Fastest Hash for lookuping

The author: Georgi 'Sanmayce' Marinov, rendered by PixAI.art from an original photo

Testfile: Fedora-Workstation-Live-42-1.1.x86_64.iso, 2,398,523,392 bytes
CPU: 11th Gen Intel Core i7-1185G7-UP3 @max 4.80GHz
Laptop: Dell Latitude 7420
RAM: 8x4GB, Form Factor: Row Of Chips, Type: LPDDR4, Rank: 2, Configured Memory Speed: 4267 MT/s
OS: Linux Fedora, as superuser in performance mode with maximum niceness
ThrottlingWISE: It took 647 seconds to hash (and LOOKUP into 512 MB hashtable, 32bit big, each bit being a slot) the 8192 BBs (Building-Blocks) at each position with the Fastest hash 'Pippip_AES_TriXZi_Mikayla_ZMMmax' i.e. roughly 8192 x 2398523392 bytes =~ 18,299 GB or 18299/647 =~ 28 GiB/s hash-n-lookup

Hasher	for 32 bytes	for 512 bytes	for 8 KB	Collisions for 8 KB	
Pippip_AES_TriXZi_Mikayla	6.088	31.177	33.053	703,775,221	
Pippip_AES_TriXZi_Mikayla_ZMMmax	6.777	37.546	70.147	703,797,895	! RIDONKULOUS !
wyhash f4	5.299	15.711	19.332	703,804,152	
MeowHash	2.002	17.284	33.283	703,809,114	
komihash 5.27	4.659	12.251	14.746	703,811,325	
XXH3_64bits 0.8.3	5.087	20.178	43.571	703,819,509	

```
#include <stdlib.h>
#include <stdint.h>
#include <immintrin.h>
#define _PADr_KAZE(x, n) ( ((x) << (n))>>(n) )
#define _PAD_KAZE(x, n) ( ((x) << (n)) )

static inline uint64_t fold64(uint64_t A, uint64_t B) {
	#if defined(__GNUC__) || defined(__clang__)
		__uint128_t r = (__uint128_t)A * B;
		return (uint64_t)r ^ (uint64_t)(r >> 64);
	#else
		uint64_t hash64 = A ^ B;
		hash64 *= 1099511628211; //591798841;
		return hash64;
	#endif
}

#if defined(ZMM)
void FNV1A_Pippip_Yurii_OOO_128bit_AES_TriXZi_Mikayla_ZMMmax (const char *str, size_t wrdlen, uint32_t seed, void *output) {
    __m128i chunkA;
    __m128i chunkB;
    __m128i stateMIX;
    uint64_t hashLH;
    uint64_t hashRH;
    stateMIX = _mm_set1_epi32( (uint32_t)wrdlen ^ seed );
    if (wrdlen > 8) {
        __m128i stateA = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i stateB = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i stateC = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        size_t Cycles, NDhead;
        if (wrdlen > 16) {
            Cycles = ((wrdlen - 1)>>5) + 1;
            NDhead = wrdlen - (Cycles<<4);
            if (Cycles & 1) {
                #pragma nounroll
                for(; Cycles--; str += 16) {
                    //_mm_prefetch(str+512, _MM_HINT_T0);
                    //_mm_prefetch(str+NDhead+512, _MM_HINT_T0);
                    chunkA = _mm_loadu_si128((__m128i *)(str));
                    stateA = _mm_aesenc_si128(stateA, chunkA);
                    chunkB = _mm_loadu_si128((__m128i *)(str+NDhead));
```

```
                    stateC = _mm_aesenc_si128(stateC, chunkA);
                    stateB = _mm_aesenc_si128(stateB, chunkB);
                    stateC = _mm_aesenc_si128(stateC, chunkB);
                }
            } else {
                if ( (Cycles & 3) == 0 ) {
                    // New codepath ZMM (-O3 -mavx512f -mvaes -mbmi2 -march=tigerlake -DZMM) [
                    Cycles = Cycles>>2;
                    __m128i stateA2;// = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
                    __m128i stateB2;// = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
                    __m128i stateC2;// = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
                    __m512i StateA_zmm = _mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL,
0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
                    __m512i StateB_zmm = _mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL,
0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
                    __m512i StateC_zmm = _mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL,
0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
                    #pragma nounroll
                    for(; Cycles--; str += 64) {
                        __m512i chunk_zmm = _mm512_loadu_si512((__m512i*)str);
                        __m512i chunk_zmm_offset = _mm512_loadu_si512((__m512i*)(str+NDhead));
                        _mm_prefetch(str+2048, _MM_HINT_T0);
                        _mm_prefetch(str+NDhead+2048, _MM_HINT_T0);
                        StateA_zmm = _mm512_aesenc_epi128(StateA_zmm, chunk_zmm);
                        StateB_zmm = _mm512_aesenc_epi128(StateB_zmm, chunk_zmm_offset);
                        StateC_zmm = _mm512_aesenc_epi128(StateC_zmm, chunk_zmm);
                        StateC_zmm = _mm512_aesenc_epi128(StateC_zmm, chunk_zmm_offset);
                    }
                    stateA2 = _mm512_extracti64x2_epi64(StateA_zmm, 0);
                    stateB2 = _mm512_extracti64x2_epi64(StateB_zmm, 0);
                    stateC2 = _mm512_extracti64x2_epi64(StateC_zmm, 0);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateA2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateB2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateC2);
                    stateA2 = _mm512_extracti64x2_epi64(StateA_zmm, 1);
                    stateB2 = _mm512_extracti64x2_epi64(StateB_zmm, 1);
                    stateC2 = _mm512_extracti64x2_epi64(StateC_zmm, 1);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateA2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateB2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateC2);
                    stateA2 = _mm512_extracti64x2_epi64(StateA_zmm, 2);
                    stateB2 = _mm512_extracti64x2_epi64(StateB_zmm, 2);
                    stateC2 = _mm512_extracti64x2_epi64(StateC_zmm, 2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateA2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateB2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateC2);
                    stateA2 = _mm512_extracti64x2_epi64(StateA_zmm, 3);
                    stateB2 = _mm512_extracti64x2_epi64(StateB_zmm, 3);
                    stateC2 = _mm512_extracti64x2_epi64(StateC_zmm, 3);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateA2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateB2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateC2);
                    // New codepath ZMM (-O3 -mavx512f -maes -mbmi2 -march=tigerlake -DZMM) ]
                } else {
                    Cycles = Cycles>>1;
                    __m128i stateA2 = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
                    __m128i stateB2 = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
                    __m128i stateC2 = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
                    #pragma nounroll
                    for(; Cycles--; str += 32) {
                        _mm_prefetch(str+512, _MM_HINT_T0);
                        _mm_prefetch(str+NDhead+512, _MM_HINT_T0);
                        chunkA = _mm_loadu_si128((__m128i *)(str));
                        __m128i chunkA2 = _mm_loadu_si128((__m128i *)(str+16));
                        stateA = _mm_aesenc_si128(stateA, chunkA);
                        stateA2 = _mm_aesenc_si128(stateA2, chunkA2);
                        chunkB = _mm_loadu_si128((__m128i *)(str+NDhead));
                        __m128i chunkB2 = _mm_loadu_si128((__m128i *)(str+NDhead+16));
                        stateC = _mm_aesenc_si128(stateC, chunkA);
                        stateB = _mm_aesenc_si128(stateB, chunkB);
                        stateC = _mm_aesenc_si128(stateC, chunkB);
                        stateC2 = _mm_aesenc_si128(stateC2, chunkA2);
                        stateB2 = _mm_aesenc_si128(stateB2, chunkB2);
                        stateC2 = _mm_aesenc_si128(stateC2, chunkB2);
                    }
                    stateMIX = _mm_aesenc_si128(stateMIX, stateA2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateB2);
                    stateMIX = _mm_aesenc_si128(stateMIX, stateC2);
                }
            }
        } else { // 9..16
            NDhead = wrdlen - (1<<3);
            hashLH = (*(uint64_t *)(str));
            hashRH = (*(uint64_t *)(str+NDhead));
            chunkA = _mm_set_epi64x(hashLH, hashLH);
            stateA = _mm_aesenc_si128(stateA, chunkA);
            chunkB = _mm_set_epi64x(hashRH, hashRH);
            stateC = _mm_aesenc_si128(stateC, chunkA);
            stateB = _mm_aesenc_si128(stateB, chunkB);
            stateC = _mm_aesenc_si128(stateC, chunkB);
        }
        stateMIX = _mm_aesenc_si128(stateMIX, stateA);
        stateMIX = _mm_aesenc_si128(stateMIX, stateB);
        stateMIX = _mm_aesenc_si128(stateMIX, stateC);
    } else {
        hashLH = _PADr_KAZE(*(uint64_t *)(str+0), (8-wrdlen)<<3);
        hashRH = _PAD_KAZE(*(uint64_t *)(str+0), (8-wrdlen)<<3);
        chunkA = _mm_set_epi64x(hashLH, hashLH);
```

```
        chunkB = _mm_set_epi64x(hashRH, hashRH);
        stateMIX = _mm_aesenc_si128(stateMIX, chunkA);
        stateMIX = _mm_aesenc_si128(stateMIX, chunkB);
    }
    #ifdef eXdupe
        _mm_storeu_si128((__m128i *)output, stateMIX); // For eXdupe
    #else
        uint64_t result[2];
        _mm_storeu_si128((__m128i *)result, stateMIX);
        uint64_t hash64 = fold64(result[0], result[1]);
        *(uint32_t*)output = (uint32_t)hash64 ^ wrdlen;
    #endif
}
#endif

GCC: (GNU) 15.1.1 20250521 (Red Hat 15.1.1-2)
gcc -DZMM -static -O3 -march=tigerlake hashBBs_r9.c xxhash.c -o hashBBs_r9_GCC_tigerlakeZMM.elf
gcc -DZMM -O3 -march=tigerlake hashBBs_r9.c -o hashBBs_r9_GCC_tigerlakeZMM.elf.asm -S
hashBBs_r9_GCC_tigerlakeZMM.elf.asm:

FNV1A_Pippip_Yurii_OOO_128bit_AES_TriXZi_Mikayla_ZMMmax:
.LFB7514:
        .cfi_startproc
        xorl    %esi, %edx
        movq    %rdi, %rax
        vpbroadcastd    %edx, %xmm0
        vmovdqa %xmm0, %xmm1
        cmpq    $8, %rsi
        jbe     .L142
        cmpq    $16, %rsi
        ja      .L154
        vmovdqa .LC8(%rip), %xmm3
        vpbroadcastq    (%rdi), %xmm4
        vpbroadcastq    -8(%rdi,%rsi), %xmm2
        vaesenc %xmm4, %xmm3, %xmm4
        vaesenc %xmm2, %xmm3, %xmm3
        vaesenc %xmm2, %xmm4, %xmm2
.L146:
        vaesenc %xmm4, %xmm1, %xmm1
        vaesenc %xmm3, %xmm1, %xmm1
        vaesenc %xmm2, %xmm1, %xmm0
        jmp     .L150
        .p2align 4,,10
        .p2align 3
.L142:
        movl    $8, %edx
        movq    (%rdi), %rax
        subl    %esi, %edx
        sall    $3, %edx
        shlx    %rdx, %rax, %rax
        shrx    %rdx, %rax, %rdx
        vpbroadcastq    %rdx, %xmm1
        vaesenc %xmm1, %xmm0, %xmm0
        vpbroadcastq    %rax, %xmm1
        vaesenc %xmm1, %xmm0, %xmm0
.L150:
        vmovq   %xmm0, %rdx
        vpextrq $1, %xmm0, %rax
        mulx    %rax, %rax, %r10
        xorq    %r10, %rax
        xorl    %eax, %esi
        movl    %esi, (%rcx)
        ret
        .p2align 4,,10
        .p2align 3
.L154:
        leaq    -1(%rsi), %rdi
        movq    %rsi, %rdx
        shrq    $5, %rdi
        incq    %rdi
        movq    %rdi, %r8
        salq    $4, %r8
        subq    %r8, %rdx
        testb   $1, %dil
        jne     .L155
        testb   $3, %dil
        jne     .L147
        vmovdqa64       .LC9(%rip), %zmm2
        shrq    $2, %rdi
        addq    $2048, %rax
        salq    $6, %rdi
        addq    %rax, %rdi
        vmovdqa64       %zmm2, %zmm7
        vmovdqa64       %zmm2, %zmm6
        .p2align 4,,10
        .p2align 3
.L148:
        vmovdqu64       -2048(%rax), %zmm3
        vmovdqu64       -2048(%rax,%rdx), %zmm1
        prefetcht0      (%rax)
        prefetcht0      (%rdx,%rax)
        addq    $64, %rax
        vaesenc %zmm3, %zmm2, %zmm2
        vaesenc %zmm1, %zmm7, %zmm4
        vaesenc %zmm3, %zmm6, %zmm5
        vaesenc %zmm1, %zmm2, %zmm1
        vmovdqa64       %zmm5, %zmm6
        vmovdqa64       %zmm4, %zmm7
```

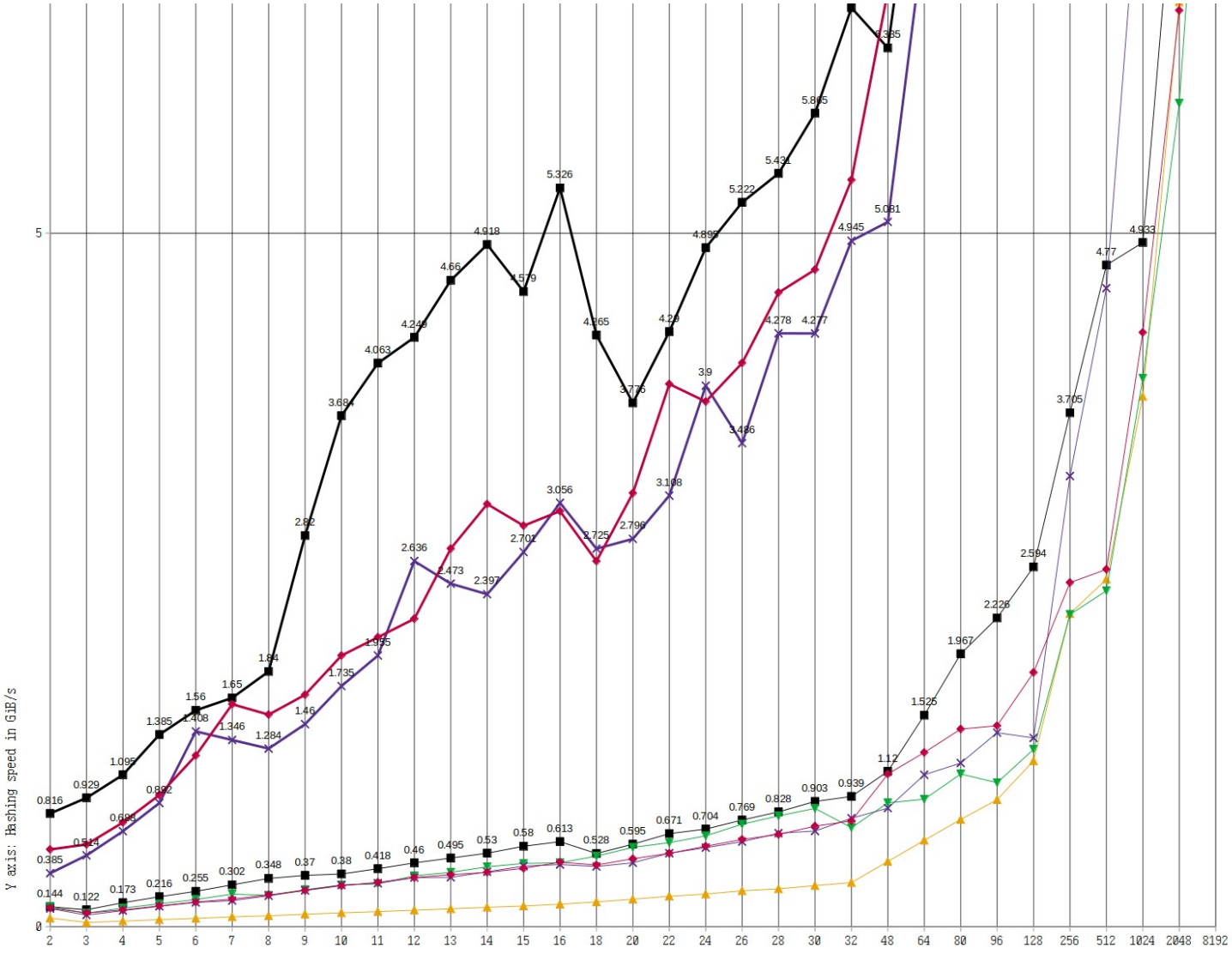
```
        vmovdqa64       %zmm1, %zmm2
        cmpq    %rdi, %rax
        jne     .L148
        vaesenc %xmm5, %xmm0, %xmm0
        vextracti64x2   $1, %zmm5, %xmm6
        vextracti64x2   $1, %zmm4, %xmm3
        vaesenc %xmm4, %xmm0, %xmm0
        vextracti64x2   $1, %zmm1, %xmm2
        vaesenc %xmm1, %xmm0, %xmm0
        vaesenc %xmm6, %xmm0, %xmm0
        vextracti64x2   $2, %zmm5, %xmm6
        vextracti64x2   $3, %zmm5, %xmm5
        vaesenc %xmm3, %xmm0, %xmm0
        vextracti64x2   $2, %zmm4, %xmm3
        vextracti64x2   $3, %zmm4, %xmm4
        vaesenc %xmm2, %xmm0, %xmm0
        vextracti64x2   $2, %zmm1, %xmm2
        vextracti64x2   $3, %zmm1, %xmm1
        vaesenc %xmm6, %xmm0, %xmm0
        vaesenc %xmm3, %xmm0, %xmm0
        vaesenc %xmm2, %xmm0, %xmm0
        vmovdqa .LC8(%rip), %xmm2
        vaesenc %xmm5, %xmm0, %xmm0
        vaesenc %xmm4, %xmm0, %xmm0
        vmovdqa %xmm2, %xmm3
        vmovdqa %xmm2, %xmm4
        vaesenc %xmm1, %xmm0, %xmm1
        vzeroupper
        jmp     .L146
        .p2align 4,,10
        .p2align 3
.L155:
        vmovdqa .LC8(%rip), %xmm2
        leaq    (%rax,%r8), %rdi
        vmovdqa %xmm2, %xmm3
        vmovdqa %xmm2, %xmm4
        .p2align 6
        .p2align 4,,10
        .p2align 3
.L145:
        vmovdqu (%rax), %xmm5
        vmovdqu (%rax,%rdx), %xmm0
        addq    $16, %rax
        vaesenc %xmm5, %xmm2, %xmm2
        vaesenc %xmm5, %xmm4, %xmm4
        vaesenc %xmm0, %xmm3, %xmm3
        vaesenc %xmm0, %xmm2, %xmm2
        cmpq    %rdi, %rax
        jne     .L145
        jmp     .L146
        .p2align 4,,10
        .p2align 3
.L147:
        vmovdqa .LC8(%rip), %xmm1
        shrq    %rdi
        addq    $512, %rax
        salq    $5, %rdi
        addq    %rax, %rdx
        vmovdqa %xmm1, %xmm9
        vmovdqa %xmm1, %xmm10
        vmovdqa %xmm1, %xmm2
        addq    %rax, %rdi
        vmovdqa %xmm1, %xmm3
        vmovdqa %xmm1, %xmm4
        .p2align 4,,10
        .p2align 3
.L149:
        vmovdqu -512(%rax), %xmm8
        vmovdqu -496(%rax), %xmm6
        prefetcht0      (%rax)
        addq    $32, %rax
        prefetcht0      (%rdx)
        vmovdqu -512(%rdx), %xmm7
        vmovdqu -496(%rdx), %xmm5
        addq    $32, %rdx
        vaesenc %xmm8, %xmm2, %xmm2
        vaesenc %xmm6, %xmm1, %xmm1
        vaesenc %xmm8, %xmm4, %xmm4
        vaesenc %xmm6, %xmm10, %xmm10
        vaesenc %xmm7, %xmm3, %xmm3
        vaesenc %xmm7, %xmm2, %xmm2
        vaesenc %xmm5, %xmm9, %xmm9
        vaesenc %xmm5, %xmm1, %xmm1
        cmpq    %rax, %rdi
        jne     .L149
        vaesenc %xmm10, %xmm0, %xmm0
        vaesenc %xmm9, %xmm0, %xmm0
        vaesenc %xmm1, %xmm0, %xmm1
        jmp     .L146
        .cfi_endproc
```

Testmachine: laptop Dell Latitude 7420, CPU 11th Gen Intel i7-1185G7 "Tiger Lake" Max Turbo: 4800 MHz, Max TDP: 28 W; LPDDR4 8x4 GB 4267 MT/s (Rank: 2)
TestOS: Linux Fedora, as superuser with niceness -20
Testcompiler: GCC 15.1.1, -static -O3 -march=tigerlake
Testfile: hashing 'Fedora-Workstation-Live-42-1.1.x86_64.iso' (2,398,523,392 bytes) at each position for orders 2..8192

```
// Let us compare the collisions for Fedora testdataset:
/*
Hasher,BB_Size,Total_Hashes,Unique_Hashes,Collisions,Time_s,Speed_GBs,SpeedRAW_GBs

Pippip_AES_TriXZi_Mikayla, 128,     2398523265,       1648394353,               750128912,        99.425,  2.876,   15.102
Pippip_AES_TriXZi_Mikayla, 256,     2398523137,       1650915581,               747607556,        133.876, 4.272,   25.848
Pippip_AES_TriXZi_Mikayla, 512,     2398522881,       1655018548,               743504333,        208.170, 5.494,   30.018
Pippip_AES_TriXZi_Mikayla, 1024,    2398522369,       1661965052,               736557317,        404.595, 5.654,   30.865
Pippip_AES_TriXZi_Mikayla, 2048,    2398521345,       1673883213,               724638132,        489.037, 9.355,   32.423
Pippip_AES_TriXZi_Mikayla, 8192,    2398515201,       1694739980,               703775221,        959.925, 19.063,  32.793

Pippip_AES_TriXZi_Mikayla_ZMMmax,   128,     2398523265,       1648406970,               750116295,        135.750, 2.106,   14.195 !
750116295-750128912= -12,617 i.e. better than Pippip_AES_TriXZi_Mikayla
Pippip_AES_TriXZi_Mikayla_ZMMmax,   256,     2398523137,       1650983359,               747539778,        138.650, 4.124,   18.868 !
747539778-747607556= -67,778 i.e. better than Pippip_AES_TriXZi_Mikayla
Pippip_AES_TriXZi_Mikayla_ZMMmax,   512,     2398522881,       1655034855,               743488026,        195.449, 5.852,   39.461 !
743488026-743504333= -16,307 i.e. better than Pippip_AES_TriXZi_Mikayla
Pippip_AES_TriXZi_Mikayla_ZMMmax,   1024,    2398522369,       1661953271,               736569098,        258.192, 8.859,   52.553 !
736569098-736557317= 11,781
Pippip_AES_TriXZi_Mikayla_ZMMmax,   2048,    2398521345,       1673890300,               724631045,        323.753, 14.131,  61.253 !
724631045-724638132= -7,087 i.e. better than Pippip_AES_TriXZi_Mikayla
Pippip_AES_TriXZi_Mikayla_ZMMmax,   8192,    2398515201,       1694717306,               703797895,        745.589, 24.543,  64.253 !
703797895-703775221= 22,674

XXH3_64bits 0.8.3,          128,     2398523265,       1648387301,               750135964,        210.107, 1.361,   8.906 ! 750135964-
750116295= 19,669 i.e. worse than Pippip_AES_TriXZi_Mikayla_ZMMmax
XXH3_64bits 0.8.3,          256,     2398523137,       1650961642,               747561495,        176.009, 3.249,   12.057 ! 747561495-
747539778= 21,717 i.e. worse than Pippip_AES_TriXZi_Mikayla_ZMMmax
XXH3_64bits 0.8.3,          512,     2398522881,       1655001212,               743521669,        248.454, 4.603,   18.357 ! 743521669-
743488026= 33,643 i.e. worse than Pippip_AES_TriXZi_Mikayla_ZMMmax
XXH3_64bits 0.8.3,          1024,    2398522369,       1661947538,               736574831,        287.056, 7.969,   25.772 ! 736574831-
736569098= 5,733 i.e. worse than Pippip_AES_TriXZi_Mikayla_ZMMmax
XXH3_64bits 0.8.3,          2048,    2398521345,       1673886241,               724635104,        642.147, 7.124,   33.020 ! 724635104-
724631045= 4,059 i.e. worse than Pippip_AES_TriXZi_Mikayla_ZMMmax
XXH3_64bits 0.8.3,          8192,    2398515201,       1694695692,               703819509,        969.465, 18.876,  40.195 ! 703819509-
703797895= 21,614 i.e. worse than Pippip_AES_TriXZi_Mikayla_ZMMmax
*/
```

Links:
https://github.com/Sanmayce/Pippip
https://github.com/rurban/smhasher/issues/322#issuecomment-3149148800
https://github.com/rrrlasse/eXdupe/issues/11#issuecomment-3149142008
https://forums.fedoraforum.org/showthread.php?334463-In-search-for-the-FASTEST-hash-(for-lookup-tables)&p=1894564#post1894564

The author: Georgi 'Sanmayce' Marinov, rendered by PixAI.art from an original photo



Testmachine: Latitude 7420 8x4 GB (Rank: 2) LPDDR4 4267 MT/s, Intel i7-1185G7 "Tigerlake", TDP: (MAX 25 W), 10 nm, Max Turbo: 4,800 MHz
Compiler used: GCC 15.1.1 (-static -O3 -march=tigerlake)
hashing 'Fedora-Workstation-Live-42-1.1.x86_64.iso' (2,398,523,392 bytes) at each position for orders 2..8192,
Note: with -mavx2 results are different, wanted the benchmark to run on Ivy Bridge as well