreinterpretq_u8_u64(
            vsraq_n_u64(paired32, paired32, 28));
    return vgetq_lane_u8(paired64, 0) |
            ((int) vgetq_lane_u8(paired64, 8) << 8);
```

WE FOUND SIMILAR WAYS TO EMULATE THROUGH INSTRUCTION NO ONE CARED BEFORE: SHIFT RIGHT AND NARROW

00000000 00000000 00000000 11111111 11111111 00000000 11111111 11111111 00000000 11111111 00000000 11111111 00000000 00000000 11111111 11111111

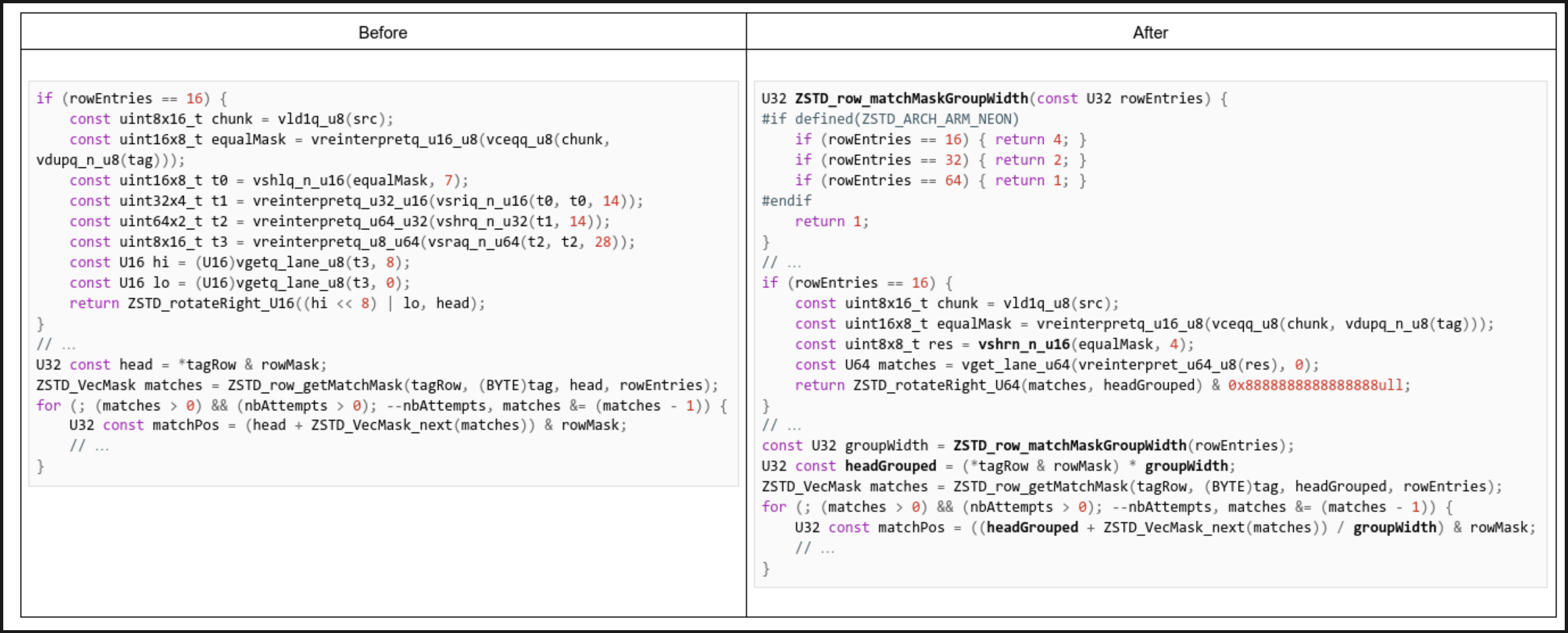
It's almost a bit mask but with groups of 4



Operation	x86 PMOVMSKB	ARM NEON shrn
Check that all do not match	result == 0	result == 0
Check that all match	result == 0xffff	result == 0xffffffffffffffffull
Find first matching	__builtin_ctz(result)	__builtin_ctzll(result) >> 2. Same as __clzll(__rbitll(result)) >> 2
Find last matching	31 - __builtin_clz(result)	15 - (__builtin_clzll(result) >> 2). Same as 15 - (__clzll(result) >> 2)

Iterate through bits (for example, with a Kernighan's algorithm)	for (; result > 0; result &= result - 1) {
	  uint32_t index = __builtin_ctz(result);
	}
		result &= 0x8888888888888888ull;
		for (; result > 0; result &= result - 1) {
		  uint32_t index = __builtin_ctzll(result) >> 2;
		  // __clzll(__rbitll(result)) >> 2 can also be used
		}
		OR
		result = __rbitll(result);
		for (; result > 0; result ^= 0xf000000000000000ull >> __builtin_clzll(result);) {
		  uint32_t index = __builtin_clzll(result) >> 2;
		  // __clzll(result) can also be used.
		}

RESULTS



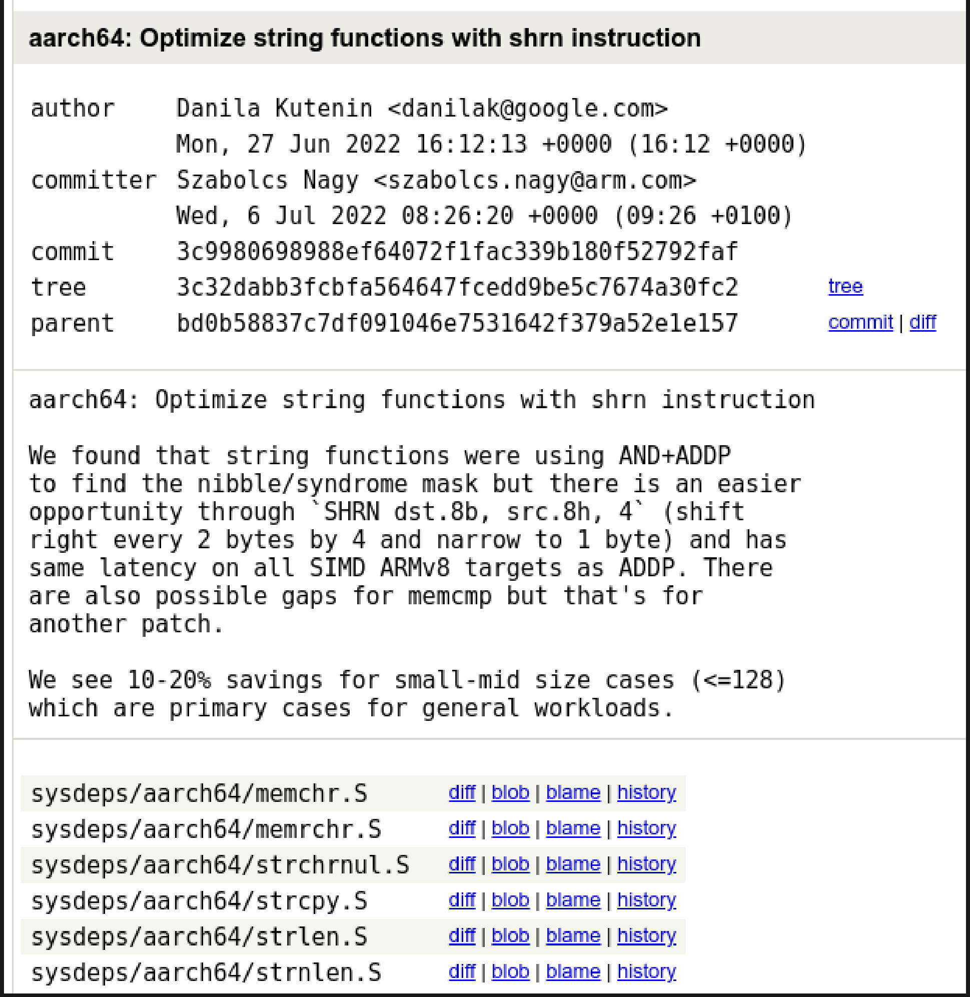
Before

```c
if (rowEntries == 16) {
    const uint8x16_t chunk = vld1q_u8(src);
    const uint16x8_t equalMask = vreinterpretq_u16_u8(vceqq_u8(chunk,
vdupq_n_u8(tag)));
    const uint16x8_t t0 = vshlq_n_u16(equalMask, 7);
    const uint32x4_t t1 = vreinterpretq_u32_u16(vsriq_n_u16(t0, t0, 14));
    const uint64x2_t t2 = vreinterpretq_u64_u32(vshrq_n_u32(t1, 14));
    const uint8x16_t t3 = vreinterpretq_u8_u64(vsraq_n_u64(t2, t2, 28));
    const U16 hi = (U16)vgetq_lane_u8(t3, 8);
    const U16 lo = (U16)vgetq_lane_u8(t3, 0);
    return ZSTD_rotateRight_U16((hi << 8) | lo, head);
}
// ...
U32 const head = *tagRow & rowMask;
ZSTD_VecMask matches = ZSTD_row_getMatchMask(tagRow, (BYTE)tag, head, rowEntries);
for (; (matches > 0) && (nbAttempts > 0); --nbAttempts, matches &= (matches - 1)) {
    U32 const matchPos = (head + ZSTD_VecMask_next(matches)) & rowMask;
    // ...
}
```

After

```c
U32 ZSTD_row_matchMaskGroupWidth(const U32 rowEntries) {
#if defined(ZSTD_ARCH_ARM_NEON)
    if (rowEntries == 16) { return 4; }
    if (rowEntries == 32) { return 2; }
    if (rowEntries == 64) { return 1; }
#endif
    return 1;
}
// ...
if (rowEntries == 16) {
    const uint8x16_t chunk = vld1q_u8(src);
    const uint16x8_t equalMask = vreinterpretq_u16_u8(vceqq_u8(chunk, vdupq_n_u8(tag)));
    const uint8x8_t res = vshrn_n_u16(equalMask, 4);
    const U64 matches = vget_lane_u64(vreinterpret_u64_u8(res), 0);
    return ZSTD_rotateRight_U64(matches, headGrouped) & 0x8888888888888888ull;
}
// ...
const U32 groupWidth = ZSTD_row_matchMaskGroupWidth(rowEntries);
U32 const headGrouped = (*tagRow & rowMask) * groupWidth;
ZSTD_VecMask matches = ZSTD_row_getMatchMask(tagRow, (BYTE)tag, headGrouped, rowEntries);
for (; (matches > 0) && (nbAttempts > 0); --nbAttempts, matches &= (matches - 1)) {
    U32 const matchPos = ((headGrouped + ZSTD_VecMask_next(matches)) / groupWidth) & rowMask;
    // ...
}
```

ZSTD 5% for compression


aarch64: Optimize string functions with shrn instruction

```
author     Danila Kutenin <danilak@google.com>
           Mon, 27 Jun 2022 16:12:13 +0000 (16:12 +0000)
committer  Szabolcs Nagy <szabolcs.nagy@arm.com>
           Wed, 6 Jul 2022 08:26:20 +0000 (09:26 +0100)
commit     3c9980698988ef64072f1fac339b180f52792faf
tree       3c32dabb3fcbfa564647fcedd9be5c7674a30fc2        tree
parent     bd0b58837c7df091046e7531642f379a52e1e157        commit | diff
```

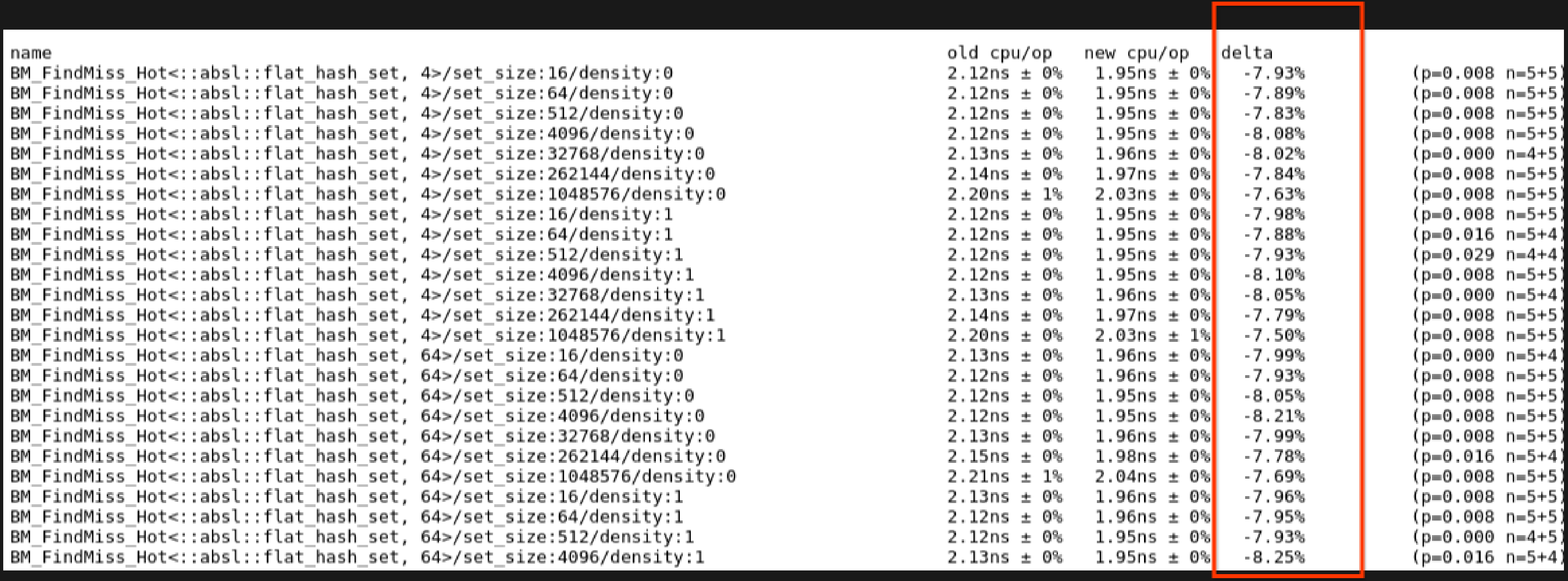
```
aarch64: Optimize string functions with shrn instruction

We found that string functions were using AND+ADDP
to find the nibble/syndrome mask but there is an easier
opportunity through `SHRN dst.8b, src.8h, 4` (shift
right every 2 bytes by 4 and narrow to 1 byte) and has
same latency on all SIMD ARMv8 targets as ADDP. There
are also possible gaps for memcmp but that's for
another patch.

We see 10-20% savings for small-mid size cases (<=128)
which are primary cases for general workloads.
```

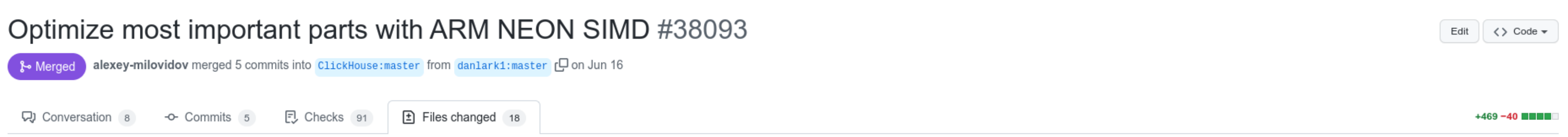
```
sysdeps/aarch64/memchr.S      diff | blob | blame | history
sysdeps/aarch64/memrchr.S     diff | blob | blame | history
sysdeps/aarch64/strchrnul.S   diff | blob | blame | history
sysdeps/aarch64/strcpy.S      diff | blob | blame | history
sysdeps/aarch64/strlen.S      diff | blob | blame | history
sysdeps/aarch64/strnlen.S     diff | blob | blame | history
```

10% for byte search (yes, C standard library)



```
name                                                                                     old cpu/op   new cpu/op   delta
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:16/density:0                          2.12ns ± 0%   1.95ns ± 0%   -7.93%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:64/density:0                          2.12ns ± 0%   1.95ns ± 0%   -7.89%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:512/density:0                         2.12ns ± 0%   1.95ns ± 0%   -7.83%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:4096/density:0                        2.12ns ± 0%   1.95ns ± 0%   -8.08%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:32768/density:0                       2.13ns ± 0%   1.96ns ± 0%   -8.02%          (p=0.000 n=4+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:262144/density:0                      2.14ns ± 0%   1.97ns ± 0%   -7.84%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:1048576/density:0                     2.20ns ± 1%   2.03ns ± 0%   -7.63%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:16/density:1                          2.12ns ± 0%   1.95ns ± 0%   -7.98%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:64/density:1                          2.12ns ± 0%   1.95ns ± 0%   -7.88%          (p=0.016 n=5+4)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:512/density:1                         2.12ns ± 0%   1.95ns ± 0%   -7.93%          (p=0.029 n=4+4)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:4096/density:1                        2.12ns ± 0%   1.95ns ± 0%   -8.10%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:32768/density:1                       2.13ns ± 0%   1.96ns ± 0%   -8.05%          (p=0.000 n=5+4)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:262144/density:1                      2.14ns ± 0%   1.97ns ± 0%   -7.79%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 4>/set_size:1048576/density:1                     2.20ns ± 0%   2.03ns ± 1%   -7.50%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:16/density:0                         2.13ns ± 0%   1.96ns ± 0%   -7.99%          (p=0.000 n=5+4)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:64/density:0                         2.12ns ± 0%   1.96ns ± 0%   -7.93%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:512/density:0                        2.12ns ± 0%   1.95ns ± 0%   -8.05%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:4096/density:0                       2.12ns ± 0%   1.95ns ± 0%   -8.21%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:32768/density:0                      2.13ns ± 0%   1.96ns ± 0%   -7.99%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:262144/density:0                     2.15ns ± 0%   1.98ns ± 0%   -7.78%          (p=0.016 n=5+4)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:1048576/density:0                    2.21ns ± 1%   2.04ns ± 0%   -7.69%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:16/density:1                         2.13ns ± 0%   1.96ns ± 0%   -7.96%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:64/density:1                         2.12ns ± 0%   1.96ns ± 0%   -7.95%          (p=0.008 n=5+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:512/density:1                        2.12ns ± 0%   1.95ns ± 0%   -7.93%          (p=0.000 n=4+5)
BM_FindMiss_Hot<::absl::flat_hash_set, 64>/set_size:4096/density:1                       2.13ns ± 0%   1.95ns ± 0%   -8.25%          (p=0.016 n=5+4)
```

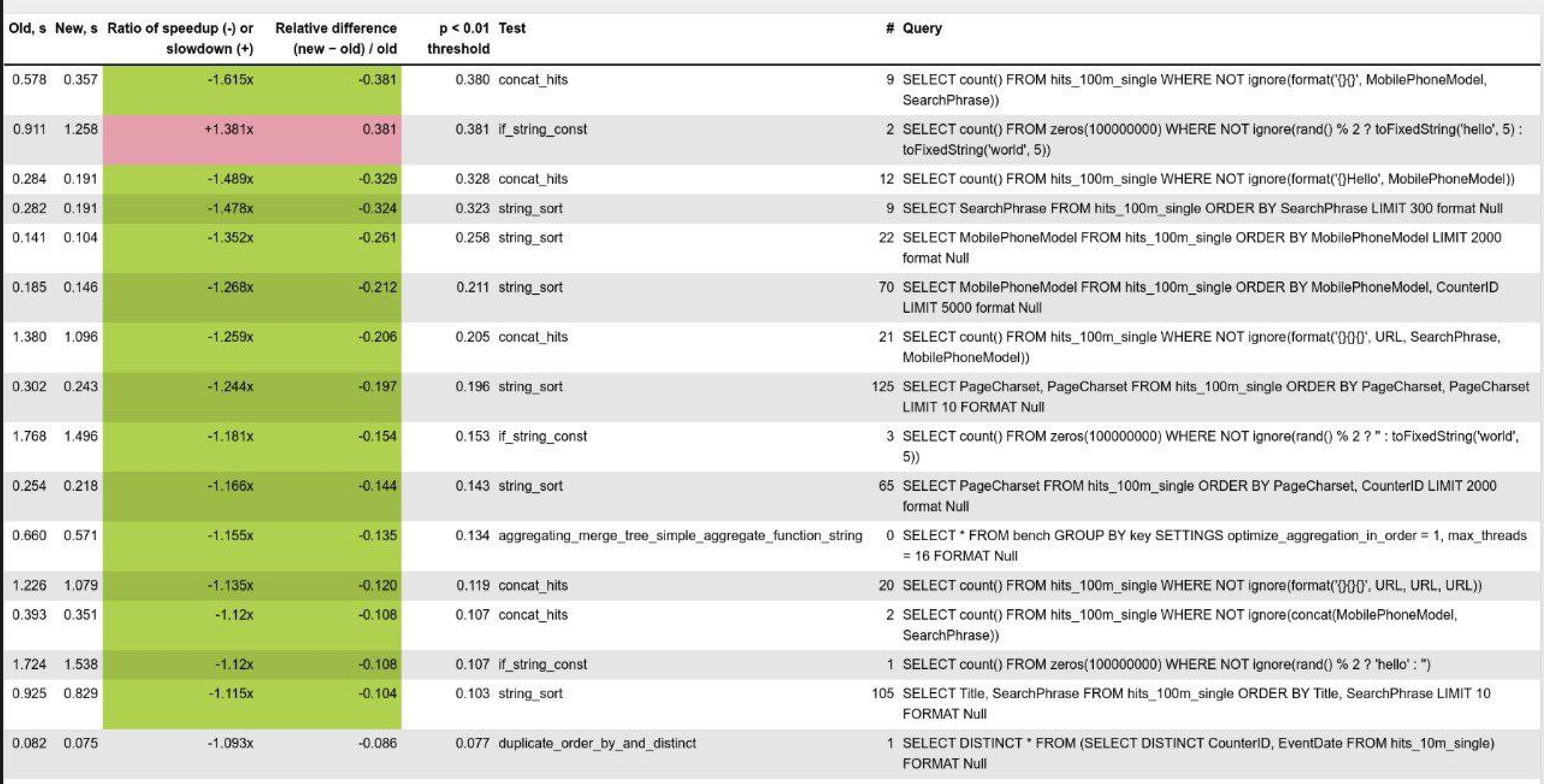
3-8% for hashtables



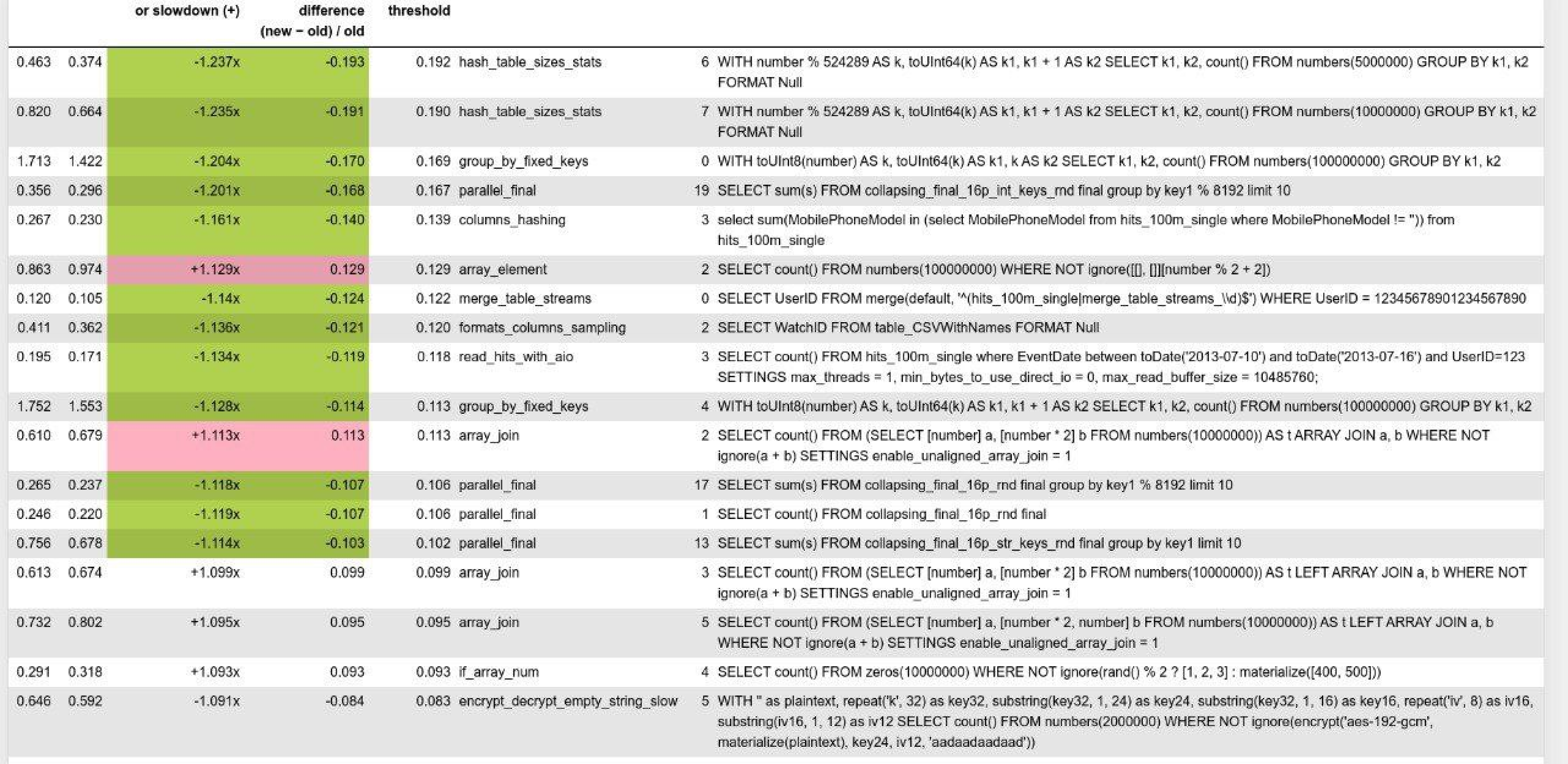
Optimize most important parts with ARM NEON SIMD #38093

Merged **alexey-milovidov** merged 5 commits into `ClickHouse:master` from `danlark1:master` on Jun 16

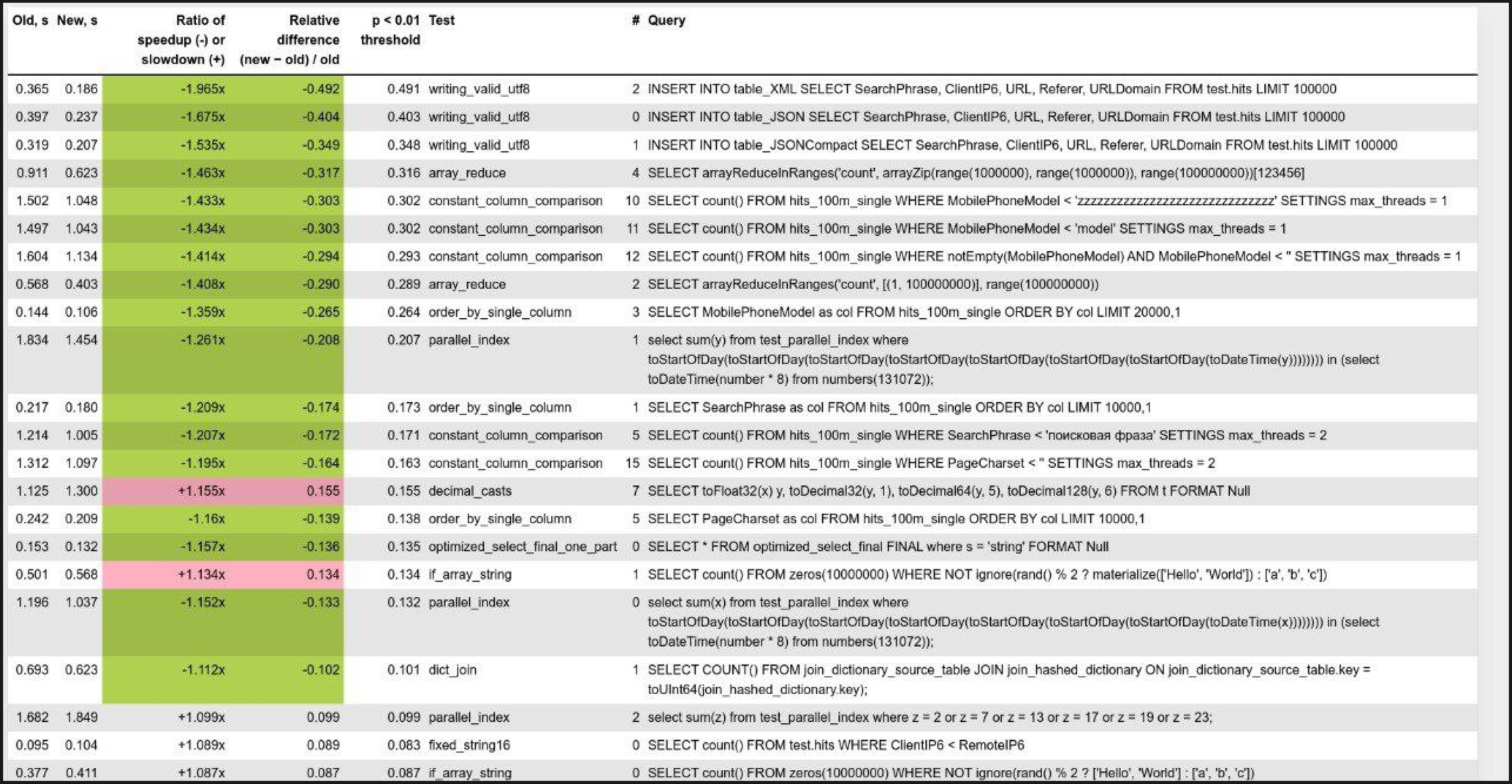
Lots of places in ClickHouse PR #38093



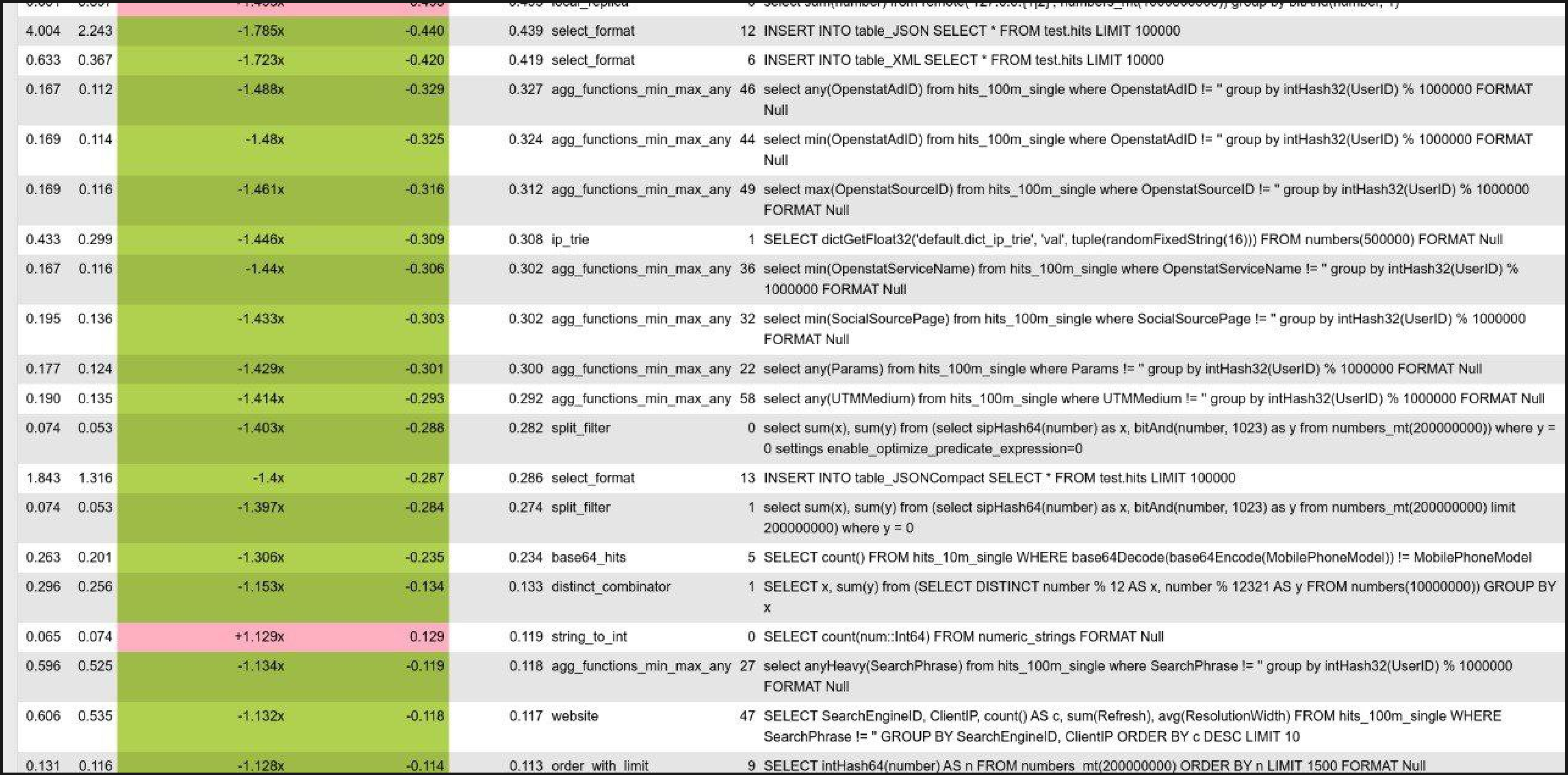
Old, s	New, s	Ratio of speedup (-) or slowdown (+)	Relative difference (new − old) / old	p < 0.01 threshold	Test	#	Query
0.578	0.357	-1.615x	-0.381	0.380	concat_hits	9	SELECT count() FROM hits_100m_single WHERE NOT ignore(format('{}{}', MobilePhoneModel, SearchPhrase))
0.911	1.258	+1.381x	0.381	0.381	if_string_const	2	SELECT count() FROM zeros(100000000) WHERE NOT ignore(rand() % 2 ? toFixedString('hello', 5) : toFixedString('world', 5))
0.284	0.191	-1.489x	-0.329	0.328	concat_hits	12	SELECT count() FROM hits_100m_single WHERE NOT ignore(format('{}Hello', MobilePhoneModel))
0.282	0.191	-1.478x	-0.324	0.323	string_sort	9	SELECT SearchPhrase FROM hits_100m_single ORDER BY SearchPhrase LIMIT 300 format Null
0.141	0.104	-1.352x	-0.261	0.258	string_sort	22	SELECT MobilePhoneModel FROM hits_100m_single ORDER BY MobilePhoneModel LIMIT 2000 format Null
0.185	0.146	-1.268x	-0.212	0.211	string_sort	70	SELECT MobilePhoneModel FROM hits_100m_single ORDER BY MobilePhoneModel, CounterID LIMIT 5000 format Null
1.380	1.096	-1.259x	-0.206	0.205	concat_hits	21	SELECT count() FROM hits_100m_single WHERE NOT ignore(format('{}{}{}', URL, SearchPhrase, MobilePhoneModel))
0.302	0.243	-1.244x	-0.197	0.196	string_sort	125	SELECT PageCharset, PageCharset FROM hits_100m_single ORDER BY PageCharset, PageCharset LIMIT 10 FORMAT Null
1.768	1.496	-1.181x	-0.154	0.153	if_string_const	3	SELECT count() FROM zeros(100000000) WHERE NOT ignore(rand() % 2 ? '' : toFixedString('world', 5))
0.254	0.218	-1.166x	-0.144	0.143	string_sort	65	SELECT PageCharset FROM hits_100m_single ORDER BY PageCharset, CounterID LIMIT 2000 format Null
0.660	0.571	-1.155x	-0.135	0.134	aggregating_merge_tree_simple_aggregate_function_string	0	SELECT * FROM bench GROUP BY key SETTINGS optimize_aggregation_in_order = 1, max_threads = 16 FORMAT Null
1.226	1.079	-1.135x	-0.120	0.119	concat_hits	20	SELECT count() FROM hits_100m_single WHERE NOT ignore(format('{}{}{}', URL, URL, URL))
0.393	0.351	-1.12x	-0.108	0.107	concat_hits	2	SELECT count() FROM hits_100m_single WHERE NOT ignore(concat(MobilePhoneModel, SearchPhrase))
1.724	1.538	-1.12x	-0.108	0.107	if_string_const	1	SELECT count() FROM zeros(100000000) WHERE NOT ignore(rand() % 2 ? 'hello' : '')
0.925	0.829	-1.115x	-0.104	0.103	string_sort	105	SELECT Title, SearchPhrase FROM hits_100m_single ORDER BY Title, SearchPhrase LIMIT 10 FORMAT Null
0.082	0.075	-1.093x	-0.086	0.077	duplicate_order_by_and_distinct	1	SELECT DISTINCT * FROM (SELECT DISTINCT CounterID, EventDate FROM hits_10m_single) FORMAT Null



		or slowdown (+)	difference	threshold			
			(new − old) / old				
0.463	0.374	-1.237x	-0.193	0.192	hash_table_sizes_stats	6	WITH number % 524289 AS k, toUInt64(k) AS k1, k1 + 1 AS k2 SELECT k1, k2, count() FROM numbers(5000000) GROUP BY k1, k2 FORMAT Null
0.820	0.664	-1.235x	-0.191	0.190	hash_table_sizes_stats	7	WITH number % 524289 AS k, toUInt64(k) AS k1, k1 + 1 AS k2 SELECT k1, k2, count() FROM numbers(10000000) GROUP BY k1, k2 FORMAT Null
1.713	1.422	-1.204x	-0.170	0.169	group_by_fixed_keys	0	WITH toUInt8(number) AS k, toUInt64(k) AS k1, k AS k2 SELECT k1, k2, count() FROM numbers(100000000) GROUP BY k1, k2
0.356	0.296	-1.201x	-0.168	0.167	parallel_final	19	SELECT sum(s) FROM collapsing_final_16p_int_keys_rnd final group by key1 % 8192 limit 10
0.267	0.230	-1.161x	-0.140	0.139	columns_hashing	3	select sum(MobilePhoneModel in (select MobilePhoneModel from hits_100m_single where MobilePhoneModel != '')) from hits_100m_single
0.863	0.974	+1.129x	0.129	0.129	array_element	2	SELECT count() FROM numbers(100000000) WHERE NOT ignore([[], []][number % 2 + 2])
0.120	0.105	-1.14x	-0.124	0.122	merge_table_streams	0	SELECT UserID FROM merge(default, '^(hits_100m_single|merge_table_streams_\\d)$') WHERE UserID = 12345678901234567890
0.411	0.362	-1.136x	-0.121	0.120	formats_columns_sampling	2	SELECT WatchID FROM table_CSVWithNames FORMAT Null
0.195	0.171	-1.134x	-0.119	0.118	read_hits_with_aio	3	SELECT count() FROM hits_100m_single where EventDate between toDate('2013-07-10') and toDate('2013-07-16') and UserID=123 SETTINGS max_threads = 1, min_bytes_to_use_direct_io = 0, max_read_buffer_size = 10485760;
1.752	1.553	-1.128x	-0.114	0.113	group_by_fixed_keys	4	WITH toUInt8(number) AS k, toUInt64(k) AS k1, k1 + 1 AS k2 SELECT k1, k2, count() FROM numbers(100000000) GROUP BY k1, k2
0.610	0.679	+1.113x	0.113	0.113	array_join	2	SELECT count() FROM (SELECT [number] a, [number * 2] b FROM numbers(10000000)) AS t ARRAY JOIN a, b WHERE NOT ignore(a + b) SETTINGS enable_unaligned_array_join = 1
0.265	0.237	-1.118x	-0.107	0.106	parallel_final	17	SELECT sum(s) FROM collapsing_final_16p_rnd final group by key1 % 8192 limit 10
0.246	0.220	-1.119x	-0.107	0.106	parallel_final	1	SELECT count() FROM collapsing_final_16p_rnd final
0.756	0.678	-1.114x	-0.103	0.102	parallel_final	13	SELECT sum(s) FROM collapsing_final_16p_str_keys_rnd final group by key1 limit 10
0.613	0.674	+1.099x	0.099	0.099	array_join	3	SELECT count() FROM (SELECT [number] a, [number * 2] b FROM numbers(10000000)) AS t LEFT ARRAY JOIN a, b WHERE NOT ignore(a + b) SETTINGS enable_unaligned_array_join = 1
0.732	0.802	+1.095x	0.095	0.095	array_join	5	SELECT count() FROM (SELECT [number] a, [number * 2, number] b FROM numbers(10000000)) AS t LEFT ARRAY JOIN a, b WHERE NOT ignore(a + b) SETTINGS enable_unaligned_array_join = 1
0.291	0.318	+1.093x	0.093	0.093	if_array_num	4	SELECT count() FROM zeros(10000000) WHERE NOT ignore(rand() % 2 ? [1, 2, 3] : materialize([400, 500]))
0.646	0.592	-1.091x	-0.084	0.083	encrypt_decrypt_empty_string_slow	5	WITH '' as plaintext, repeat('k', 32) as key32, substring(key32, 1, 24) as key24, substring(key32, 1, 16) as key16, repeat('iv', 8) as iv16, substring(iv16, 1, 12) as iv12 SELECT count() FROM numbers(2000000) WHERE NOT ignore(encrypt('aes-192-gcm', materialize(plaintext), key24, iv12, 'aadaadaadaad'))



Old, s	New, s	Ratio of speedup (-) or slowdown (+)	Relative difference (new − old) / old	p < 0.01 threshold	Test	#	Query
0.365	0.186	-1.965x	-0.492	0.491	writing_valid_utf8	2	INSERT INTO table_XML SELECT SearchPhrase, ClientIP6, URL, Referer, URLDomain FROM test.hits LIMIT 100000
0.397	0.237	-1.675x	-0.404	0.403	writing_valid_utf8	0	INSERT INTO table_JSON SELECT SearchPhrase, ClientIP6, URL, Referer, URLDomain FROM test.hits LIMIT 100000
0.319	0.207	-1.535x	-0.349	0.348	writing_valid_utf8	1	INSERT INTO table_JSONCompact SELECT SearchPhrase, ClientIP6, URL, Referer, URLDomain FROM test.hits LIMIT 100000
0.911	0.623	-1.463x	-0.317	0.316	array_reduce	4	SELECT arrayReduceInRanges('count', arrayZip(range(1000000), range(1000000)), range(100000000))[123456]
1.502	1.048	-1.433x	-0.303	0.302	constant_column_comparison	10	SELECT count() FROM hits_100m_single WHERE MobilePhoneModel < 'zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz' SETTINGS max_threads = 1
1.497	1.043	-1.434x	-0.303	0.302	constant_column_comparison	11	SELECT count() FROM hits_100m_single WHERE MobilePhoneModel < 'model' SETTINGS max_threads = 1
1.604	1.134	-1.414x	-0.294	0.293	constant_column_comparison	12	SELECT count() FROM hits_100m_single WHERE notEmpty(MobilePhoneModel) AND MobilePhoneModel < '' SETTINGS max_threads = 1
0.568	0.403	-1.408x	-0.290	0.289	array_reduce	2	SELECT arrayReduceInRanges('count', [(1, 100000000)], range(100000000))
0.144	0.106	-1.359x	-0.265	0.264	order_by_single_column	3	SELECT MobilePhoneModel as col FROM hits_100m_single ORDER BY col LIMIT 20000,1
1.834	1.454	-1.261x	-0.208	0.207	parallel_index	1	select sum(y) from test_parallel_index where toStartOfDay(toStartOfDay(toStartOfDay(toStartOfDay(toStartOfDay(toStartOfDay(toStartOfDay(toDateTime(y)))))))) in (select toDateTime(number * 8) from numbers(131072));
0.217	0.180	-1.209x	-0.174	0.173	order_by_single_column	1	SELECT SearchPhrase as col FROM hits_100m_single ORDER BY col LIMIT 10000,1
1.214	1.005	-1.207x	-0.172	0.171	constant_column_comparison	5	SELECT count() FROM hits_100m_single WHERE SearchPhrase < 'поисковая фраза' SETTINGS max_threads = 2
1.312	1.097	-1.195x	-0.164	0.163	constant_column_comparison	15	SELECT count() FROM hits_100m_single WHERE PageCharset < '' SETTINGS max_threads = 2
1.125	1.300	+1.155x	0.155	0.155	decimal_casts	7	SELECT toFloat32(x) y, toDecimal32(y, 1), toDecimal64(y, 5), toDecimal128(y, 6) FROM t FORMAT Null
0.242	0.209	-1.16x	-0.139	0.138	order_by_single_column	5	SELECT PageCharset as col FROM hits_100m_single ORDER BY col LIMIT 10000,1
0.153	0.132	-1.157x	-0.136	0.135	optimized_select_final_one_part	0	SELECT * FROM optimized_select_final FINAL where s = 'string' FORMAT Null
0.501	0.568	+1.134x	0.134	0.134	if_array_string	1	SELECT count() FROM zeros(10000000) WHERE NOT ignore(rand() % 2 ? materialize(['Hello', 'World']) : ['a', 'b', 'c'])
1.196	1.037	-1.152x	-0.133	0.132	parallel_index	0	select sum(x) from test_parallel_index where toStartOfDay(toStartOfDay(toStartOfDay(toStartOfDay(toStartOfDay(toStartOfDay(toStartOfDay(toDateTime(x)))))))) in (select toDateTime(number * 8) from numbers(131072));
0.693	0.623	-1.112x	-0.102	0.101	dict_join	1	SELECT COUNT() FROM join_dictionary_source_table JOIN join_hashed_dictionary ON join_dictionary_source_table.key = toUInt64(join_hashed_dictionary.key);
1.682	1.849	+1.099x	0.099	0.099	parallel_index	2	select sum(z) from test_parallel_index where z = 2 or z = 7 or z = 13 or z = 17 or z = 19 or z = 23;
0.095	0.104	+1.089x	0.089	0.083	fixed_string16	0	SELECT count() FROM test.hits WHERE ClientIP6 < RemoteIP6
0.377	0.411	+1.087x	0.087	0.087	if_array_string	0	SELECT count() FROM zeros(10000000) WHERE NOT ignore(rand() % 2 ? ['Hello', 'World'] : ['a', 'b', 'c'])



4.004	2.243	-1.785x	-0.440	0.439	select_format	12	INSERT INTO table_JSON SELECT * FROM test.hits LIMIT 100000
0.633	0.367	-1.723x	-0.420	0.419	select_format	6	INSERT INTO table_XML SELECT * FROM test.hits LIMIT 10000
0.167	0.112	-1.488x	-0.329	0.327	agg_functions_min_max_any	46	select any(OpenstatAdID) from hits_100m_single where OpenstatAdID != '' group by intHash32(UserID) % 1000000 FORMAT Null
0.169	0.114	-1.48x	-0.325	0.324	agg_functions_min_max_any	44	select min(OpenstatAdID) from hits_100m_single where OpenstatAdID != '' group by intHash32(UserID) % 1000000 FORMAT Null
0.169	0.116	-1.461x	-0.316	0.312	agg_functions_min_max_any	49	select max(OpenstatSourceID) from hits_100m_single where OpenstatSourceID != '' group by intHash32(UserID) % 1000000 FORMAT Null
0.433	0.299	-1.446x	-0.309	0.308	ip_trie	1	SELECT dictGetFloat32('default.dict_ip_trie', 'val', tuple(randomFixedString(16))) FROM numbers(500000) FORMAT Null
0.167	0.116	-1.44x	-0.306	0.302	agg_functions_min_max_any	36	select min(OpenstatServiceName) from hits_100m_single where OpenstatServiceName != '' group by intHash32(UserID) % 1000000 FORMAT Null
0.195	0.136	-1.433x	-0.303	0.302	agg_functions_min_max_any	32	select min(SocialSourcePage) from hits_100m_single where SocialSourcePage != '' group by intHash32(UserID) % 1000000 FORMAT Null
0.177	0.124	-1.429x	-0.301	0.300	agg_functions_min_max_any	22	select any(Params) from hits_100m_single where Params != '' group by intHash32(UserID) % 1000000 FORMAT Null
0.190	0.135	-1.414x	-0.293	0.292	agg_functions_min_max_any	58	select any(UTMMedium) from hits_100m_single where UTMMedium != '' group by intHash32(UserID) % 1000000 FORMAT Null
0.074	0.053	-1.403x	-0.288	0.282	split_filter	0	select sum(x), sum(y) from (select sipHash64(number) as x, bitAnd(number, 1023) as y from numbers_mt(200000000)) where y = 0 settings enable_optimize_predicate_expression=0
1.843	1.316	-1.4x	-0.287	0.286	select_format	13	INSERT INTO table_JSONCompact SELECT * FROM test.hits LIMIT 100000
0.074	0.053	-1.397x	-0.284	0.274	split_filter	1	select sum(x), sum(y) from (select sipHash64(number) as x, bitAnd(number, 1023) as y from numbers_mt(200000000) limit 200000000) where y = 0
0.263	0.201	-1.306x	-0.235	0.234	base64_hits	5	SELECT count() FROM hits_10m_single WHERE base64Decode(base64Encode(MobilePhoneModel)) != MobilePhoneModel
0.296	0.256	-1.153x	-0.134	0.133	distinct_combinator	1	SELECT x, sum(y) from (SELECT DISTINCT number % 12 AS x, number % 12321 AS y FROM numbers(10000000)) GROUP BY x
0.065	0.074	+1.129x	0.129	0.119	string_to_int	0	SELECT count(num::Int64) FROM numeric_strings FORMAT Null
0.596	0.525	-1.134x	-0.119	0.118	agg_functions_min_max_any	27	select anyHeavy(SearchPhrase) from hits_100m_single where SearchPhrase != '' group by intHash32(UserID) % 1000000 FORMAT Null
0.606	0.535	-1.132x	-0.118	0.117	website	47	SELECT SearchEngineID, ClientIP, count() AS c, sum(Refresh), avg(ResolutionWidth) FROM hits_100m_single WHERE SearchPhrase != '' GROUP BY SearchEngineID, ClientIP ORDER BY c DESC LIMIT 10
0.131	0.116	-1.128x	-0.114	0.113	order_with_limit	9	SELECT intHash64(number) AS n FROM numbers_mt(200000000) ORDER BY n LIMIT 1500 FORMAT Null

Overall: 1-1.5% for all queries. 10-15% for string processing. Also: compiler flags, better branching, etc.

Future: SVE, more optimizations, software guides. Gap in software is huge

Thanks

Twitter @Danlark1

LinkedIn @Danlark1

Telegram @Danlark

This presentation https://danlark1.github.io/clickhouse-arm/