



Hasher	[Raw Hashing Speed]	[Sorted]
	for 32 bytes	for 8 KB Collisions for 8 KB
Pippip AES TriXZi Mikayla	15.328 GiB/s	38.751 GiB/s 783,775,221
Pippip AES TriXZi Mikayla_ZMMmax	15.747 GiB/s	50.999 GiB/s 783,797,895
xxhash 14	14.119 GiB/s	29.795 GiB/s 783,884,152
MeowHash	4.302 GiB/s	53.853 GiB/s 783,889,114
Komihash 5.27	9.523 GiB/s	22.725 GiB/s 783,811,325
XXH3 64bits 0.8.3	10.921 GiB/s	67.232 GiB/s 783,819,589



Regarding Collisions, a quick test for 10 billion 128 bytes long keys (2 collisions for Pippip, 4 collisions for xxh3):

```
$ ./knight_tour_omp_progress_64x64variants_Pippip3072 2621440
$ sort --temporary-directory=./ -u --buffer-size=2G --parallel=16 dump_10B_Pippip3072.txt -o dump_10B_Pippip3072.txt.Linuxsort
10737418240 dump_10B_Pippip3072.txt
10737418238 dump_10B_Pippip3072.txt.Linuxsort
```

```
$ ./knight_tour_omp_progress_64x64variants_Pippip 2621440
$ sort --temporary-directory=./ -u --buffer-size=2G --parallel=16 dump_10B_Pippip.txt -o dump_10B_Pippip.txt.Linuxsort
10737418240 dump_10B_Pippip.txt
10737418238 dump_10B_Pippip.txt.Linuxsort
```

```
$ ./knight_tour_omp_progress_64x64variants_FNV1A 2621440
$ sort --temporary-directory=./ -u --buffer-size=2G --parallel=16 dump_10B_FNV1A.txt -o dump_10B_FNV1A.txt.Linuxsort
10737418240 dump_10B_FNV1A.txt
10737418237 dump_10B_FNV1A.txt.Linuxsort
```

```
$ ./knight_tour_omp_progress_64x64variants_xxh3 2621440
$ sort --temporary-directory=./ -u --buffer-size=2G --parallel=16 dump_10B_xxh3.txt -o dump_10B_xxh3.txt.Linuxsort
10737418240 dump_10B_xxh3.txt
10737418236 dump_10B_xxh3.txt.Linuxsort
```

<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.fedoraproject.org/showthread.php?334463-In-search-for-the-FAASTEST-hash-\(for-lookup-tables\)&p=1894564#post1894564](https://forums.fedoraproject.org/showthread.php?334463-In-search-for-the-FAASTEST-hash-(for-lookup-tables)&p=1894564#post1894564)

```
// Below 3 rosters show:  

// 'Pippip_AES_TriXZi_Mikayla_ZMMmax' versus 'XXH3_64bits 0.8.3' is:  

// 6.777/5.087= 1.33x faster for 32 B keys, on i7-1185G7-UP3  

// 37.546/20.178= 1.86x faster for 512 B keys, on i7-1185G7-UP3  

// 70.147/43.571= 1.60x faster for 8 KB keys, on i7-1185G7-UP3  

// 15.747/10.921= 1.44x faster for 32 B keys, on Ryzen 7 PRO 7840U  

// 63.997/49.417= 1.29x faster for 512 B keys, on Ryzen 7 PRO 7840U  

// 90.999/67.232= 1.35x faster for 8 KB keys, on Ryzen 7 PRO 7840U  

// 11.067/8.652= 1.27x faster for 32 B keys, on i7-11850H  

// 59.841/35.368= 1.69x faster for 512 B keys, on i7-11850H  

// 104.710/72.097= 1.45x faster for 8 KB keys, on i7-11850H  

// Or, the bottomline: 'Pippip_AES_TriXZi_Mikayla_ZMMmax' versus 'XXH3_64bits 0.8.3' is from 27% to 86% faster.
```

```
/*  

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, Max TDP: 28W, 10nm  

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  

+-----[ Raw Hashing Speed, in GiB/s ]-----+-----[ Sorted ]+-  

| 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 | ! ri-donkey-lous !  

| Pippip_AES_TriXZi_Mikayla_ZMMmax | 6.777 | 37.546 | 70.147 | 703,797,895 |  

| 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 |  

+-----+-----+-----+-----+-----+
```

Note1: Compiler used: GCC 15.1.1 (-DZMM -static -O3 -march=tigerlake)
Note2: The user was superuser, CPU was set to performance mode, niceness was maximum: "nice -n -20"
*/

```
/*  

Testfile: Fedora-Workstation-Live-42-1.1.x86_64.iso, 2,398,523,392 bytes  

CPU: AMD Ryzen 7 PRO 7840U @max 5134MHz, Max TDP: 30W, 4nm  

Laptop: ThinkPad P14s Gen 4  

RAM: 4x16GB, Form Factor: 7, Type: LPDDR5, Rank: 2, Configured Memory Speed: 6400 MT/s  

OS: Linux Fedora, as superuser in performance mode with maximum niceness  

ThrottlingWISE: It took 558 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 18299/558 ≈ 32 GiB/s hash-n-lookup  

+-----[ Raw Hashing Speed, in GiB/s ]-----+-----[ Sorted ]+-  

| Hasher | for 32 bytes | for 512 bytes | for 8 KB | Collisions for 8 KB |  

+-----+-----+-----+-----+-----+  

| Pippip_AES_TriXZi_Mikayla | 15.328 | 50.195 | 38.751 | 703,775,221 | ! ri-donkey-lous !  

| Pippip_AES_TriXZi_Mikayla_ZMMmax | 15.747 | 63.997 | 90.999 | 703,797,895 |  

| wyhash f4 | 14.119 | 30.206 | 29.705 | 703,804,152 |  

| MeowHash | 4.302 | 34.045 | 53.853 | 703,809,114 |  

| komihash 5.27 | 9.523 | 20.983 | 22.725 | 703,811,325 |  

| XXH3_64bits 0.8.3 | 10.921 | 49.417 | 67.232 | 703,819,509 |  

+-----+-----+-----+-----+-----+
```

Note1: Compiler used: GCC 15.1.1 (-DZMM -static -O3 -march=znver4)
Note2: The user was superuser, CPU was set to performance mode, niceness was maximum: "nice -n -20"
*/

```
/*  

Testfile: Fedora-Workstation-Live-42-1.1.x86_64.iso, 2,398,523,392 bytes  

CPU: 11th Gen Intel Core i7-11850H @max 4.80GHz, Max TDP: 45W, 10nm  

Laptop: Dell Precision 7560  

RAM: 2x16GB (Rank: 1) + 2x32GB (Rank: 2), Form Factor: SODIMM, Type: DDR4, Configured Memory Speed: 2933 MT/s  

OS: Linux Fedora, as superuser in performance mode with maximum niceness  

ThrottlingWISE: It took 467 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 18299/467 ≈ 39 GiB/s hash-n-lookup  

+-----[ Raw Hashing Speed, in GiB/s ]-----+-----[ Sorted ]+-  

| Hasher | for 32 bytes | for 512 bytes | for 8 KB | Collisions for 8 KB |  

+-----+-----+-----+-----+-----+  

| Pippip_AES_TriXZi_Mikayla | 11.729 | 49.496 | 47.438 | 703,775,221 | ! ri-donkey-lous !  

| Pippip_AES_TriXZi_Mikayla_ZMMmax | 11.067 | 59.841 | 104.710 | 703,797,895 |  

| wyhash f4 | 10.975 | 26.433 | 29.851 | 703,804,152 |  

| MeowHash | 3.066 | 26.823 | 48.520 | 703,809,114 |  

| komihash 5.27 | 7.363 | 19.846 | 26.002 | 703,811,325 |  

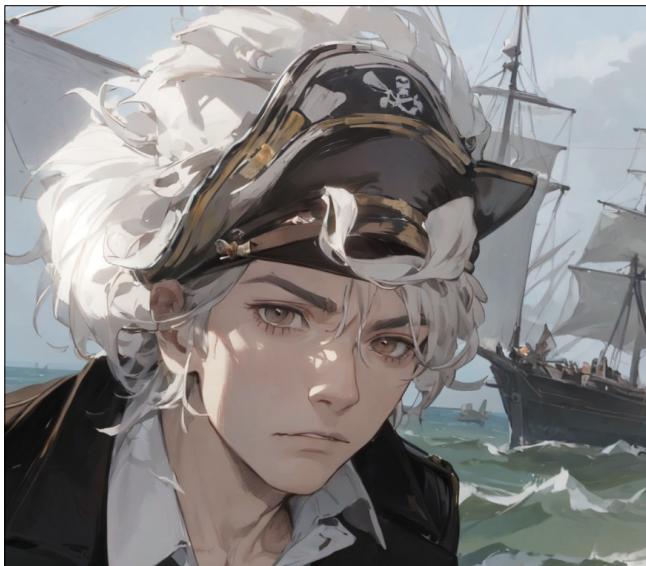
| XXH3_64bits 0.8.3 | 8.652 | 35.368 | 72.097 | 703,819,509 |  

+-----+-----+-----+-----+-----+
```

Note1: Compiler used: GCC 15.1.1 (-DZMM -static -O3 -march=tigerlake)
Note2: The user was superuser, CPU was set to performance mode, niceness was maximum: "nice -n -20"
*/

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.fedoraproject.org/showthread.php?7334463-In-search-for-the-FAATEST-hash-\(for-lookup-tables\)&p=1894564#post1894564](https://forums.fedoraproject.org/showthread.php?7334463-In-search-for-the-FAATEST-hash-(for-lookup-tables)&p=1894564#post1894564)



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



Pippip.h:

```
// FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla_ZMMmax[3072]: the 100% FREE lookuper, last update: 2025-Aug-17, Kaze
(sanmayce@sanmayce.com). [
// Note3: 'ZMMmax3072' just doubles the state of 'ZMMmax', being 1536=4x128bit+4x128bit+4x128bit, by being bidirectional.
// Note2: 'ZMMmax' variant outperforms all hashers known to me, see below.
// Note1: This latest revision was written when Mikayla "saveafox" left this world.

// "There it now stands for ever. Black on white.
// I can't get away from it. Ahoy, Yorikke, ahoy, hoy, ho!
// Go to hell now if you wish. What do I care? It's all the same now to me.
// I am part of you now. Where you go I go, where you leave I leave, when you go to the devil I go. Married.
// Vanished from the living. Damned and doomed. Of me there is not left a breath in all the vast world.
// Ahoy, Yorikke! Ahoy, hoy, ho!
// I am not buried in the sea,
// The death ship is now part of me
// So far from sunny New Orleans
// So far from lovely Louisiana."
// /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B.TRAVEN/
//
// "Walking home to our good old Yorikke, I could not help thinking of this beautiful ship, with a crew on board that had faces as if they
// were seeing ghosts by day and by night.
// Compared to that gilded Empress, the Yorikke was an honorable old lady with lavender sachets in her drawers.
// Yorikke did not pretend to anything she was not. She lived up to her looks. Honest to her lowest ribs and to the leaks in her bilge.
// Now, what is this? I find myself falling in love with that old jane.
// All right, I cannot pass by you, Yorikke; I have to tell you I love you. Honest, baby, I love you.
// I have six black finger-nails, and four black and green-blue nails on my toes, which you, honey, gave me when necking you.
// Grate-bars have crushed some of my toes. And each finger-nail has its own painful story to tell.
// My chest, my back, my arms, my legs are covered with scars of burns and scorchings.
// Each scar, when it was being created, caused me pains which I shall surely never forget.
// But every outcry of pain was a love-cry for you, honey.
// You are no hypocrite. Your heart does not bleed tears when you do not feel heart-aches deeply and truly.
// You do not dance on the water if you do not feel like being jolly and kicking chasers in the pants.
// Your heart never lies. It is fine and clean like polished gold. Never mind the rags, honey dear.
// When you laugh, your whole soul and all your body is laughing.
// And when you weep, sweetly, then you weep so that even the reefs you pass feel like weeping with you.
// I never want to leave you again, honey. I mean it. Not for all the rich and elegant buckets in the world.
// I love you, my gypsy of the sea!"
// /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B.TRAVEN/
//
// Dedicated to Pippip, the main character in the 'Das Totenschiff' roman, actually the B.Trauen himself, his real name was Hermann Albert
Otto Maksymilian Feige.
// CAUTION: Add 8 more bytes to the buffer being hashed, usually malloc(...+8) - to prevent out of boundary reads!
// Many thanks go to Yurii 'Hordi' Hordiienko, he lessened with 3 instructions the original 'Pippip', thus:
//

#include <stdlib.h>
#include <stdint.h>
#include <immintrin.h>
#if defined(ZMM)

#endif
#define HashInBITS 32 // default 26 i.e. 2^26 i.e. 64MS(Mega Slots); slots contain 8bytes pointers or 512MB, because many netbooks have 512MB
free (1GB in total)!
#define MASK ((1ULL << HashInBITS) - 1)
#define _PADr_KAZE(x, n) ( ((x) << (n))>>(n) )
#define _PAD_KAZE(x, n) ( ((x) << (n)) )

typedef struct {
    uint64_t low;
    uint64_t high;
}

FNV1A_Pippip_128bit_AES_TriXZi_Mikayla (FNV1A-Deathship-Dualheaded a.k.a. FNV1A-Totenköpfe a.k.a. FNV1A-Totenschiff), Update: 2025-Aug-19, Kaze(sanmayce@sanmayce.com), 3/46
```

```

} hashPippip128_t;

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
}

static inline uint32_t fold32(uint32_t A, uint32_t B) {
    #if defined(_GNUC_) || defined(_clang_)
        uint64_t r = (uint64_t)A * (uint64_t)B;
        return (uint32_t)r ^ (uint32_t)(r >> 32);
    #else
        uint32_t hash32 = A ^ B;
        hash32 *= 591798841;
        return hash32;
    #endif
}

// The ZMMmax dominates! In contrast to above variant it ourperforms xxh3 by 64/41=1.56x
// Let us compare the collisions for Fedora testdataset:
/*
Setting CPU to performance mode...
performance
Size: 2398523392
Read: 2398523392
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, 16509115581, 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
*/
#endif

void FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla_ZMMmax (const char *str, size_t wrdlen, uint32_t seed, void *output) {
    __m128i chunkA;
    __m128i chunkB;
    __m128i stateMIX;
    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, 0x9e3779b97fda7c15);
        __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 );
__m512i StateB_zmm = __mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
__m512i StateC_zmm = __mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
__m512i StateA_zmm = __mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
__m512i StateB_zmm = __mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
__m512i StateC_zmm = __mm512_set_epi64( 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 -maxv512f -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, chunkB);
        stateB = __mm_aesenc_si128(stateB, chunkB2);
        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);
}
#ifndef 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 ^ wrflen;
#endif
}

// The above monster GCC: (GNU) 15.1.1 assembler:
// 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
/*
FNV1A_Pippip_Yurii_000_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
vpxtrq $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, %xmm5, %xmm6
vextracti64x2 $1, %xmm4, %xmm3
vaesenc %xmm4, %xmm0, %xmm0
vextracti64x2 $1, %xmm1, %xmm2
vaesenc %xmm1, %xmm0, %xmm0
vaesenc %xmm6, %xmm0, %xmm0
vextracti64x2 $2, %xmm5, %xmm6
vextracti64x2 $3, %xmm5, %xmm5
vaesenc %xmm3, %xmm0, %xmm0
vextracti64x2 $2, %xmm4, %xmm3
vextracti64x2 $3, %xmm4, %xmm4
vaesenc %xmm2, %xmm0, %xmm0
vextracti64x2 $2, %xmm1, %xmm2
vextracti64x2 $3, %xmm1, %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
.ident "GCC: (GNU) 15.1.1 20250521 (Red Hat 15.1.1-2)"
*/
#endif defined(ZMM)
void FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla_ZMMmax3072 (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;
                __m128i stateB2;
                __m128i stateC2;
                __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 );
                __m512i StateAR_zmm = __mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
                __m512i StateBR_zmm = __mm512_set_epi64( 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL, 0x6c62272e07bb0142ULL, 0x9e3779b97f4a7c15ULL );
                __m512i StateCR_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+256, _MM_HINT_T0);
                    __mm_prefetch(str+NDhead+256, _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);
                    __m512i chunk_zmmR = __mm512_loadu_si512((__m512i *)str+Cycles*64);
                    __m512i chunk_zmm_offsetR = __mm512_loadu_si512((__m512i *)str+NDhead+Cycles*64);
                    StateAR_zmm = __mm512_aesenc_epi128(StateAR_zmm, chunk_zmmR);
                    StateBR_zmm = __mm512_aesenc_epi128(StateBR_zmm, chunk_zmm_offsetR);
                    StateCR_zmm = __mm512_aesenc_epi128(StateCR_zmm, chunk_zmmR);
                    StateCR_zmm = __mm512_aesenc_epi128(StateCR_zmm, chunk_zmm_offsetR);
                }
            }
        }
    }
}

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);

stateA2 = __mm512_extracti64x2_epi64(StateAR_zmm, 0);
stateB2 = __mm512_extracti64x2_epi64(StateBR_zmm, 0);
stateC2 = __mm512_extracti64x2_epi64(StateCR_zmm, 0);
stateMIX = __mm_aesenc_si128(stateMIX, stateA2);
stateMIX = __mm_aesenc_si128(stateMIX, stateB2);
stateMIX = __mm_aesenc_si128(stateMIX, stateC2);

stateA2 = __mm512_extracti64x2_epi64(StateAR_zmm, 1);
stateB2 = __mm512_extracti64x2_epi64(StateBR_zmm, 1);
stateC2 = __mm512_extracti64x2_epi64(StateCR_zmm, 1);
stateMIX = __mm_aesenc_si128(stateMIX, stateA2);

```

```

stateMIX = _mm_aesenc_si128(stateMIX, stateB2);
stateMIX = _mm_aesenc_si128(stateMIX, stateC2);

stateA2 = _mm512_extracti64x2_epi64(StateAR_zmm, 2);
stateB2 = _mm512_extracti64x2_epi64(StateBR_zmm, 2);
stateC2 = _mm512_extracti64x2_epi64(StateCR_zmm, 2);
stateMIX = _mm_aesenc_si128(stateMIX, stateA2);
stateMIX = _mm_aesenc_si128(stateMIX, stateB2);
stateMIX = _mm_aesenc_si128(stateMIX, stateC2);

stateA2 = _mm512_extracti64x2_epi64(StateAR_zmm, 3);
stateB2 = _mm512_extracti64x2_epi64(StateBR_zmm, 3);
stateC2 = _mm512_extracti64x2_epi64(StateCR_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);
}
#endif
#endif

// Cycles = ((wrdlen - 1)>>5) + 1;
// NDhead = wrdlen - (Cycles<<4);
// And some visualization for XMM-WORD:
/*
k1= 33..64 Cycles= (k1-1)/32+1=2; MARGINAL CASES:
2nd head starts at 33-2*16=1 or:
0123456789012345 0123456789012345 0
Head1: [XMM-WORD ] [XMM-WORD ]
Head2: [XMM-WORD ] [XMM-WORD ]

2nd head starts at 64-2*16=32 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345
Head1: [XMM-WORD ] [XMM-WORD ] [XMM-WORD ]
Head2: [XMM-WORD ] [XMM-WORD ] [XMM-WORD ]

k1=65..96 Cycles= (k1-1)/32+1=3; MARGINAL CASES:
2nd head starts at 65-3*16=17 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345 0
Head1: [XMM-WORD ] [XMM-WORD ] [XMM-WORD ]

```

```

        Head2:           [XMM-WORD]   ] [XMM-WORD]   ] [XMM-WORD]   ]
2nd head starts at 96-3*16=48 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345 0123456789012345 0123456789012345
Head1: [XMM-WORD]   ] [XMM-WORD]   ] [XMM-WORD]   ] [XMM-WORD]   ] [XMM-WORD]   ] [XMM-WORD]   ]
Head2:                                     [XMM-WORD]   ] [XMM-WORD]   ] [XMM-WORD]   ]
*/
// And some visualization for Q-WORD:
/*
k1= 9..16 Cycles= (k1-1)/16+1=1; MARGINAL CASES:
2nd head starts at 9-1*8=1 or:
012345678
Head1: [Q-WORD]
Head2: [Q-WORD]

2nd head starts at 16-1*8=8 or:
0123456789012345
Head1: [Q-WORD]
Head2: [Q-WORD]

k1=17..24 Cycles= (k1-1)/16+1=2; MARGINAL CASES:
2nd head starts at 17-2*8=1 or:
01234567890123456
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

2nd head starts at 24-2*8=8 or:
012345678901234567890123
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

k1=25..32 Cycles= (k1-1)/16+1=2; MARGINAL CASES:
2nd head starts at 25-2*8=9 or:
0123456789012345678901234
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

2nd head starts at 32-2*8=16 or:
01234567890123456789012345678901
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

k1=33..40 Cycles= (k1-1)/16+1=3; MARGINAL CASES:
2nd head starts at 33-3*8=9 or:
012345678901234567890123456789012
Head1: [Q-WORD][Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD][Q-WORD]

2nd head starts at 40-3*8=16 or:
0123456789012345678901234567890123456789
Head1: [Q-WORD][Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD][Q-WORD]

k1=41..48 Cycles= (k1-1)/16+1=3; MARGINAL CASES:
2nd head starts at 41-3*8=17 or:
01234567890123456789012345678901234567890
Head1: [Q-WORD][Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD][Q-WORD]

2nd head starts at 48-3*8=24 or:
012345678901234567890123456789012345678901234567
Head1: [Q-WORD][Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD][Q-WORD]
*/
// The more the merrier, therefore I added the 10,000 GitHub stars performer xxhash also:
// https://github.com/Cyan4973/xxHash/issues/1029
//
// Pippip is not an extremely fast hash, it is the spirit of the author materialized disregarding anything outside the "staying true to oneself", or as one bona fide man Otto/Pippip once said:
//
// Translate as verbatim as possible:
// In 1926, Traven wrote that the only biography of a writer should be his works: «Die Biographie eines schöpferischen Menschen ist ganz und gar unwichtig. Wenn der Mensch in seinen Werken nicht zu erkennen ist, dann ist entweder der Mensch nichts wert oder seine Werke sind nichts wert. Darum sollte der schöpferische Mensch keine andere Biographie haben als seine Werke» (Hauschild, B. Traven: Die unbekannten Jahre, op. cit., p. 31.)
//
// In 1926, Traven wrote that the only biography of a writer should be his works:
// "The biography of a creative person is completely and utterly unimportant.
// If the person is not recognizable in his works, then either the person is worthless or his works are worthless.
// Therefore, the creative person should have no other biography than his works" (Hauschild, B. Traven: Die unbekannten Jahre, op. cit., p. 31.)
//
// https://godbolt.org/
// x86-64 clang 20.1.0
// -O3 -maes -mbmi2
/*
...
*/
// https://godbolt.org/
// x86-64 gcc 14.2
// -O3 -maes -mbmi2
/*
...
*/

```

*/

Regarding Collisions, a quick test for 10 billion 128 bytes long keys:

```
$ ./knight_tour_omp_progress_64x64variants_Pippip3072 2621440
$ sort --temporary-directory=./ -u --buffer-size=2G --parallel=16 dump_10B_Pippip3072.txt -o dump_10B_Pippip3072.txt.Linuxsort
10737418240 dump_10B_Pippip3072.txt
10737418238 dump_10B_Pippip3072.txt.Linuxsort

$ ./knight_tour_omp_progress_64x64variants_Pippip 2621440
$ sort --temporary-directory=./ -u --buffer-size=2G --parallel=16 dump_10B_Pippip.txt -o dump_10B_Pippip.txt.Linuxsort
10737418240 dump_10B_Pippip.txt
10737418238 dump_10B_Pippip.txt.Linuxsort

$ ./knight_tour_omp_progress_64x64variants_FNV1A 2621440
$ sort --temporary-directory=./ -u --buffer-size=2G --parallel=16 dump_10B_FNV1A.txt -o dump_10B_FNV1A.txt.Linuxsort
10737418240 dump_10B_FNV1A.txt
10737418237 dump_10B_FNV1A.txt.Linuxsort

$ ./knight_tour_omp_progress_64x64variants_xxh3 2621440
$ sort --temporary-directory=./ -u --buffer-size=2G --parallel=16 dump_10B_xxh3.txt -o dump_10B_xxh3.txt.Linuxsort
10737418240 dump_10B_xxh3.txt
10737418236 dump_10B_xxh3.txt.Linuxsort

knight_tour_omp_progress_64x64variants_Pippip3072.c:

// gcc -O3 -fopenmp -DExdupe -DZMM -march=tigerlake knight_tour_omp_progress_64x64variants_Pippip3072.c xxhash.c -o
knight_tour_omp_progress_64x64variants_Pippip3072
// ./knight_tour_omp_progress_64variants 1000 > tours.txt
// ./knight_tour_omp_progress_64variants 1000 > dump.txt 2> progress.txt

#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>

#define SIZE 8
#define MAX_MOVES 64

#include "xxhash.h"
#include "Pippip.h"

int moves_x[8] = {2, 1, -1, -2, -2, -1, 1, 2};
int moves_y[8] = {1, 2, 2, 1, -1, -2, -2, -1};
long long global_tour_count = 0;
long long max_tours = 0;

int is_valid(int x, int y, int board[SIZE][SIZE]) {
    return x >= 0 && x < SIZE && y >= 0 && y < SIZE && board[x][y] == 0;
}

void pos_to_str(int x, int y, char *str) {
    str[0] = 'A' + y;
    str[1] = '1' + x;
}

// 64 Expression 240 + 28 + 0xb3
// Decimal 1099511628211 14695981039346656037
// Hexadecimal 0x00000100000001b3 0xcbf29ce484222325

// 128 Representation 288 + 28 + 0x3b
// Decimal 309485009821345068724781371 144066263297769815596495629667062367629
// Hexadecimal 0x00000000010000000000000000000013b 0x6c62272e07bb014262b821756295c58d

uint64_t FNV1a_hash64(const char *data, size_t len) {
    uint64_t h = 14695981039346656037ULL;
    for (size_t i = 0; i < len; i++) {
        h ^= (uint64_t)(unsigned char) data[i];
        h *= 1099511628211ULL;
    }
    return h;
}

void generate_and_hash_variations(char *tour) {
    char variant[MAX_MOVES * 2];

    // For each lowercase position (0 to 63)
    for (int lowercase_pos = 0; lowercase_pos < MAX_MOVES; lowercase_pos++) {
        // Copy base tour and lowercase one letter
        strcpy(variant, tour);
        int letter_index = lowercase_pos * 2;
        if (variant[letter_index] >= 'A' && variant[letter_index] <= 'H') {
            variant[letter_index] += 32; // Lowercase the letter
        }

        // For each swap position (0 to 63)
        for (int swap_pos = 0; swap_pos < MAX_MOVES; swap_pos++) {
            // Copy the lowercased variant
            char swap_variant[MAX_MOVES * 2];
            strcpy(swap_variant, variant);

            // Swap letter and digit in the selected pair
            int pair_start = swap_pos * 2;
            char temp = swap_variant[pair_start];
            swap_variant[pair_start] = swap_variant[pair_start + 1];
            swap_variant[pair_start + 1] = temp;
        }
    }
}
```

```

// Compute hash and print as 16 HEX + LF
//uint64_t hash = FNV1a_hash64(swap_variant, MAX_MOVES * 2);
//printf("%016llx\n", hash);
    uint64_t hashPippip[2];
    FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla_ZMMmax3072 (swap_variant, MAX_MOVES * 2, 0, hashPippip);
    printf("%016llx\n", hashPippip[0]);
}

void knight_tour(int x, int y, int move_count, int board[SIZE][SIZE], char *tour) {
    if (global_tour_count >= max_tours) {
        return;
    }

    if (move_count == MAX_MOVES) {
        tour[MAX_MOVES * 2 - 1] = '\0';
        #pragma omp critical
        {
            if (global_tour_count < max_tours) {
                generate_and_hash_variations(tour);
                global_tour_count++;
                if (max_tours >= 100 && global_tour_count % (max_tours / 100) == 0) {
                    fprintf(stderr, "Progress: %3lld%%\n", (global_tour_count * 100) / max_tours);
                }
            }
        }
        return;
    }

    for (int i = 0; i < 8 && global_tour_count < max_tours; i++) {
        int next_x = x + moves_x[i];
        int next_y = y + moves_y[i];

        if (is_valid(next_x, next_y, board)) {
            board[next_x][next_y] = move_count + 1;
            pos_to_str(next_x, next_y, tour + move_count * 2);
            knight_tour(next_x, next_y, move_count + 1, board, tour);
            board[next_x][next_y] = 0;
        }
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <N>\n", argv[0]);
        return 1;
    }

    max_tours = atol(argv[1]);
    if (max_tours <= 0) {
        fprintf(stderr, "N must be a positive integer\n");
        return 1;
    }

    //#pragma omp parallel num_threads(64)
    #pragma omp parallel num_threads(1)
    {
        int board[SIZE][SIZE];
        char tour[MAX_MOVES * 2];
        memset(board, 0, sizeof(board));

        int tid = omp_get_thread_num();
        int start_x = tid / SIZE;
        int start_y = tid % SIZE;

        board[start_x][start_y] = 1;
        pos_to_str(start_x, start_y, tour);
        knight_tour(start_x, start_y, 1, board, tour);
    }

    return 0;
}
*/

```

Grok 3:
The total number of possible open Knight's Tours on an 8x8 chessboard-meaning all directed sequences where the knight visits every square exactly once, starting from any square-is enormous. The 64-threaded program would attempt to enumerate all such tours across all 64 starting positions (one thread per square), resulting in a grand total that sums to the overall count of directed open tours.

Based on computational enumerations, the exact number of directed open Knight's Tours is 19,591,828,170,979,904 (approximately 1.96×10^{16}). This is on the order of quadrillions-specifically, about 19.6 quadrillion.

The 64-threaded program, which generates all possible open Knight's Tours for each of the 64 starting squares (A1 to H8) on an 8x8 chessboard, would theoretically print approximately 19.6 quadrillion lines if it were allowed to run to completion. This is because the total number of directed open Knight's Tours (where each tour is a unique sequence of 64 moves visiting every square exactly once) is estimated to be 19,591,828,170,979,904 ($\approx 1.96 \times 10^{16}$), as noted earlier. Each tour is output as a single 128-byte line (e.g., "A1H3..."). Breakdown:

Total Tours: ~19.6 quadrillion (1.96×10^{16}) directed open tours across all 64 starting positions.

Output: Each tour is a 128-byte line, so the program would produce 19.6 quadrillion lines, one for each tour.

Per Thread: The number of tours varies by starting position (e.g., corner squares like A1 produce fewer tours than central squares like D4). On average, each thread (handling one starting square) would generate $\sim 1.96 \times 10^{16} / 64 \approx 3.06 \times 10^{14}$ tours (306 trillion lines per thread).

Data Volume: Each line is 128 bytes, so the total output would be $\sim 19.6 \times 10^{15} \times 128 \text{ bytes} \approx 2.5 \times 10^{18} \text{ bytes} \approx 2.5 \text{ exabytes}$ (2.5 million terabytes). This is an infeasible amount of data to store or process in practice.

Practical Considerations:

Runtime: Generating 19.6 quadrillion tours is computationally infeasible. Even if each tour took 1 microsecond to compute and print (optimistic), it would take $\sim 19.6 \times 10^{15} \text{ } \mu\text{s} \approx 620,000$ years to complete on a single thread. With 64 threads, assuming perfect parallelization and no overhead, it would still take $\sim 9,700$ years.
Output Bottleneck: Writing 2.5 exabytes to disk or stdout would be a major bottleneck, limited by I/O speed (e.g., modern SSDs write at ~1-7 GB/s, so writing 2.5 EB would take years).
Memory: The memory footprint remains low (~192 KB actual usage, ~512 MB allocated), as the program only stores one tour at a time per thread. The output volume doesn't affect RAM usage, as tours are printed immediately.

Clarifications:

Directed Tours: The 19.6 quadrillion figure is for directed open tours (each path counted separately from its reverse). If you meant undirected tours (path and reverse counted as one), the number is halved (~9.8 quadrillion lines). If you meant closed tours (where the knight returns to the start), the number is much smaller (~2.65 $\times 10^{13}$ lines), but the program generates open tours.
Program Behavior: Each of the 64 threads independently enumerates tours for its starting square, and the combined output across all threads sums to the total ~19.6 quadrillion lines.

How Backtracking Works in the Program
The program uses a recursive backtracking algorithm in the `knight_tour` function to find all possible Knight's Tours for a given starting position. Backtracking involves:

Exploring a Path: Trying a valid knight move, marking it as visited, and recursively exploring further moves.
Backtracking: If a path doesn't lead to a solution (a complete tour of 64 moves), the algorithm "undoes" the last move, reverts the state, and tries another move.

The key to understanding the low memory usage lies in how the program stores and manages the state of each path (the "tracks") during this process.

Where Are the Tracks Written?
The "tracks" refer to the state of the knight's path as it explores the board. This state is stored in two main data structures, both of which are highly compact:

The board Array:

Definition: `int board[SIZE][SIZE]` ($8 \times 8 = 64$ integers, 256 bytes at 4 bytes per integer).
Purpose: Tracks which squares have been visited and in what order.

`board[x][y] = 0` means the square (x,y) is unvisited.
`board[x][y] = move_count` indicates the square was visited at step `move_count` (1 to 64).

Backtracking Mechanism:

When a move is made to $(\text{next_x}, \text{next_y})$, the program sets `board[next_x][next_y] = move_count + 1` to mark it as visited.
When backtracking, it resets `board[next_x][next_y] = 0` to mark the square as unvisited, allowing it to try a different move.

Memory Usage: The board array is fixed at 256 bytes per thread, regardless of recursion depth. It's reused for all paths explored by a thread, as moves are marked and unmarked during backtracking.

The tour Array:

Definition: `char tour[MAX_MOVES * 2]` (64 moves \times 2 chars = 128 bytes).
Purpose: Stores the current tour as a string (e.g., "A1H3...") for output when a valid tour is found.
Backtracking Mechanism:

For each move, the program writes the position (e.g., "C2") to `tour[move_count * 2]` using `pos_to_str`.
When backtracking, the program doesn't explicitly erase tour entries because the next move overwrites them at the same `move_count * 2` index.
This ensures tour always reflects the current path.
When a complete tour is found (`move_count == 64`), tour is printed, and backtracking continues to explore other paths.

Memory Usage: Fixed at 128 bytes per thread, reused for all paths.

Recursive Call Stack:

Definition: Each recursive call to `knight_tour` creates a stack frame with local variables (`x, y, move_count, i, next_x, next_y`) and overhead (e.g., return address).
Purpose: The call stack implicitly tracks the sequence of moves by maintaining the state of each recursive call (e.g., current position and move count).
Backtracking Mechanism:

When a recursive call returns (either after finding a tour or hitting a dead end), the stack frame is popped, automatically reverting to the previous state (prior position and move count).
This eliminates the need to explicitly store the entire path history, as the call stack naturally handles the "undo" process.

Memory Usage:

Each stack frame is small: ~24 bytes (3 ints for `x, y, move_count + 1` int for `i + 2` ints for `next_x, next_y`) + ~16 bytes overhead (return address, alignment) = ~40 bytes.

Maximum recursion depth is 64 (for a complete tour), so $64 \times 40 \text{ bytes} = \sim 2,560 \text{ bytes}$ (2.56 KB) per thread for the deepest path.

Why Is Memory Usage So Low?

The memory footprint per thread (~3 KB) is low because:

Compact State Representation:

The board (256 bytes) and tour (128 bytes) arrays are fixed-size and reused across all paths. They don't grow with the number of paths explored.

Only one path is explored at a time per thread, and backtracking reverts changes in-place (e.g., setting `board[x][y] = 0` or overwriting tour).

Implicit Path Tracking via Recursion:

The recursive call stack acts as the "history" of moves, eliminating the need for an explicit data structure to store the path.
Each recursive call adds only ~40 bytes, and the maximum depth (64) keeps the stack usage at ~2.56 KB.

No Additional Data Structures:

The program doesn't store all possible paths or maintain a history of explored moves. It only keeps the current path's state in board and tour.

When a tour is found, it's printed immediately, and no additional memory is needed to store completed tours.

Thread-Local Storage:

Each of the 64 threads has its own board and tour (396 bytes total), allocated on the stack in main. This is fixed and doesn't grow with recursion.

OpenMP ensures each thread's stack is isolated, avoiding shared memory overhead.

Total Memory Breakdown (Per Thread)

Stack Variables in main: 256 bytes (board) + 128 bytes (tour) + 12 bytes (tid, start_x, start_y) = 396 bytes.

Recursive Stack: ~2,560 bytes (64 calls \times 40 bytes).
Total per Thread: 396 + 2,560 = ~2,956 bytes (~3 KB).
Across 64 Threads: 64 \times 3 KB = ~192 KB (actual usage), though the system allocates 64 \times 8 MB = 512 MB for thread stacks.

Visualizing Backtracking

Consider a thread starting at A1 (0,0):

Initial State: board[0][0] = 1, tour = "A1", move_count = 1.

Try Move to C2 (2,1): Set board[2][1] = 2, append "C2" to tour, recurse with move_count = 2.

Dead End: If no valid moves exist, unset board[2][1] = 0, pop the stack frame (reverting x, y, move_count), and try another move (e.g., B3).

Complete Tour: If move_count == 64, print tour and backtrack by unsetting the last move and popping the stack.

The board tracks visited squares, tour builds the output string, and the call stack tracks the current position and depth, all in a compact manner.

Why Not More Memory?

You might expect backtracking to require more memory due to the vast number of possible tours (potentially millions for some starting positions). However:

Single Path at a Time: Only the current path is stored in board and tour. Backtracking reverts changes in-place, so no additional memory is needed for alternative paths.

No Path Storage: Completed tours are printed immediately, not stored, avoiding memory accumulation.

Fixed Recursion Depth: The recursion depth is capped at 64, limiting stack growth to ~2.56 KB per thread.

Efficient Data Structures: The 8x8 board (256 bytes) and 128-byte tour are minimal for representing the state of an 8x8 chessboard and a 64-move path.

The arrays moves_x and moves_y in the Knight's Tour program define the possible moves a knight can make on a chessboard. Let's break down their purpose and values:

```
int moves_x[8] = {2, 1, -1, -2, -2, -1, 1, 2};  
int moves_y[8] = {1, 2, 2, 1, -1, -2, -2, -1};
```

Explanation:

A knight in chess moves in an "L-shape": two squares in one direction (horizontally or vertically) and one square perpendicular, or one square in one direction and two squares perpendicular. This results in up to eight possible moves from any given position on the board. These arrays represent the changes in the x (row) and y (column) coordinates for each of those moves.

Chessboard Coordinates: The program uses an 8x8 grid where rows (0 to 7) correspond to 1 to 8, and columns (0 to 7) correspond to A to H. For example, position A1 is (0,0), H8 is (7,7).

Knight's Moves: The knight's possible moves relative to its current position (x, y) are:

Two squares up/down and one square left/right.
One square up/down and two squares left/right.

Array Structure:

moves_x[i] gives the change in the row (x-coordinate) for the i-th move.

moves_y[i] gives the change in the column (y-coordinate) for the i-th move.

Each pair (moves_x[i], moves_y[i]) corresponds to one of the eight possible knight moves.

Move Breakdown:

The arrays are paired, so let's list the moves they represent:

Index, moves_x, moves_y, Move Description (Relative to Current Position), "Example from A1 (0,0)"

0, 2, 1, "Down 2, Right 1", "C2 (2,1)"

1, 1, 2, "Down 1, Right 2", "B3 (1,2)"

2, -1, 2, "Up 1, Right 2", "(invalid, out of board)"

3, -2, 1, "Up 2, Right 1", "(invalid, out of board)"

4, -2, -1, "Up 2, Left 1", "(invalid, out of board)"

5, -1, -2, "Up 1, Left 2", "(invalid, out of board)"

6, 1, -2, "Down 1, Left 2", "(invalid, out of board)"

7, 2, -2, "Down 2, Left 2", "(invalid, out of board)"

Positive/Negative Values:

Positive moves_x (e.g., 2, 1) means moving down (increasing row index, toward 8).

Negative moves_x (e.g., -1, -2) means moving up (decreasing row index, toward 1).

Positive moves_y (e.g., 1, 2) means moving right (increasing column index, toward H).

Negative moves_y (e.g., -1, -2) means moving left (decreasing column index, toward A).

Order of Moves: The order (e.g., starting with {2,1}, then {1,2}, etc.) is arbitrary but covers all eight possible L-shape moves in a consistent sequence. This order is often chosen for convenience and doesn't affect the correctness of the algorithm.

Validation: The is_valid function checks if a move lands within the 8x8 board (x: 0-7, y: 0-7) and if the square is unvisited (board[x][y] == 0). This handles cases where moves from A1 (or other edge squares) would go off the board.

Why These Values?

The values in moves_x and moves_y correspond to the knight's L-shape moves:

Two squares in one direction, one in the other: ($\pm 2, \pm 1$) or ($\pm 1, \pm 2$).

The arrays list all combinations: (2,1), (1,2), (-1,2), (-2,1), (-2,-1), (-1,-2), (1,-2), (2,-2).

This ensures the program systematically tries every possible knight move from any position.

Usage in the Program:

In the knight_tour function, these arrays are used to compute the next position:

```
int next_x = x + moves_x[i];  
int next_y = y + moves_y[i];
```

For each index i (0 to 7), the program calculates the new position and checks if it's valid before proceeding with the recursive backtracking.

Visualizing the Moves:

From a position like D4 (x=3, y=3), applying the moves gives:

(3+2, 3+1) = (5,4) \rightarrow F5

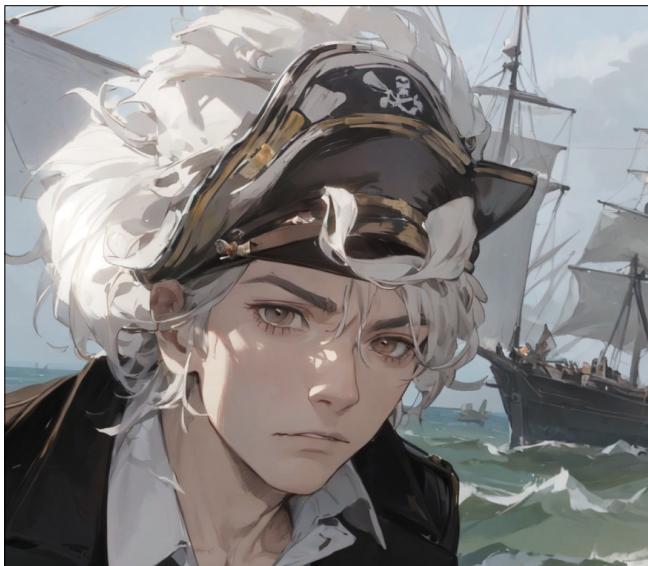
(3+1, 3+2) = (4,5) \rightarrow F6

(3-1, 3+2) = (2,5) \rightarrow F3

(3-2, 3+1) = (1,4) \rightarrow E2

And so on, covering all eight possibilities.

*/



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



```
// FNV1A_Pippip_Yurii_000_128bit_AES_TriXZ, last update: 2025-May-19, Kaze (sanmayce@sanmayce.com). [  

// "There it now stands for ever. Black on white.  

// I can't get away from it. Ahoy, Yorikke, ahoy, hoy, ho!  

// Go to hell now if you wish. What do I care? It's all the same now to me.  

// I am part of you now. Where you go I go, where you leave I leave, when you go to the devil I go. Married.  

// Vanished from the living. Damned and doomed. Of me there is not left a breath in all the vast world.  

// Ahoy, Yorikke! Ahoy, hoy, ho!  

// I am not buried in the sea,  

// The death ship is now part of me  

// So far from sunny New Orleans  

// So far from lovely Louisiana."  

// /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B.TRAVEN/  

//  

// "Walking home to our good old Yorikke, I could not help thinking of this beautiful ship, with a crew on board that had faces as if they were seeing ghosts by day and by night.  

// Compared to that gilded Empress, the Yorikke was an honorable old lady with lavender sachets in her drawers.  

// Yorikke did not pretend to anything she was not. She lived up to her looks. Honest to her lowest ribs and to the leaks in her bilge.  

// Now, what is this? I find myself falling in love with that old Jane.  

// All right, I cannot pass by you, Yorikke; I have to tell you I love you. Honest, baby, I love you.  

// I have six black finger-nails, and four black and green-blue nails on my toes, which you, honey, gave me when necking you.  

// Grate-bars have crushed some of my toes. And each finger-nail has its own painful story to tell.  

// My chest, my back, my arms, my legs are covered with scars of burns and scorchings.  

// Each scar, when it was being created, caused me pains which I shall surely never forget.  

// But every outcry of pain was a love-cry for you, honey.  

// You are no hypocrite. Your heart does not bleed tears when you do not feel heart-aches deeply and truly.  

// You do not dance on the water if you do not feel like being jolly and kicking chasers in the pants.  

// Your heart never lies. It is fine and clean like polished gold. Never mind the rags, honey dear.  

// When you laugh, your whole soul and all your body is laughing.  

// And when you weep, sweetly, then you weep so that even the reefs you pass feel like weeping with you.  

// I never want to leave you again, honey. I mean it. Not for all the rich and elegant buckets in the world.  

// I love you, my gypsy of the sea!"  

// /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B.TRAVEN/  

//  

// Dedicated to Pippip, the main character in the 'Das Totenschiff' roman, actually the B.Trauen himself, his real name was Hermann Albert Otto Maksymilian Feige.  

// CAUTION: Add 8 more bytes to the buffer being hashed, usually malloc(...+8) - to prevent out of boundary reads!  

// Many thanks go to Yurii 'Hordi' Hordienko, he lessened with 3 instructions the original 'Pippip', thus:
```

```
#include <stdlib.h>
#include <stdint.h>
#include <immintrin.h>
#define HashInBITS 32 // default 26 i.e. 2^26 i.e. 64MS(Mega Slots); slots contain 8bytes pointers or 512MB, because many netbooks have 512MB free (1GB in total) !
#define MASK ((1ULL << HashInBITS) - 1)
#define _PADr_KAZE(x, n) ( ((x) << (n))>>(n) )
#define _PAD_KAZE(x, n) ( ((x) << (n)) )  

typedef struct {
    uint64_t low;
    uint64_t high;
} hashPippip128_t;  

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
}
```

```

#endif
}

static inline uint32_t fold32(uint32_t A, uint32_t B) {
    #if defined(_GNUC_) || defined(_clang_)
        uint64_t r = (uint64_t)A * (uint64_t)B;
        return (uint32_t)r ^ (uint32_t)(r >> 32);
    #else
        uint32_t hash32 = A ^ B;
        hash32 *= 591798841;
        return hash32;
    #endif
}

uint32_t FNV1A_Pippip_Yurii_000_128bit_AES_TriXZ(const char *str, size_t wrdlen) {
    /*#define AESless
    #ifndef AESless
    uint64_t hash64A = 14695981039346656037ULL;
    uint64_t hash64B = 14695981039346656037ULL;
    //uint64_t hash64C = 14695981039346656037ULL;
    if (wrdlen > 8) {
        size_t Cycles, NDhead;
        Cycles = ((wrdlen - 1)>>4) + 1;
        NDhead = wrdlen - (Cycles<<3);
        /*#pragma nounroll
        for(; Cycles--; str += 8) {
            hash64A = fold64(hash64A, *((uint64_t *)str)); //(( hash64A ^ *((uint64_t *)str)) ) * PRIME;
            hash64B = fold64(hash64B, *((uint64_t *)str+NDhead)); //(( hash64B ^ *((uint64_t *)str+NDhead)) ) * PRIME;
            //hash64C = fold64(hash64C, hash64A);
            //hash64C = fold64(hash64C, hash64B);
        }
    } else {
        //hash64A = ( hash64A ^ _PADr_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3) ) * 1099511628211;
        //hash64B = ( hash64B ^ _PADr_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3) ) * 1099511628211;
        hash64A = fold64( hash64A, _PADr_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3) );
        hash64B = fold64( hash64B, _PADr_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3) );
    }
    //uint64_t hash64 = fold64(hash64A, hash64B);
    //uint64_t hash64 = fold64(hash64, hash64C);
    //uint32_t hash32 = fold32((uint32_t)(hash64), (uint32_t)(hash64>>32));
    //return hash32 & MASK;
    return (uint32_t)fold64(hash64A, hash64B);
    /*#else
    uint64_t hashLH, hashRH;
    __m128i chunkA;
    __m128i stateMIX;
    __m128i stateA = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
    stateMIX = _mm_set1_epi8((char)wrdlen);
    // Initialize hash state with FNV-like seed
    if (wrdlen > 8) { // as a minimum it has to be > 16 but I need NON-AES to handle 0..32
        __m128i stateB = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i stateC = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i chunkB;
        //__m128i len_vec;
        size_t Cycles, NDhead;
        if (wrdlen > 16) {
            Cycles = ((wrdlen - 1)>>5) + 1;
            NDhead = wrdlen - (Cycles<<4);
            /*#pragma nounroll
            for(; Cycles--; str += 16) {
                _mm_prefetch(str+128, _MM_HINT_T0);
                _mm_prefetch(str+NDhead+128, _MM_HINT_T0);
                //hash64A = fold64(hash64A, *((uint64_t *)str)); //(( hash64A ^ *((uint64_t *)str)) ) * PRIME;
                chunkA = _mm_loadu_si128((__m128i *)str);
                stateA = _mm_aesenc_si128(stateA, chunkA);
                //hash64B = fold64(hash64B, *((uint64_t *)str+NDhead)); //(( hash64B ^ *((uint64_t *)str+NDhead)) ) * PRIME;
                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 { // 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);
        // Finalize with length, it strengthens a bit the final
        //len_vec = _mm_set_epi64x(wrdlen+1, wrdlen+1);
        //stateMIX = _mm_aesenc_si128(stateMIX, len_vec);
        // uint64_t result[2];
        // _mm_storeu_si128((__m128i *)result, stateMIX); // Store 128-bit state as four 32-bit values
        // uint64_t hash64 = fold64(result[0], result[1]);
        // return (uint32_t)hash64;
        //uint32_t hash32 = fold32((uint32_t)(hash64), (uint32_t)(hash64>>32));
        //return hash32 & MASK;
    } else {
        //uint64_t hash64A = 14695981039346656037ULL;
        //uint64_t hash64B = 14695981039346656037ULL;
        //uint64_t hash64C = 14695981039346656037ULL;
        //hash64A = ( hash64A ^ _PADr_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3) ) * 1099511628211;
    }
}

```

```

//uint64_t hash64 = fold64(hash64A, hash64B);
//hash64 = fold64(hash64, hash64C);
//return (uint32_t)hash64;
//hashLH = _PAD_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3);
hashRH = _PAD_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3);
//chunkA = _mm_set_epi64x(hashLH, hashRH);
chunkA = _mm_set_epi64x(hashRH, hashRH);
stateMIX = _mm_aesenc_si128(stateMIX, stateA);
stateMIX = _mm_aesenc_si128(stateMIX, chunkA);
//uint32_t hash32 = fold32((uint32_t)(hash64), (uint32_t)(hash64>>32));
//return hash32 & MASK;
}

uint64_t result[2];
_mm_storeu_si128((__m128i *)result, stateMIX); // Store 128-bit state as four 32-bit values
uint64_t hash64 = fold64(result[0], result[1]);
return (uint32_t)hash64;
#endif
}

//      Cycles = ((wrdlen - 1)>>5) + 1;
//      NDhead = wrdlen - (Cycles<<4);
// And some visualization for XMM-WORD:
/*
kl= 33..64 Cycles= (kl-1)/32+1=2; MARGINAL CASES:
2nd head starts at 33-2*16=1 or:
0123456789012345 0123456789012345 0
Head1: [XMM-WORD] [XMM-WORD]
Head2: [XMM-WORD] [XMM-WORD]

2nd head starts at 64-2*16=32 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345
Head1: [XMM-WORD] [XMM-WORD]
Head2: [XMM-WORD] [XMM-WORD] [XMM-WORD] [XMM-WORD]

kl=65..96 Cycles= (kl-1)/32+1=3; MARGINAL CASES:
2nd head starts at 65-3*16=17 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345 0
Head1: [XMM-WORD] [XMM-WORD] [XMM-WORD]
Head2: [XMM-WORD] [XMM-WORD] [XMM-WORD] [XMM-WORD]

2nd head starts at 96-3*16=48 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345 0123456789012345 0123456789012345
Head1: [XMM-WORD] [XMM-WORD] [XMM-WORD]
Head2: [XMM-WORD] [XMM-WORD] [XMM-WORD] [XMM-WORD]

*/
// And some visualization for Q-WORD:
/*
kl= 9..16 Cycles= (kl-1)/16+1=1; MARGINAL CASES:
2nd head starts at 9-1*8=1 or:
012345678
Head1: [Q-WORD]
Head2: [Q-WORD]

2nd head starts at 16-1*8=8 or:
0123456789012345
Head1: [Q-WORD]
Head2: [Q-WORD]

kl=17..24 Cycles= (kl-1)/16+1=2; MARGINAL CASES:
2nd head starts at 17-2*8=1 or:
01234567890123456
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

2nd head starts at 24-2*8=8 or:
012345678901234567890123
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

kl=25..32 Cycles= (kl-1)/16+1=2; MARGINAL CASES:
2nd head starts at 25-2*8=9 or:
0123456789012345678901234
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

2nd head starts at 32-2*8=16 or:
01234567890123456789012345678901
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

kl=33..40 Cycles= (kl-1)/16+1=3; MARGINAL CASES:
2nd head starts at 33-3*8=9 or:
012345678901234567890123456789012
Head1: [Q-WORD][Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD][Q-WORD]

2nd head starts at 40-3*8=16 or:
0123456789012345678901234567890123456789
Head1: [Q-WORD][Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD][Q-WORD]

kl=41..48 Cycles= (kl-1)/16+1=3; MARGINAL CASES:
2nd head starts at 41-3*8=17 or:
01234567890123456789012345678901234567890
Head1: [Q-WORD][Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD][Q-WORD]

```

```

2nd head starts at 48-3*8=24 or:
012345678901234567890123456789012345678901234567
Head1: [Q-WORD][Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD][Q-WORD]
 */

// The more the merrier, therefore I added the 10,000 GitHub stars performer xxhash also:
// https://github.com/Cyan4973/xxHash/issues/1029
//
// Pippip is not an extremely fast hash, it is the spirit of the author materialized disregarding anything outside the "staying true to oneself", or as one bona fide man Otto/Pippip once said:
//
// Translate as verbatim as possible:
// In 1926, Traven wrote that the only biography of a writer should be his
// works: «Die Biographie eines schöpferischen Menschen ist ganz und gar unwichtig.
// Wenn der Mensch in seinen Werken nicht zu erkennen ist, dann ist entweder der
// Mensch nichts wert oder seine Werke sind nichts wert. Darum sollte der schöpferische
// Mensch keine andere Biographie haben als seine Werke» (Hauschild, B. Traven: Die
// unbekannten Jahre, op. cit., p. 31.)
//
// In 1926, Traven wrote that the only biography of a writer should be his works:
// "The biography of a creative person is completely and utterly unimportant.
// If the person is not recognizable in his works, then either the person is worthless or his works are worthless.
// Therefore, the creative person should have no other biography than his works" (Hauschild, B. Traven: Die unbekannten Jahre, op. cit., p. 31.)

// https://godbolt.org/
// x86-64 clang 20.1.0
// -O3 -mavx2
/*
...
*/

// https://godbolt.org/
// x86-64 gcc 14.2
// -O3 -mavx2
/*
...
*/

// gcc -D_N_Pippip -O3 -maes -mavx2 -mbmi2 count_hash_collisions.c -o count_hash_collisions_Pippip.elf.asm -S
// gcc version 14.2.1
/*
FNV1A_Pippip_Yurii_000_128bit_AES_TriXZ:
.LFB6582:
    .cfi_startproc
    vmovd    %esi, %xmm1
    vpbroadcastb %xmm1, %xmm1
    cmpq    $8, %rsi
    jbe     .L28
    cmpq    $16, %rsi
    jbe     .L29
    leaq    -1(%rsi), %rdx
    vmovdqa .LC0(%rip), %xmm0
    shrq    $5, %rdx
    leaq    1(%rdx), %rax
    vmovdqa %xmm0, %xmm4
    salq    $4, %rdx
    vmovdqa %xmm0, %xmm5
    salq    $4, %rax
    leaq    144(%rdi,%rdx), %rdx
    subq    %rax, %rsi
    leaq    128(%rdi), %rax
    .p2align 6
    .p2align 4
    .p2align 3
.L30:
    vmovdqu -128(%rax), %xmm3
    vmovdqu -128(%rax,%rsi), %xmm2
    prefetcht0 (%rax)
    prefetcht0 (%rsi,%rax)
    addq    $16, %rax
    vaesenc %xmm3, %xmm0, %xmm0
    vaesenc %xmm3, %xmm5, %xmm5
    vaesenc %xmm2, %xmm4, %xmm4
    vaesenc %xmm2, %xmm0, %xmm0
    cmpq    %rdx, %rax
    jne     .L30
    jmp     .L31
    .p2align 4,,10
    .p2align 3
.L28:
    movl    $8, %edx
    movq    (%rdi), %rax
    vaesenc .LC0(%rip), %xmm1, %xmm0
    subl    %esi, %edx
    sall    $3, %edx
    shlx    %rdx, %rax, %rax
    vmovq   %rax, %xmm6
    vpunpcklqdq %xmm6, %xmm6, %xmm1
    vaesenc %xmm1, %xmm0, %xmm0
.L32:
    vmovq   %xmm0, %rsi
    vpextrq $1, %xmm0, %rax
    movq    %rsi, %rdx
    mulx    %rax, %rsi, %rax
    xorl    %esi, %eax
    ret

FNV1A_Pippip_128bit_AES_TriXZi_Mikayla (FNV1A-Deathship-Dualheaded a.k.a. FNV1A-Totenköpfe a.k.a. FNV1A-Totenschiff), Update: 2025-Aug-19, Kaze(sanmayce@sanmayce.com), 18/46

```

```

.p2align 4,,10
.p2align 3
.L29:
    vmovdqa .LC0(%rip), %xmm4
    vmovddup (%rdi), %xmm5
    vmovddup -8(%rdi,%rsi), %xmm0
    vaesenc %xmm5, %xmm4, %xmm5
    vaesenc %xmm0, %xmm4, %xmm4
    vaesenc %xmm0, %xmm5, %xmm0
.L31:
    vaesenc %xmm5, %xmm1, %xmm1
    vaesenc %xmm4, %xmm1, %xmm1
    vaesenc %xmm0, %xmm1, %xmm0
    jmp .L32
.cfi_endproc
*/
// The 'Latinic Wordlists Extravaganza' benchmark - hashing all words 1..64 bytes long within De/Es/It/Fr Wikipedia XML dumps:

// -rw-r--r-- 1 sammayce sammayce 30128912152 May 11 23:59 dewiki-20250420-pages-articles.xml
// -rwxrwxrwx 1 sammayce sammayce 7474538699 May 11 23:59 dewiki-20250420-pages-articles.xml.bz2
// -rw-r--r-- 1 sammayce sammayce 23853339437 May 13 05:16 dewiki-20250420-pages-articles.xml.wrd ! Extracted back-to-back all German words
1..64 of size !
// -rw-r--r-- 1 sammayce sammayce 301196511 May 13 07:37 dewiki-20250420-pages-articles.xml.wrd.sorted.unique
// Hashing (using 32bit hashtable i.e. 4,294,967,295 billion slots) all lines of 'dewiki-20250420-pages-articles.xml.wrd.sorted.unique':
// +-----+-----+-----+
// | Hasher, GCC | Total_Hashes i.e. lines | Collisions |
// +-----+-----+-----+
// | komihash 5.27, (64bit hash) | 26,966,343 | 84,628 |
// | a5hash 5.16, (64bit hash) | 26,966,343 | 84,402 |
// | Pippip_AES_TriXZ, (64bit hash) | 26,966,343 | 84,064 | ! Best Dispersion !
// | wypath f4, (64bit hash) | 26,966,343 | 84,579 |
// | XXH3_64bits 0.8.3, (64bit hash) | 26,966,343 | 84,320 |
// +-----+-----+-----+

// -rw-r--r-- 1 sammayce sammayce 21141682874 May 12 01:36 eswiki-20250420-pages-articles.xml
// -rwxrwxrwx 1 sammayce sammayce 4793923558 May 12 01:36 eswiki-20250420-pages-articles.xml.bz2
// -rw-r--r-- 1 sammayce sammayce 16868574569 May 14 19:54 eswiki-20250420-pages-articles.xml.wrd ! Extracted back-to-back all Spanish words
1..64 of size !
// -rw-r--r-- 1 sammayce sammayce 198580774 May 14 21:24 eswiki-20250420-pages-articles.xml.wrd.sorted.unique
// Hashing (using 32bit hashtable i.e. 4,294,967,295 billion slots) all lines of 'eswiki-20250420-pages-articles.xml.wrd.sorted.unique':
// +-----+-----+-----+
// | Hasher, GCC | Total_Hashes i.e. lines | Collisions |
// +-----+-----+-----+
// | komihash 5.27, (64bit hash) | 19,812,803 | 45,367 |
// | a5hash 5.16, (64bit hash) | 19,812,803 | 45,355 |
// | Pippip_AES_TriXZ, (64bit hash) | 19,812,803 | 45,332 | ! Best Dispersion !
// | wypath f4, (64bit hash) | 19,812,803 | 45,653 |
// | XXH3_64bits 0.8.3, (64bit hash) | 19,812,803 | 45,602 |
// +-----+-----+-----+

// -rw-r--r-- 1 sammayce sammayce 29456490747 May 11 23:55 frwiki-20250420-pages-articles.xml
// -rwxrwxrwx 1 sammayce sammayce 6527378243 May 11 23:55 frwiki-20250420-pages-articles.xml.bz2
// -rw-r--r-- 1 sammayce sammayce 23183518578 May 14 17:35 frwiki-20250420-pages-articles.xml.wrd ! Extracted back-to-back all French words
1..64 of size !
// -rw-r--r-- 1 sammayce sammayce 236507028 May 14 19:15 frwiki-20250420-pages-articles.xml.wrd.sorted.unique
// Hashing (using 32bit hashtable i.e. 4,294,967,295 billion slots) all lines of 'frwiki-20250420-pages-articles.xml.wrd.sorted.unique':
// +-----+-----+-----+
// | Hasher, GCC | Total_Hashes i.e. lines | Collisions |
// +-----+-----+-----+
// | komihash 5.27, (64bit hash) | 23,963,517 | 66,985 |
// | a5hash 5.16, (64bit hash) | 23,963,517 | 66,361 | ! Best Dispersion !
// | Pippip_AES_TriXZ, (64bit hash) | 23,963,517 | 66,766 | ! 2nd Best Dispersion !
// | wypath f4, (64bit hash) | 23,963,517 | 67,101 |
// | XXH3_64bits 0.8.3, (64bit hash) | 23,963,517 | 66,869 |
// +-----+-----+-----+

// -rw-r--r-- 1 sammayce sammayce 17364869252 May 11 23:47 itwiki-20250420-pages-articles.xml
// -rwxrwxrwx 1 sammayce sammayce 3980099894 May 11 23:47 itwiki-20250420-pages-articles.xml.bz2
// -rw-r--r-- 1 sammayce sammayce 13707182867 May 14 21:45 itwiki-20250420-pages-articles.xml.wrd ! Extracted back-to-back all Italian words
1..64 of size !
// -rw-r--r-- 1 sammayce sammayce 163763913 May 14 22:43 itwiki-20250420-pages-articles.xml.wrd.sorted.unique
// Hashing (using 32bit hashtable i.e. 4,294,967,295 billion slots) all lines of 'itwiki-20250420-pages-articles.xml.wrd.sorted.unique':
// +-----+-----+-----+
// | Hasher, GCC | Total_Hashes i.e. lines | Collisions |
// +-----+-----+-----+
// | komihash 5.27, (64bit hash) | 16,698,363 | 32,703 |
// | a5hash 5.16, (64bit hash) | 16,698,363 | 32,305 | ! Best Dispersion !
// | Pippip_AES_TriXZ, (64bit hash) | 16,698,363 | 32,561 | ! 3rd Best Dispersion !
// | wypath f4, (64bit hash) | 16,698,363 | 32,811 |
// | XXH3_64bits 0.8.3, (64bit hash) | 16,698,363 | 32,366 |
// +-----+-----+-----+
```

// Example of how the German wordlist was prepared [

```
/*
-rwxrwxrwx 1 sammayce sammayce 3123 May 11 23:00 rip_French_words.c
-rwxrwxrwx 1 sammayce sammayce 16912 May 11 23:00 rip_French_words.elf
-rwxrwxrwx 1 sammayce sammayce 2263 May 11 22:26 rip_German_words.c
-rwxrwxrwx 1 sammayce sammayce 16912 May 11 23:01 rip_German_words.elf
-rwxrwxrwx 1 sammayce sammayce 2482 May 11 22:33 rip_Italian_words.c
-rwxrwxrwx 1 sammayce sammayce 16920 May 11 23:02 rip_Italian_words.elf
-rwxrwxrwx 1 sammayce sammayce 2562 May 11 22:31 rip_Spanish_words.c
-rwxrwxrwx 1 sammayce sammayce 16920 May 11 23:01 rip_Spanish_words.elf
```

[sammayce@leprechaun2 rip_Wikipedias]\$./rip_German_words.elf dewiki-20250420-pages-articles.xml> dewiki-20250420-pages-articles.xml.wrd

```
-rw-r--r-- 1 sammayce sammayce 30128912152 May 11 23:59 dewiki-20250420-pages-articles.xml
-rwxrwxrwx 1 sammayce sammayce 7474538699 May 11 23:59 dewiki-20250420-pages-articles.xml.bz2
```

```
-rw-r--r-- 1 sammayce sammayce 23853339437 May 13 05:16 dewiki-20250420-pages-articles.xml.wrd
```

```
[sammayce@leprechaun2 rip_Wikipedias]$ wc -l dewiki-20250420-pages-articles.xml.wrd  
3573439395 dewiki-20250420-pages-articles.xml.wrd
```

```
[sammayce@leprechaun2 rip_Wikipedias]$ head dewiki-20250420-pages-articles.xml.wrd  
mediawiki  
xmlns  
http  
www  
mediawiki  
org  
xml  
export  
xmlns  
xsi
```

```
[sammayce@leprechaun2 rip_Wikipedias]$ tail dewiki-20250420-pages-articles.xml.wrd  
owizcruo  
y  
kw  
jgocy  
t  
doshjtg  
sha  
revision  
page  
mediawiki
```

```
[sammayce@leprechaun2 rip_Wikipedias]$ head dewiki-20250420-pages-articles.xml  
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.11/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.mediawiki.org/xml/export-0.11/ http://www.mediawiki.org/xml/export-0.11.xsd" version="0.11" xml:lang="de">  
  <siteinfo>  
    <sitename>Wikipedia</sitename>  
    <dbname>dewiki</dbname>  
    <base>https://de.wikipedia.org/wiki/Wikipedia:Hauptseite</base>  
    <generator>MediaWiki 1.44.0-wmf.25</generator>  
    <case>first-letter</case>  
    <namespaces>  
      <namespace key="-2" case="first-letter">Medium</namespace>  
      <namespace key="1" case="first-letter">Spezial</namespace>
```

```
[sammayce@leprechaun2 rip_Wikipedias]$ sort -u dewiki-20250420-pages-articles.xml.wrd -o dewiki-20250420-pages-articles.xml.wrd.sorted.unique  
-rw-r--r-- 1 sammayce sammayce 30128912152 May 11 23:59 dewiki-20250420-pages-articles.xml  
-rwxrwxrwx 1 sammayce sammayce 7474538699 May 11 23:59 dewiki-20250420-pages-articles.xml.bz2  
-rw-r--r-- 1 sammayce sammayce 23853339437 May 13 05:16 dewiki-20250420-pages-articles.xml.wrd  
-rw-r--r-- 1 sammayce sammayce 301196511 May 13 07:37 dewiki-20250420-pages-articles.xml.wrd.sorted.unique
```

```
[sammayce@leprechaun2 rip_Wikipedias]$ wc -l dewiki-20250420-pages-articles.xml.wrd.sorted.unique  
26966343 dewiki-20250420-pages-articles.xml.wrd.sorted.unique
```

```
[sammayce@leprechaun2 rip_Wikipedias]$ head dewiki-20250420-pages-articles.xml.wrd.sorted.unique  
a  
A  
a  
AA  
aa  
Aa  
AA  
aä  
Aä
```

```
[sammayce@leprechaun2 rip_Wikipedias]$  
*/  
// Example of how the German wordlist was prepared ]  
  
// Benchmark 'hashBB' r5_Delphi - 32bit hashtable extravaganza - hashing BBs of 'SUPRAPIG_Delphi_Classics_Complete_Works_of_128_authors.tar'  
1,576,788,480 bytes  
// Fedora Linux Release 41 (MATE-Compiz) 64-bit  
// Kernel Linux 6.12.13-200.fc41.x86_64 x86_64  
// Compiler: gcc 14.2.1  
// Compiler options: gcc -O3 -mavx2 -march=skylake hashBBs_r2.c -o hashBBs_r2_skylake.elf  
// Testmachine: DELL Latitude 7370, m7-6Y75 'Skylake', TDP: 4.5 W (up to 7 W), CPU max 3100 MHz, 2x4GB LPDDR3 1867 MT/s  
// +-----+-----+-----+-----+-----+-----+-----+  
// | Hasher, GCC | BB_Size | Total_Hashes | Unique_Hashes | Collisions | Speed |  
// +-----+-----+-----+-----+-----+-----+-----+  
// | Pippip_AES_TriXZ, (64bit hash) | 2 | 1,576,788,479 | 11,892 | 1,576,776,587 | 0.500 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 3 | 1,576,788,478 | 142,211 | 1,576,646,267 | 0.485 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 4 | 1,576,788,477 | 878,530 | 1,575,909,947 | 0.381 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 5 | 1,576,788,476 | 3,633,307 | 1,573,155,169 | 0.327 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 6 | 1,576,788,475 | 10,997,967 | 1,565,790,508 | 0.317 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 7 | 1,576,788,474 | 26,881,608 | 1,549,906,866 | 0.330 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 8 | 1,576,788,473 | 55,905,056 | 1,520,883,417 | 0.352 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 9 | 1,576,788,472 | 102,323,398 | 1,474,465,074 | 0.376 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 10 | 1,576,788,471 | 168,547,400 | 1,408,241,071 | 0.406 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 11 | 1,576,788,470 | 253,717,510 | 1,323,070,960 | 0.440 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 12 | 1,576,788,469 | 354,433,614 | 1,222,354,855 | 0.473 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 13 | 1,576,788,468 | 465,391,720 | 1,111,396,748 | 0.509 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 14 | 1,576,788,467 | 579,910,854 | 996,877,613 | 0.543 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 15 | 1,576,788,466 | 691,525,242 | 885,263,224 | 0.523 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 16 | 1,576,788,465 | 795,155,891 | 781,632,574 | 0.619 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 18 | 1,576,788,463 | 967,095,589 | 609,692,874 | 0.556 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 20 | 1,576,788,461 | 1,088,218,981 | 488,569,480 | 0.615 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 22 | 1,576,788,459 | 1,166,836,293 | 409,952,166 | 0.674 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 24 | 1,576,788,457 | 1,215,248,992 | 361,539,465 | 0.737 GiB/s |  
// | Pippip_AES_TriXZ, (64bit hash) | 26 | 1,576,788,455 | 1,244,041,242 | 332,747,213 | 0.796 GiB/s |
```

// Pippip_AES_TriXZ, (64bit hash)	28	1,576,788,453	1,260,832,334	315,956,119	0.857 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	30	1,576,788,451	1,270,530,879	306,257,572	0.915 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	32	1,576,788,449	1,276,203,270	300,585,179	0.978 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	48	1,576,788,433	1,286,125,029	290,663,404	1.253 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	64	1,576,788,417	1,288,439,640	288,348,777	1.671 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	80	1,576,788,401	1,290,018,253	286,770,148	1.944 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	96	1,576,788,385	1,291,247,523	285,540,862	2.328 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	128	1,576,788,353	1,293,152,573	283,635,780	2.668 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	256	1,576,788,225	1,297,502,326	279,285,899	3.721 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	512	1,576,787,969	1,301,334,142	275,453,827	5.542 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	1,024	1,576,787,457	1,304,661,606	272,125,851	5.705 GiB/s
// Pippip_AES_TriXZ, (64bit hash)	8,192	1,576,780,289	1,313,075,177	263,705,112	9.392 GiB/s

11,892+142,211+878,530+3,633,307+10,997,967+26,881,608+55,905,056+102,323,398+168,547,400+253,717,510+354,433,614+465,391,720+579,910,854+691,525,242+795,155,891+967,095,589+1,088,218,981+1,166,836,293+1,215,248,992+1,244,041,242+1,260,832,334+1,270,530,879+1,276,283,270+1,286,125,029+1,288,439,640+1,290,018,253+1,291,247,523+1,293,152,573+1,297,502,326+1,301,334,142+1,304,661,606+1,313,075,177=

Total Unique_Hashes = 24,664,020,049 (Dispersion Quality i.e. the-bigger-the-better), 3rd place out of 4

// wyhash f4, (64bit hash)	2	1,576,788,479	11,892	1,576,776,587	0.260 GiB/s
// wyhash f4, (64bit hash)	3	1,576,788,478	142,210	1,576,646,268	0.331 GiB/s
// wyhash f4, (64bit hash)	4	1,576,788,477	878,507	1,575,909,970	0.292 GiB/s
// wyhash f4, (64bit hash)	5	1,576,788,476	3,633,315	1,573,155,161	0.250 GiB/s
// wyhash f4, (64bit hash)	6	1,576,788,475	10,997,967	1,565,790,508	0.235 GiB/s
// wyhash f4, (64bit hash)	7	1,576,788,474	26,880,987	1,549,907,487	0.239 GiB/s
// wyhash f4, (64bit hash)	8	1,576,788,473	55,905,605	1,520,882,868	0.250 GiB/s
// wyhash f4, (64bit hash)	9	1,576,788,472	102,321,573	1,474,466,899	0.267 GiB/s
// wyhash f4, (64bit hash)	10	1,576,788,471	168,548,338	1,408,240,133	0.273 GiB/s
// wyhash f4, (64bit hash)	11	1,576,788,470	253,718,514	1,323,069,956	0.310 GiB/s
// wyhash f4, (64bit hash)	12	1,576,788,469	354,432,053	1,222,356,416	0.334 GiB/s
// wyhash f4, (64bit hash)	13	1,576,788,468	465,393,568	1,111,394,900	0.358 GiB/s
// wyhash f4, (64bit hash)	14	1,576,788,467	579,906,801	996,881,666	0.384 GiB/s
// wyhash f4, (64bit hash)	15	1,576,788,466	691,522,152	885,266,314	0.410 GiB/s
// wyhash f4, (64bit hash)	16	1,576,788,465	795,185,322	781,603,143	0.435 GiB/s
// wyhash f4, (64bit hash)	18	1,576,788,463	967,124,869	609,663,594	0.464 GiB/s
// wyhash f4, (64bit hash)	20	1,576,788,461	1,088,196,724	488,591,737	0.513 GiB/s
// wyhash f4, (64bit hash)	22	1,576,788,459	1,166,869,746	409,918,713	0.565 GiB/s
// wyhash f4, (64bit hash)	24	1,576,788,457	1,215,268,911	361,519,546	0.615 GiB/s
// wyhash f4, (64bit hash)	26	1,576,788,455	1,244,076,171	332,712,284	0.666 GiB/s
// wyhash f4, (64bit hash)	28	1,576,788,453	1,260,833,689	315,954,764	0.717 GiB/s
// wyhash f4, (64bit hash)	30	1,576,788,451	1,270,541,051	306,247,400	0.767 GiB/s
// wyhash f4, (64bit hash)	32	1,576,788,449	1,276,180,805	300,607,644	0.818 GiB/s
// wyhash f4, (64bit hash)	48	1,576,788,433	1,286,111,802	290,676,631	1.093 GiB/s
// wyhash f4, (64bit hash)	64	1,576,788,417	1,288,426,637	288,361,780	1.350 GiB/s
// wyhash f4, (64bit hash)	80	1,576,788,401	1,290,022,729	286,765,672	1.543 GiB/s
// wyhash f4, (64bit hash)	96	1,576,788,385	1,291,250,651	285,537,734	1.478 GiB/s
// wyhash f4, (64bit hash)	128	1,576,788,353	1,293,151,774	283,636,579	1.823 GiB/s
// wyhash f4, (64bit hash)	256	1,576,788,225	1,297,496,736	279,291,489	3.338 GiB/s
// wyhash f4, (64bit hash)	512	1,576,787,969	1,301,362,429	275,425,540	3.513 GiB/s
// wyhash f4, (64bit hash)	1,024	1,576,787,457	1,304,634,872	272,152,585	5.773 GiB/s
// wyhash f4, (64bit hash)	8,192	1,576,780,289	1,313,084,490	263,695,799	13.962 GiB/s

11,892+142,210+878,507+3,633,315+10,997,967+26,880,987+55,905,605+102,321,573+168,548,338+253,718,514+354,432,053+465,393,568+579,906,801+691,522,152+795,185,322+967,124,869+1,088,196,724+1,166,869,746+1,215,268,911+1,244,076,171+1,260,833,689+1,270,541,051+1,276,180,805+1,286,111,802+1,288,426,637+1,290,022,729+1,291,250,651+1,293,151,774+1,297,496,736+1,301,362,429+1,304,634,872+1,313,084,490=

Total Unique_Hashes = 24,664,112,890 (Dispersion Quality i.e. the-bigger-the-better), 1st place out of 4

// komihash 5.27, (64bit hash)	2	1,576,788,479	11,892	1,576,776,587	0.231 GiB/s
// komihash 5.27, (64bit hash)	3	1,576,788,478	142,204	1,576,646,274	0.295 GiB/s
// komihash 5.27, (64bit hash)	4	1,576,788,477	878,513	1,575,909,964	0.283 GiB/s
// komihash 5.27, (64bit hash)	5	1,576,788,476	3,633,240	1,573,155,236	0.243 GiB/s
// komihash 5.27, (64bit hash)	6	1,576,788,475	10,998,131	1,565,790,344	0.229 GiB/s
// komihash 5.27, (64bit hash)	7	1,576,788,474	26,880,788	1,549,907,686	0.233 GiB/s
// komihash 5.27, (64bit hash)	8	1,576,788,473	55,906,102	1,520,882,371	0.238 GiB/s
// komihash 5.27, (64bit hash)	9	1,576,788,472	102,322,531	1,474,465,941	0.256 GiB/s
// komihash 5.27, (64bit hash)	10	1,576,788,471	168,546,813	1,408,241,658	0.275 GiB/s
// komihash 5.27, (64bit hash)	11	1,576,788,470	253,720,484	1,323,067,986	0.296 GiB/s
// komihash 5.27, (64bit hash)	12	1,576,788,469	354,433,776	1,222,354,693	0.322 GiB/s
// komihash 5.27, (64bit hash)	13	1,576,788,468	465,404,536	1,111,383,932	0.346 GiB/s
// komihash 5.27, (64bit hash)	14	1,576,788,467	579,916,059	996,872,408	0.370 GiB/s
// komihash 5.27, (64bit hash)	15	1,576,788,466	691,508,865	885,279,601	0.395 GiB/s
// komihash 5.27, (64bit hash)	16	1,576,788,465	795,195,395	781,595,070	0.404 GiB/s
// komihash 5.27, (64bit hash)	18	1,576,788,463	967,124,401	609,664,062	0.453 GiB/s
// komihash 5.27, (64bit hash)	20	1,576,788,461	1,088,201,898	488,586,563	0.502 GiB/s
// komihash 5.27, (64bit hash)	22	1,576,788,459	1,166,840,670	409,947,789	0.553 GiB/s
// komihash 5.27, (64bit hash)	24	1,576,788,457	1,215,258,203	361,530,254	0.601 GiB/s
// komihash 5.27, (64bit hash)	26	1,576,788,455	1,244,050,649	332,737,806	0.650 GiB/s
// komihash 5.27, (64bit hash)	28	1,576,788,453	1,260,830,316	315,958,137	0.701 GiB/s
// komihash 5.27, (64bit hash)	30	1,576,788,451	1,270,494,766	306,293,685	0.748 GiB/s
// komihash 5.27, (64bit hash)	32	1,576,788,449	1,276,161,279	300,627,170	0.663 GiB/s
// komihash 5.27, (64bit hash)	48	1,576,788,433	1,286,119,995	290,668,438	0.941 GiB/s
// komihash 5.27, (64bit hash)	64	1,576,788,417	1,288,426,362	288,362,055	0.884 GiB/s
// komihash 5.27, (64bit hash)	80	1,576,788,401	1,290,013,842	286,774,559	1.074 GiB/s
// komihash 5.27, (64bit hash)	96	1,576,788,385	1,291,264,494	285,523,891	1.261 GiB/s
// komihash 5.27, (64bit hash)	128	1,576,788,353	1,293,132,802	283,655,551	1.643 GiB/s
// komihash 5.27, (64bit hash)	256	1,576,788,225	1,297,485,629	279,302,596	1.874 GiB/s
// komihash 5.27, (64bit hash)	512	1,576,787,969	1,301,326,829	275,461,140	3.201 GiB/s
// komihash 5.27, (64bit hash)	1,024	1,576,787,457	1,304,642,072	272,145,385	5.119 GiB/s
// komihash 5.27, (64bit hash)	8,192	1,576,780,289	1,313,079,327	263,700,962	10.793 GiB/s

11,892+142,204+878,513+3,633,240+10,998,131+26,880,788+55,906,102+102,322,531+168,546,813+253,720,484+354,433,776+465,404,536+579,916,059+691,508,865+795,193,395+967,124,401+1,088,201,898+1,166,840,670+1,215,258,203+1,244,050,649+1,260,830,316+1,270,494,766+1,276,161,279+1,286,119,995+1,288,426,362+1,290,013,842+1,291,264,494+1,293,132,802+1,297,485,629+1,301,326,829+1,304,642,072+1,313,079,327=


```
#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
}
uint32_t FNV1A_Pippip_Yurii_000_128bit_AES_TriXZ(const char *str, size_t wrdlen) {
    #ifdef AESless
        uint64_t hash64A = 14695981039346656037ULL;
        uint64_t hash64B = 14695981039346656037ULL;
    if (wrdlen > 8) {
        size_t Cycles, NDhead;
        Cycles = ((wrdlen - 1)>>4) + 1;
        NDhead = wrdlen - (Cycles<<3);
        for(; Cycles--; str += 8) {
            hash64A = fold64(hash64A, *((uint64_t *)str));
            hash64B = fold64(hash64B, *((uint64_t *)str+NDhead)));
        }
    } else {
        hash64A = fold64( hash64A, _PADr_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3));
        hash64B = fold64( hash64B, _PADr_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3));
    }
    return (uint32_t)fold64(hash64A, hash64B);           The author: Georgi 'Sanmayce' Marinov, rendered by PixAI.art from an original photo
    #else
        uint64_t hashLH, hashRH;
        __m128i chunkA;
        __m128i stateMIX;
        __m128i stateA = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        stateMIX = _mm_set1_epi8((char)wrdlen);
    if (wrdlen > 8) {
        __m128i stateB = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i stateC = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i chunkB;
        size_t Cycles, NDhead;
        if (wrdlen > 16) {
            Cycles = ((wrdlen - 1)>>5) + 1;
            NDhead = wrdlen - (Cycles<<4);
            /*#pragma nounroll
            for(; Cycles--; str += 16) {
                _mm_prefetch(str+128, _MM_HINT_T0);
                _mm_prefetch(str+NDhead+128, _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 { // 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 {
        hashRH = _PAD_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3);
        chunkA = _mm_set_epi64x(hashRH, hashRH);
        stateMIX = _mm_aesenc_si128(stateMIX, stateA);
        stateMIX = _mm_aesenc_si128(stateMIX, chunkA);
    }
    uint64_t result[2];
    _mm_storeu_si128((__m128i *)result, stateMIX);
    uint64_t hash64 = fold64(result[0], result[1]);
    return (uint32_t)hash64;
    #endif
}
```



// [https://forums.fedoraforum.org/showthread.php?334463-In-search-for-the-FASTEAST-hash-\(for-lookup-tables\)](https://forums.fedoraforum.org/showthread.php?334463-In-search-for-the-FASTEAST-hash-(for-lookup-tables))
 // <https://github.com/Cyan4973/xxHash/issues/1029/>

```
#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
}
uint32_t FNV1A_Pippip_Yurii_000_128bit_AES_TriXZ(const char *str, size_t wrdlen) {
    #ifdef AESless
        uint64_t hash64A = 14695981039346656037ULL;
        uint64_t hash64B = 14695981039346656037ULL;
    if (wrdlen > 8) {
        size_t Cycles, NDhead;
        Cycles = ((wrdlen - 1)>>4) + 1;
        NDhead = wrdlen - (Cycles<<3);
        for(; Cycles--; str += 8) {
            hash64A = fold64(hash64A, *((uint64_t *)str));
            hash64B = fold64(hash64B, *((uint64_t *)str+NDhead)));
        }
    } else {
        hash64A = fold64( hash64A, _PADr_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3));
        hash64B = fold64( hash64B, _PAD_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3));
    }
    return (uint32_t)fold64(hash64A, hash64B);
    #else
        uint64_t hashLH, hashRH;
        __m128i chunkA;
        __m128i stateMIX;
        __m128i stateA = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        stateMIX = _mm_set1_epi8((char)wrdlen);
    if (wrdlen > 8) {
        __m128i stateB = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i stateC = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i chunkB;
        size_t Cycles, NDhead;
    if (wrdlen > 16) {
        Cycles = ((wrdlen - 1)>>5) + 1;
        NDhead = wrdlen - (Cycles<<4);
        /*#pragma nounroll
        for(; Cycles--; str += 16) {
            _mm_prefetch(str+128, _MM_HINT_T0);
            _mm_prefetch(str+NDhead+128, _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 { // 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 {
    hashRH = _PAD_KAZE(*((uint64_t *)str+0), (8-wrdlen)<<3);
    chunkA = _mm_set_epi64x(hashRH, hashRH);
    stateMIX = _mm_aesenc_si128(stateMIX, stateA);
    stateMIX = _mm_aesenc_si128(stateMIX, chunkA);
}
    uint64_t result[2];
    _mm_storeu_si128((__m128i *)result, stateMIX);
    uint64_t hash64 = fold64(result[0], result[1]);
    return (uint32_t)hash64;
    #endif
}
```



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



// [https://forums.fedoraproject.org/showthread.php?334463-In-search-for-the-FAASTEST-hash-\(for-lookup-tables\)](https://forums.fedoraproject.org/showthread.php?334463-In-search-for-the-FAASTEST-hash-(for-lookup-tables))
 // <https://github.com/Cyan4973/xxHash/issues/1029/>

```
#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
}
uint32_t FNV1A_Pippip_Yurii_000_128bit_AES_TriXZ(const char *str, size_t wrdlen) {
    #ifdef AESless
        uint64_t hash64A = 14695981039346656037ULL;
        uint64_t hash64B = 14695981039346656037ULL;
    if (wrdlen > 8) {
        size_t Cycles, NDhead;
        Cycles = ((wrdlen - 1)>>4) + 1;
        NDhead = wrdlen - (Cycles<<3);
        for(; Cycles--; str += 8) {
            hash64A = fold64(hash64A, *((uint64_t *) (str)));
            hash64B = fold64(hash64B, *((uint64_t *) (str+NDhead)));
        }
    } else {
        hash64A = fold64( hash64A, _PADr_KAZE(*((uint64_t *) (str+0)), (8-wrdlen)<<3));
        hash64B = fold64( hash64B, _PAD_KAZE(*((uint64_t *) (str+0)), (8-wrdlen)<<3));
    }
    return (uint32_t)fold64(hash64A, hash64B);           The author: Georgi 'Sanmayce' Marinov, rendered by PixAI.art from an original photo
    #else
        uint64_t hashLH, hashRH;
        __m128i chunkA;
        __m128i stateMIX;
        __m128i stateA = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        stateMIX = _mm_set1_epi8((char)wrdlen);
    if (wrdlen > 8) {
        __m128i stateB = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i stateC = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        __m128i chunkB;
        size_t Cycles, NDhead;
    if (wrdlen > 16) {
        Cycles = ((wrdlen - 1)>>5) + 1;
        NDhead = wrdlen - (Cycles<<4);
        /*#pragma nounroll
        for(; Cycles--; str += 16) {
            _mm_prefetch(str+128, _MM_HINT_T0);
            _mm_prefetch(str+NDhead+128, _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 { // 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 {
    hashRH = _PAD_KAZE(*((uint64_t *) (str+0)), (8-wrdlen)<<3);
    chunkA = _mm_set_epi64x(hashRH, hashRH);
    stateMIX = _mm_aesenc_si128(stateMIX, stateA);
    stateMIX = _mm_aesenc_si128(stateMIX, chunkA);
}
    uint64_t result[2];
    _mm_storeu_si128((__m128i *)result, stateMIX);
    uint64_t hash64 = fold64(result[0], result[1]);
    return (uint32_t)hash64;
#endif
}
```



// [https://forums.fedoraforum.org/showthread.php?334463-In-search-for-the-FASTEAST-hash-\(for-lookup-tables\)](https://forums.fedoraforum.org/showthread.php?334463-In-search-for-the-FASTEAST-hash-(for-lookup-tables))
 // <https://github.com/Cyan4973/xxHash/issues/1029/>

```
// gcc -D_N_Pippip -O3 -maes -mavx2 -mbmi2 count_hash_collisions.c -o count_hash_collisions_Pippip.elf.asm -S
// gcc version 14.2.1
```

FNV1A_Pippip_Yurii_000_128bit_AES_TriXZ:

```
.LFB6582:
    .cfi_startproc
    vmovd    %esi, %xmm1
    vpbroadcastb   %xmm1, %xmm1
    cmpq    $8, %rsi
    jbe     .L28
    cmpq    $16, %rsi
    jbe     .L29
    leaq    -1(%rsi), %rdx
    vmovdqa .LC0(%rip), %xmm0
    shrq    $5, %rdx
    leaq    1(%rdx), %rax
    vmovdqa %xmm0, %xmm4
    salq    $4, %rdx
    vmovdqa %xmm0, %xmm5
    salq    $4, %rax
    leaq    144(%rdi,%rdx), %rdx
    subq    %rax, %rsi
    leaq    128(%rdi), %rax
    .p2align 6
    .p2align 4
    .p2align 3
.L30:
    vmovdqu -128(%rax), %xmm3
    vmovdqu -128(%rax,%rsi), %xmm2
    prefetcht0 (%rax)
    prefetcht0 (%rsi,%rax)
    addq    $16, %rax
    vaesenc %xmm3, %xmm0, %xmm0
    vaesenc %xmm3, %xmm5, %xmm5
    vaesenc %xmm2, %xmm4, %xmm4
    vaesenc %xmm2, %xmm0, %xmm0
    cmpq    %rdx, %rax
    jne     .L30
    jmp     .L31
    .p2align 4,,10
    .p2align 3
.L28:
    movl    $8, %edx
    movq    (%rdi), %rax
    vaesenc .LC0(%rip), %xmm1, %xmm0
    subl    %esi, %edx
    sall    $3, %edx
    shlx    %rdx, %rax, %rax
    vmovq    %rax, %xmm6
    vpunpcklqdq %xmm6, %xmm6, %xmm1
    vaesenc %xmm1, %xmm0, %xmm0
.L32:
    vmovq    %xmm0, %rsi
    vpxtrq    $1, %xmm0, %rax
    movq    %rsi, %rdx
    mulx    %rax, %rsi, %rax
    xorl    %esi, %eax
    ret
    .p2align 4,,10
    .p2align 3
.L29:
    vmovdqa .LC0(%rip), %xmm4
    vmovddup (%rdi), %xmm5
    vmovddup -8(%rdi,%rsi), %xmm0
    vaesenc %xmm5, %xmm4, %xmm5
    vaesenc %xmm0, %xmm4, %xmm4
    vaesenc %xmm0, %xmm5, %xmm0
.L31:
    vaesenc %xmm5, %xmm1, %xmm1
    vaesenc %xmm4, %xmm1, %xmm1
    vaesenc %xmm0, %xmm1, %xmm0
    jmp     .L32
    .cfi_endproc
```

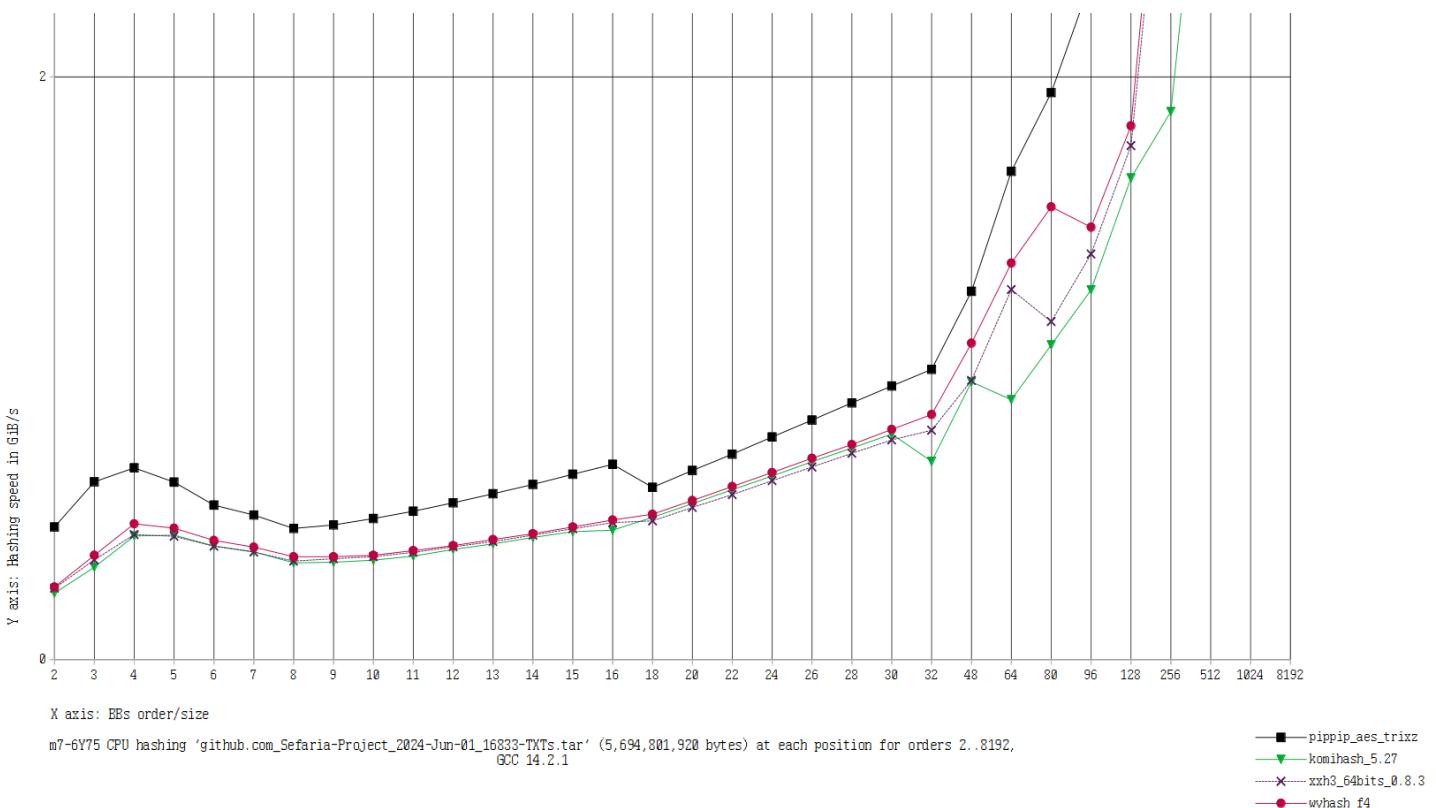
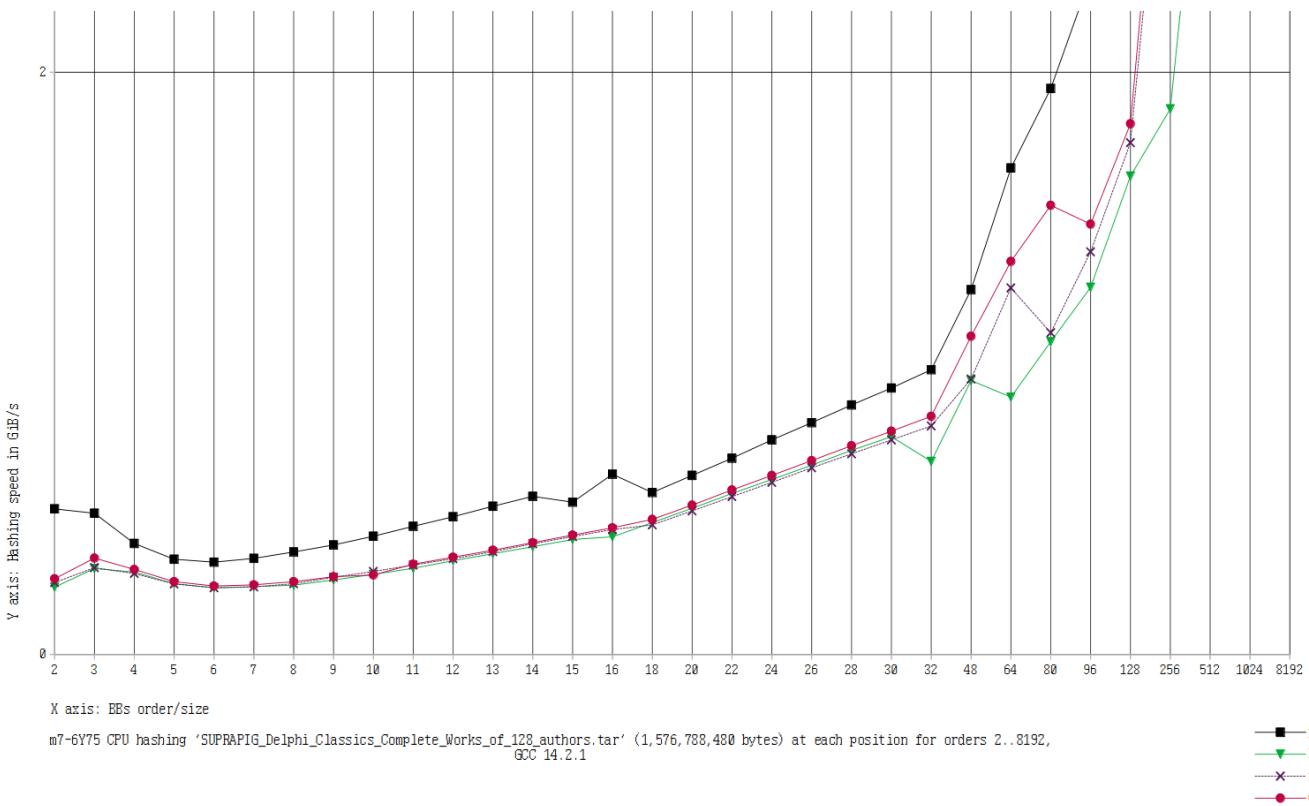
In 1926, Traven wrote that the only biography of a writer should be his works:

"The biography of a creative person is completely and utterly unimportant. If the person is not recognizable in his works, then either the person is worthless or his works are worthless. Therefore, the creative person should have no other biography than his works."

«Die Biographie eines schöpferischen Menschen ist ganz und gar unwichtig. Wenn der Mensch in seinen Werken nicht zu erkennen ist, dann ist entweder der Mensch nichts wert oder seine Werke sind nichts wert. Darum sollte der schöpferische Mensch keine andere Biographie haben als seine Werke.»
 (Hauschild, B. Traven: Die unbekannten Jahre, op. cit., p. 31.)

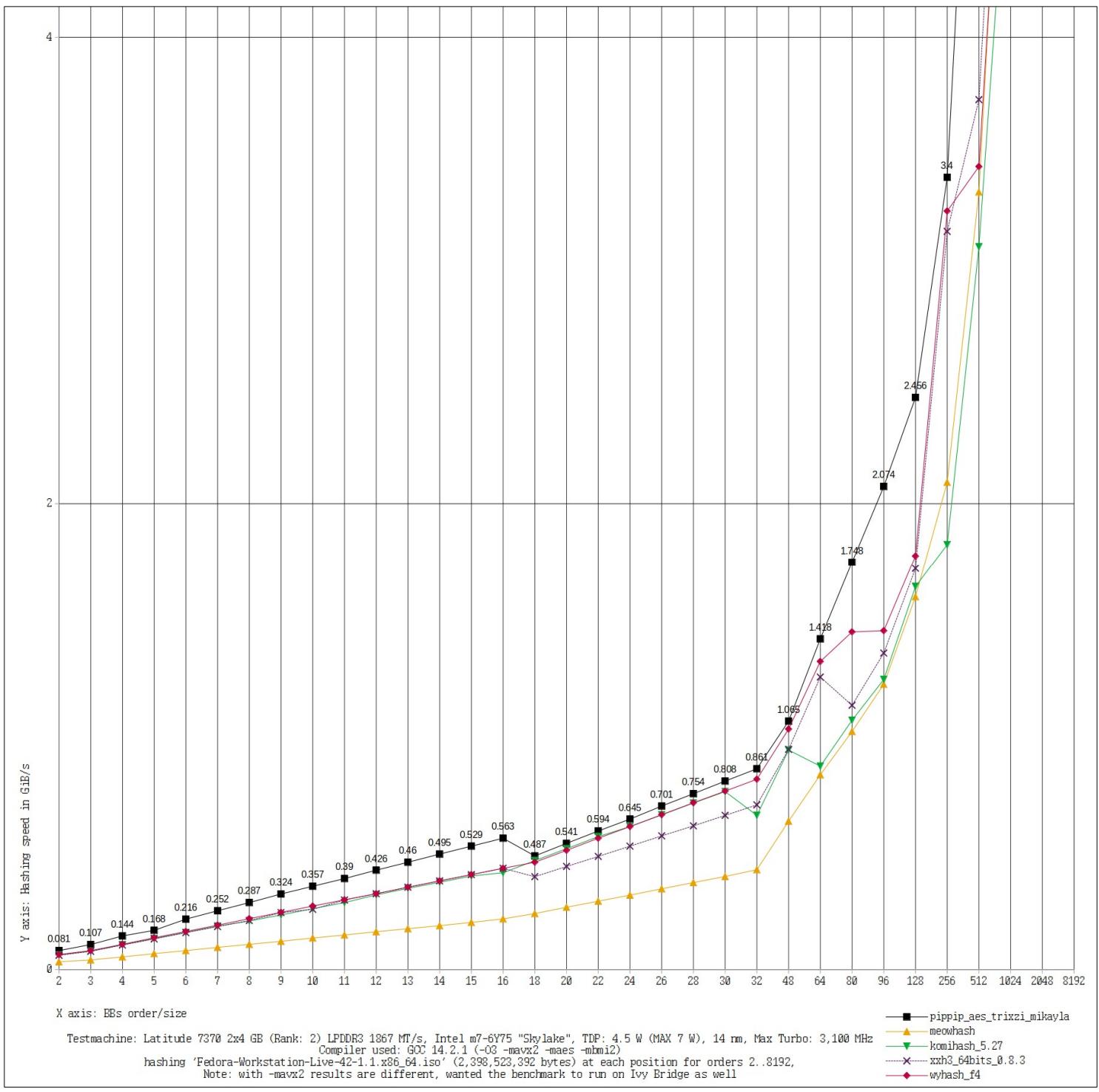


```
// "There it now stands for ever. Black on white.
// I can't get away from it. Ahoy, Yorikke, ahoy, hoy, ho!
// Go to hell now if you wish. What do I care? It's all the same now to me.
// I am part of you now. Where you go I go, where you leave I leave, when you
go to the devil I go. Married.
// Vanished from the living. Damned and doomed. Of me there is not left a
breath in all the vast world.
// Ahoy, Yorikke! Ahoy, hoy, ho!
// I am not buried in the sea,
// The death ship is now part of me
// So far from sunny New Orleans
// So far from lovely Louisiana."
// An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by
B.TRAVEN/
// "Walking home to our good old Yorikke, I could not help thinking of this
beautiful ship, with a crew on board that had faces as if they were seeing
ghosts by day and by night.
// Compared to that gilded Empress, the Yorikke was an honorable old lady with
lavender sachets in her drawers.
// Yorikke did not pretend to anything she was not. She lived up to her looks.
Honest to her lowest ribs and to the leaks in her bilge.
// Now, what is this? I find myself falling in love with that old jane.
// All right, I cannot pass by you, Yorikke; I have to tell you I love you.
Honest, baby, I love you.
// I have six black finger-nails, and four black and green-blue nails on my
toes, which you, honey, gave me when necking you.
// Grate-bars have crushed some of my toes. And each finger-nail has its own
painful story to tell.
// My chest, my back, my arms, my legs are covered with scars of burns and
scorchings.
// Each scar, when it was being created, caused me pains which I shall surely
never forget.
// But every outcry of pain was a love-cry for you, honey.
// You are no hypocrite. Your heart does not bleed tears when you do not feel
heart-aches deeply and truly.
// You do not dance on the water if you do not feel like being jolly and
kicking chasers in the pants.
// Your heart never lies. It is fine and clean like polished gold. Never mind
the rags, honey dear.
// When you laugh, your whole soul and all your body is laughing.
// And when you weep, sweetly, then you weep so that even the reefs you pass
feel like weeping with you.
// I never want to leave you again, honey. I mean it. Not for all the rich and
elegant buckets in the world.
// I love you, my gypsy of the sea!"
// An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by
B.TRAVEN/
// Dedicated to Pippip, the main character in the 'Das Totenschiff' roman,
actually the B.Traven himself, his real name was Hermann Albert Otto
MakSYMILIAN Feige.
// CAUTION: Add 8 more bytes to the buffer being hashed, usually malloc(...+8)
- to prevent out of boundary reads!
// Many thanks go to Yurii 'Hordi' Hordienko, he lessened with 3 instructions
the original 'Pippip'.
```



~~Fastest Hash for lookuring~~





Contents

- 1] <https://github.com/Sanmayce/Pippip/blob/main/Pippip.h>
- 2] <https://github.com/Sanmayce/Pippip/blob/main/Pippipify.c>
- 3] https://github.com/Sanmayce/Pippip/blob/main/pippip_hash_arm.c
- 4] <https://github.com/rurban/smhasher/issues/322>
- 5] <https://github.com/Sanmayce/DoubleDeuce/releases/tag/r7>
- 6] <https://pixai.art/@user-1720950639428298604/artworks>
- 7] Hashing Fedora42 on m7 CPU
- 8] chatGPT oversimplified Pippipify

```
1]
// FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla: the 100% FREE lookuper, last update: 2025-Jul-17, Kaze (sanmayce@sanmayce.com). [
// This latest revision was written when Mikayla "saveafox" left this world.
```

```
// "There it now stands for ever. Black on white.  

// I can't get away from it. Ahoy, Yorikke, ahoy, hoy, ho!  

// Go to hell now if you wish. What do I care? It's all the same now to me.  

// I am part of you now. Where you go I go, where you leave I leave, when you go to the devil I go. Married.  

// Vanished from the living. Damned and doomed. Of me there is not left a breath in all the vast world.  

// Ahoy, Yorikke! Ahoy, hoy, ho!  

// I am not buried in the sea,  

// The death ship is now part of me  

// So far from sunny New Orleans  

// So far from lovely Louisiana."  

// /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B.TRAVEN/
//
```

```

// "Walking home to our good old Yorikke, I could not help thinking of this beautiful ship, with a crew on board that had faces as if they
// were seeing ghosts by day and by night.
// Compared to that gilded Empress, the Yorikke was an honorable old lady with lavender sachets in her drawers.
// Yorikke did not pretend to anything she was not. She lived up to her looks. Honest to her lowest ribs and to the leaks in her bilge.
// Now, what is this? I find myself falling in love with that old jane.
// All right, I cannot pass by you, Yorikke; I have to tell you I love you. Honest, baby, I love you.
// I have six black finger-nails, and four black and green-blue nails on my toes, which you, honey, gave me when necking you.
// Grate-bars have crushed some of my toes. And each finger-nail has its own painful story to tell.
// My chest, my back, my arms, my legs are covered with scars of burns and scorchings.
// Each scar, when it was being created, caused me pains which I shall surely never forget.
// But every outcry of pain was a love-cry for you, honey.
// You are no hypocrite. Your heart does not bleed tears when you do not feel heart-aches deeply and truly.
// You do not dance on the water if you do not feel like being jolly and kicking chasers in the pants.
// Your heart never lies. It is fine and clean like polished gold. Never mind the rags, honey dear.
// When you laugh, your whole soul and all your body is laughing.
// And when you weep, sweetly, then you weep so that even the reefs you pass feel like weeping with you.
// I never want to leave you again, honey. I mean it. Not for all the rich and elegant buckets in the world.
// I love you, my gypsy of the sea!"
```

// An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B.TRAVEN/
//
// Dedicated to Pippip, the main character in the 'Das Totenschiff' roman, actually the B.Trauen himself, his real name was Hermann Albert Otto Maksymilian Feige.
// CAUTION: Add 8 more bytes to the buffer being hashed, usually malloc(...+8) - to prevent out of boundary reads!
// Many thanks go to Yurii 'Hordi' Hordienko, he lessened with 3 instructions the original 'Pippip', thus:
//

```

#include <stdlib.h>
#include <stdint.h>
#include <immintrin.h>
#define HashInBITS 32 // default 26 i.e. 2^26 i.e. 64MS(Mega Slots); slots contain 8bytes pointers or 512MB
free (1GB in total)!
#define MASK ((1ULL << HashInBITS) - 1)
#define _PADr_KAZE(x, n) ( ((x) << (n))>>(n) )
#define _PAD_KAZE(x, n) ( ((x) << (n)) )

typedef struct {
    uint64_t low;
    uint64_t high;
} hashPippip128_t;

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
}

static inline uint32_t fold32(uint32_t A, uint32_t B) {
    #if defined(_GNUC_) || defined(_clang_)
        uint64_t r = (uint64_t)A * (uint64_t)B;
        return (uint32_t)r ^ (uint32_t)(r >> 32);
    #else
        uint32_t hash32 = A ^ B;
        hash32 *= 591798841;
        return hash32;
    #endif
}

void FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla (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, 0x9e3779b97fda7c15);
        __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);
                    stateB = _mm_aesenc_si128(stateB, chunkB);
                    stateC = _mm_aesenc_si128(stateC, chunkB);
                }
            } 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);
}
#endif 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
}

//      Cycles = ((wrdlen - 1)>>5) + 1;
//      NDhead = wrdlen - (Cycles<<4);
// And some visualization for XMM-WORD:
/*
kl= 33..64 Cycles= (kl-1)/32+1=2; MARGINAL CASES:
2nd head starts at 33-2*16=1 or:
0123456789012345 0123456789012345 0
Head1: [XMM-WORD] [XMM-WORD]
Head2: [XMM-WORD] [XMM-WORD]

2nd head starts at 64-2*16=32 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345
Head1: [XMM-WORD] [XMM-WORD]
Head2: [XMM-WORD] [XMM-WORD] [XMM-WORD] [XMM-WORD]

kl=65..96 Cycles= (kl-1)/32+1=3; MARGINAL CASES:
2nd head starts at 65-3*16=17 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345 0
Head1: [XMM-WORD] [XMM-WORD] [XMM-WORD]
Head2: [XMM-WORD] [XMM-WORD] [XMM-WORD] [XMM-WORD]

2nd head starts at 96-3*16=48 or:
0123456789012345 0123456789012345 0123456789012345 0123456789012345 0123456789012345 0123456789012345
Head1: [XMM-WORD] [XMM-WORD] [XMM-WORD]
Head2: [XMM-WORD] [XMM-WORD] [XMM-WORD] [XMM-WORD] [XMM-WORD]

*/
// And some visualization for Q-WORD:
/*
kl= 9..16 Cycles= (kl-1)/16+1=1; MARGINAL CASES:
2nd head starts at 9-1*8=1 or:
012345678
Head1: [Q-WORD]
Head2: [Q-WORD]

2nd head starts at 16-1*8=8 or:
0123456789012345
Head1: [Q-WORD]
Head2: [Q-WORD]

kl=17..24 Cycles= (kl-1)/16+1=2; MARGINAL CASES:
2nd head starts at 17-2*8=1 or:
01234567890123456
Head1: [Q-WORD][Q-WORD]
Head2: [Q-WORD][Q-WORD]

2nd head starts at 24-2*8=8 or:
012345678901234567890123
Head1: [Q-WORD][Q-WORD]

```

```

Head2: [Q-WORD][Q-WORD]

k1=25..32 Cycles= (k1-1)/16+1=2; MARGINAL CASES:
    2nd head starts at 25-2*8=9 or:
        0123456789012345678901234
    Head1: [Q-WORD][Q-WORD]
    Head2: [Q-WORD][Q-WORD]

    2nd head starts at 32-2*8=16 or:
        01234567890123456789012345678901
    Head1: [Q-WORD][Q-WORD]
    Head2: [Q-WORD][Q-WORD]

k1=33..40 Cycles= (k1-1)/16+1=3; MARGINAL CASES:
    2nd head starts at 33-3*8=9 or:
        012345678901234567890123456789012
    Head1: [Q-WORD][Q-WORD][Q-WORD]
    Head2: [Q-WORD][Q-WORD][Q-WORD]

    2nd head starts at 40-3*8=16 or:
        0123456789012345678901234567890123456789
    Head1: [Q-WORD][Q-WORD][Q-WORD]
    Head2: [Q-WORD][Q-WORD][Q-WORD]

k1=41..48 Cycles= (k1-1)/16+1=3; MARGINAL CASES:
    2nd head starts at 41-3*8=17 or:
        01234567890123456789012345678901234567890
    Head1: [Q-WORD][Q-WORD][Q-WORD]
    Head2: [Q-WORD][Q-WORD][Q-WORD]

    2nd head starts at 48-3*8=24 or:
        012345678901234567890123456789012345678901234567
    Head1: [Q-WORD][Q-WORD][Q-WORD]
    Head2: [Q-WORD][Q-WORD][Q-WORD]

/*
// The more the merrier, therefore I added the 10,000 GitHub stars performer xxhash also:
// https://github.com/Cyan4973/xxHash/issues/1029
//
// Pippip is not an extremely fast hash, it is the spirit of the author materialized disregarding anything outside the "staying true to oneself", or as one bona fide man Otto/Pippip once said:
//
// Translate as verbatim as possible:
// In 1926, Traven wrote that the only biography of a writer should be his
// works: «Die Biographie eines schöpferischen Menschen ist ganz und gar unwichtig.
// Wenn der Mensch in seinen Werken nicht zu erkennen ist, dann ist entweder der
// Mensch nichts wert oder seine Werke sind nichts wert. Darum sollte der schöpferische
// Mensch keine andere Biographie haben als seine Werke» (Hauschild, B. Traven: Die
// unbekannten Jahre, op. cit., p. 31.)
//
// In 1926, Traven wrote that the only biography of a writer should be his works:
// "The biography of a creative person is completely and utterly unimportant.
// If the person is not recognizable in his works, then either the person is worthless or his works are worthless.
// Therefore, the creative person should have no other biography than his works" (Hauschild, B. Traven: Die unbekannten Jahre, op. cit., p.
// 31.)

// https://godbolt.org/
// x86-64 clang 20.1.0
// -O3 -maes -mbmi2
/*
...
*/

// https://godbolt.org/
// x86-64 gcc 14.2
// -O3 -maes -mbmi2
/*
...
*/

2]
// Pippipify.c, written by Grok, 2025-Jul-25

/*
[sanmayce@d8 hashBBs_ver7]$ gcc Pippipify.c -maes -o Pippipify.elf
[sanmayce@d8 hashBBs_ver7]$ ls -l
-rw-r--r-- 1 sammayce sammayce 5694801920 Jun  1 2024 github.com_Sefaria-Project_2024-Jun-01_16833-TXTs.tar

[sanmayce@d8 hashBBs_ver7]$ ./Pippipify.elf github.com_Sefaria-Project_2024-Jun-01_16833-TXTs.tar
Streaming Mode (chunk size: 4096 bytes):
32-bit hash: 0xee686b67
64-bit hash: 0xe11caa51ee686b67
128-bit hash: 0xfe2a30199f0cb14d46071046668b6a2f

In-One-Go Mode:
Debug: File size = 5694801920 bytes
Debug: Successfully read 5694801920 bytes
Debug: Successfully initialized context
Debug: Successfully updated context
Debug: Successfully finalized 32-bit hash
32-bit hash: 0xee686b67
64-bit hash: 0xe11caa51ee686b67
128-bit hash: 0xfe2a30199f0cb14d46071046668b6a2f
[sanmayce@d8 hashBBs_ver7]$ */

#include <stdio.h>
#include <stdlib.h>

FNV1A_Pippip_128bit_AES_TriXZi_Mikayla (FNV1A-Deathship-Dualheaded a.k.a. FNV1A-Totenköpfe a.k.a. FNV1A-Totenschiff), Update: 2025-Aug-19, Kaze(sanmayce@sanmayce.com), 32/46

```

```

#include <stdint.h>
#include <string.h>
#include <immintrin.h>

// Default hash size is 32 bits unless eXdupe is defined (128 bits).
// HashInBITS defines the slot size for deduplication (default 2^26 = 64M slots,
// each containing an 8-byte pointer, requiring 512MB total).
#define HashInBITS 32
#define MASK ((1ULL << HashInBITS) - 1)
#define _PADr_KAZE(x, n) (((x) << (n)) >> (n))
#define _PAD_KAZE(x, n) (((x) << (n)))

// Chunk size for streaming mode (4KB = 2^12, aligns with Lasse's requirement)
#define CHUNK_SIZE 4096

typedef struct {
    uint64_t low;
    uint64_t high;
} hashPippip128_t;

// Streaming context for incremental hashing
typedef struct {
    __m128i stateMIX; // Current hash state
    uint64_t total_bytes; // Total bytes processed
    uint32_t seed; // Initial seed
    uint8_t buffer[16]; // Buffer for partial chunks
    size_t buffer_len; // Bytes in buffer
} PippipContext;

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;
        return hash64;
    #endif
}

static inline uint32_t fold32(uint32_t A, uint32_t B) {
    #if defined(__GNUC__) || defined(__clang__)
        uint64_t r = (uint64_t)A * (uint64_t)B;
        return (uint32_t)r ^ (uint32_t)(r >> 32);
    #else
        uint32_t hash32 = A ^ B;
        hash32 *= 591798841;
        return hash32;
    #endif
}

// Initialize streaming context
void pippip_init(PippipContext *ctx, uint32_t seed) {
    ctx->stateMIX = _mm_set1_epi32((uint32_t)0 ^ seed);
    ctx->total_bytes = 0;
    ctx->seed = seed;
    ctx->buffer_len = 0;
}

// Update streaming context with a chunk of data
void pippip_update(PippipContext *ctx, const char *data, size_t len) {
    size_t i = 0;
    __m128i stateA, stateB, stateC, chunkA;

    // Process buffered data if any
    if (ctx->buffer_len > 0) {
        size_t to_copy = 16 - ctx->buffer_len < len ? 16 - ctx->buffer_len : len;
        memcpy(ctx->buffer + ctx->buffer_len, data, to_copy);
        ctx->buffer_len += to_copy;
        data += to_copy;
        len -= to_copy;
        ctx->total_bytes += to_copy;
    }

    if (ctx->buffer_len == 16) {
        chunkA = _mm_loadu_si128((__m128i *)ctx->buffer);
        stateA = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        stateB = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        stateC = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        stateA = _mm_aesenc_si128(stateA, chunkA);
        stateC = _mm_aesenc_si128(stateC, chunkA);
        stateB = _mm_aesenc_si128(stateB, chunkA);
        stateC = _mm_aesenc_si128(stateC, chunkA);
        ctx->stateMIX = _mm_aesenc_si128(ctx->stateMIX, stateA);
        ctx->stateMIX = _mm_aesenc_si128(ctx->stateMIX, stateB);
        ctx->stateMIX = _mm_aesenc_si128(ctx->stateMIX, stateC);
        ctx->buffer_len = 0;
    }

    // Process full 16-byte chunks
    while (len >= 16) {
        stateA = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        stateB = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        stateC = _mm_set_epi64x(0x6c62272e07bb0142, 0x9e3779b97f4a7c15);
        _mm_prefetch(data + 512, _MM_HINT_T0);
        chunkA = _mm_loadu_si128((__m128i *) (data + i));
        stateA = _mm_aesenc_si128(stateA, chunkA);
        stateC = _mm_aesenc_si128(stateC, chunkA);
    }
}

```

```

stateB = _mm_aesenc_si128(stateB, chunkA);
stateC = _mm_aesenc_si128(stateC, chunkA);
ctx->stateMIX = _mm_aesenc_si128(ctx->stateMIX, stateA);
ctx->stateMIX = _mm_aesenc_si128(ctx->stateMIX, stateB);
ctx->stateMIX = _mm_aesenc_si128(ctx->stateMIX, stateC);
i += 16;
len -= 16;
}
ctx->total_bytes += i;

// Buffer remaining data
if (len > 0) {
    memcpy(ctx->buffer + ctx->buffer_len, data + i, len);
    ctx->buffer_len += len;
    ctx->total_bytes += len;
}

// Finalize streaming hash (32-bit output)
uint32_t pippip_final(PippipContext *ctx) {
    __m128i chunkA, chunkB;
    uint64_t hashLH, hashRH;

    // Process any remaining buffered data
    if (ctx->buffer_len > 0) {
        hashLH = _PADr_KAZE(*((uint64_t *)ctx->buffer), (16 - ctx->buffer_len) << 3);
        hashRH = _PAD_KAZE(*((uint64_t *)ctx->buffer), (16 - ctx->buffer_len) << 3);
        chunkA = _mm_set_epi64x(hashLH, hashRH);
        chunkB = _mm_set_epi64x(hashRH, hashRH);
        ctx->stateMIX = _mm_aesenc_si128(ctx->stateMIX, chunkA);
        ctx->stateMIX = _mm_aesenc_si128(ctx->stateMIX, chunkB);
        ctx->buffer_len = 0;
    }

    uint64_t result[2];
    _mm_storeu_si128((__m128i *)result, ctx->stateMIX);
    uint64_t hash64 = fold64(result[0], result[1]);
    return (uint32_t)hash64 ^ ctx->total_bytes;
}

// Original function (32-bit or 128-bit output based on eXdupe)
void FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla(const char *str, size_t wrdlen, uint32_t seed, void *output) {
    PippipContext ctx;
    pippip_init(&ctx, seed);
    pippip_update(&ctx, str, wrdlen);
    #ifdef eXdupe
        uint64_t result[2];
        _mm_storeu_si128((__m128i *)result, ctx.stateMIX);
        ((hashPippip128_t *)output)->low = result[0];
        ((hashPippip128_t *)output)->high = result[1];
    #else
        *(uint32_t *)output = pippip_final(&ctx);
    #endif
}

// 64-bit hash function
uint64_t pippip_hash64(const char *str, size_t len, uint32_t seed) {
    PippipContext ctx;
    pippip_init(&ctx, seed);
    pippip_update(&ctx, str, len);
    uint64_t result[2];
    _mm_storeu_si128((__m128i *)result, ctx.stateMIX);
    return fold64(result[0], result[1]) ^ ctx.total_bytes;
}

// Flexible-length hash function
void pippip_hash(const char *str, size_t len, uint32_t seed, void *output, size_t result_len) {
    PippipContext ctx;
    pippip_init(&ctx, seed);
    pippip_update(&ctx, str, len);
    uint64_t result[2];
    _mm_storeu_si128((__m128i *)result, ctx.stateMIX);
    uint64_t hash64 = fold64(result[0], result[1]) ^ ctx.total_bytes;

    if (result_len == 4) {
        *(uint32_t *)output = (uint32_t)hash64;
    } else if (result_len == 8) {
        *(uint64_t *)output = hash64;
    } else if (result_len == 16) {
        ((hashPippip128_t *)output)->low = result[0];
        ((hashPippip128_t *)output)->high = result[1];
    } else {
        // Unsupported length, default to 32-bit
        *(uint32_t *)output = (uint32_t)hash64;
    }
}

// Compute hash in streaming mode
int compute_streaming_hash(const char *filename, uint32_t *hash32, uint64_t *hash64, hashPippip128_t *hash128) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        fprintf(stderr, "Error: Cannot open file %s\n", filename);
        return -1;
    }

    PippipContext ctx;
    pippip_init(&ctx, 42); // Fixed seed for consistency
    char buffer[CHUNK_SIZE];

```

```

size_t bytes_read;

while ((bytes_read = fread(buffer, 1, CHUNK_SIZE, file)) > 0) {
    pippip_update(&ctx, buffer, bytes_read);
}

if (ferror(file)) {
    fprintf(stderr, "Error: Failed to read file %s\n", filename);
    fclose(file);
    return -1;
}

// Compute 32-bit hash
*hash32 = pippip_final(&ctx);

// Compute 64-bit and 128-bit hashes
uint64_t result[2];
_mm_storeu_si128((__m128i *)result, ctx.stateMIX);
*hash64 = fold64(result[0], result[1]) ^ ctx.total_bytes;
hash128->low = result[0];
hash128->high = result[1];

fclose(file);
return 0;
}

// Compute hash in one-go mode
int compute_one_go_hash(const char *filename, uint32_t *hash32, uint64_t *hash64, hashPippip128_t *hash128) {
FILE *file = fopen(filename, "rb");
if (!file) {
    fprintf(stderr, "Error: Cannot open file %s\n", filename);
    return -1;
}

// Get file size
if (fseek(file, 0, SEEK_END) != 0) {
    fprintf(stderr, "Error: Cannot seek to end of file %s\n", filename);
    fclose(file);
    return -1;
}
off_t file_size = ftell(file);
if (file_size < 0) {
    fprintf(stderr, "Error: Cannot determine size of file %s\n", filename);
    fclose(file);
    return -1;
}
rewind(file);

fprintf(stderr, "Debug: File size = %lld bytes\n", (long long)file_size);

// Allocate aligned memory for file
char *buffer = NULL;
if (posix_memalign((void **)&buffer, 16, file_size + 16) != 0) {
    fprintf(stderr, "Error: Aligned memory allocation failed for %lld bytes\n", (long long)file_size);
    fclose(file);
    return -1;
}

// Initialize buffer with zeros to avoid uninitialized memory access
memset(buffer, 0, file_size + 16);

// Read file
size_t bytes_read = fread(buffer, 1, file_size, file);
if (bytes_read != (size_t)file_size) {
    fprintf(stderr, "Error: Read %zu bytes, expected %lld bytes from file %s\n",
            bytes_read, (long long)file_size, filename);
    free(buffer);
    fclose(file);
    return -1;
}

fprintf(stderr, "Debug: Successfully read %zu bytes\n", bytes_read);

// Compute hashes using streaming API for consistency
PippipContext ctx;
pippip_init(&ctx, 42);
fprintf(stderr, "Debug: Successfully initialized context\n");
pippip_update(&ctx, buffer, file_size);
fprintf(stderr, "Debug: Successfully updated context\n");
*hash32 = pippip_final(&ctx);
fprintf(stderr, "Debug: Successfully finalized 32-bit hash\n");

// Compute 64-bit and 128-bit hashes
uint64_t result[2];
_mm_storeu_si128((__m128i *)result, ctx.stateMIX);
*hash64 = fold64(result[0], result[1]) ^ ctx.total_bytes;
hash128->low = result[0];
hash128->high = result[1];

free(buffer);
fclose(file);
return 0;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
}

```

```

}

uint32_t hash32_stream, hash32_one_go;
uint64_t hash64_stream, hash64_one_go;
hashPippip128_t hash128_stream, hash128_one_go;

// Streaming mode
printf("Streaming Mode (chunk size: %d bytes):\n", CHUNK_SIZE);
if (compute_streaming_hash(argv[1], &hash32_stream, &hash64_stream, &hash128_stream) == 0) {
    printf(" 32-bit hash: 0x%08x\n", hash32_stream);
    printf(" 64-bit hash: 0x%016llx\n", (unsigned long long)hash64_stream);
    printf(" 128-bit hash: 0x%016llx%016llx\n",
           (unsigned long long)hash128_stream.high, (unsigned long long)hash128_stream.low);
} else {
    return 1;
}

// In-one-go mode
printf("\nIn-One-Go Mode:\n");
if (compute_one_go_hash(argv[1], &hash32_one_go, &hash64_one_go, &hash128_one_go) == 0) {
    printf(" 32-bit hash: 0x%08x\n", hash32_one_go);
    printf(" 64-bit hash: 0x%016llx\n", (unsigned long long)hash64_one_go);
    printf(" 128-bit hash: 0x%016llx%016llx\n",
           (unsigned long long)hash128_one_go.high, (unsigned long long)hash128_one_go.low);
} else {
    return 1;
}

return 0;
}

3]
// Written by Grok, 2025-Jul-25
// gcc -o pippip_hash_arm pippip_hash_arm.c -march=armv8-a+aes -O2

#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include <arm_neon.h>

#define HashInBITS 32 // default 26 i.e. 2^26 i.e. 64MS(Mega Slots); slots contain 8bytes pointers or 512MB
#define MASK ((1ULL << HashInBITS) - 1)
#define _PADr_KAZE(x, n) (((x) << (n)) >> (n))
#define _PAD_KAZE(x, n) (((x) << (n)))

typedef struct {
    uint64_t low;
    uint64_t high;
} hashPippip128_t;

static inline uint64_t fold64(uint64_t A, uint64_t B) {
    #if defined(__GNUC__) || defined(__clang__)
        // Use 64-bit multiply and high multiply for 128-bit result
        uint64_t low = A * B; // Low 64 bits
        uint64_t high;
        // UMULH equivalent: get high 64 bits of 128-bit product
        #ifdef __aarch64__
            high = __builtin_aarch64_umulh(A, B);
        #else
            // Fallback for non-AArch64 or older compilers
            uint64_t a_low = A & 0xFFFFFFFF;
            uint64_t a_high = A >> 32;
            uint64_t b_low = B & 0xFFFFFFFF;
            uint64_t b_high = B >> 32;
            uint64_t mid = a_low * b_high + a_high * b_low;
            low = (a_low * b_low) + (mid << 32);
            high = (a_high * b_high) + (mid >> 32) + (low < (mid << 32));
        #endif
        return low ^ high;
    #else
        uint64_t hash64 = A ^ B;
        hash64 *= 1099511628211;
        return hash64;
    #endif
}

static inline uint32_t fold32(uint32_t A, uint32_t B) {
    #if defined(__GNUC__) || defined(__clang__)
        uint64_t r = (uint64_t)A * (uint64_t)B;
        return (uint32_t)r ^ (uint32_t)(r >> 32);
    #else
        uint32_t hash32 = A ^ B;
        hash32 *= 591798841;
        return hash32;
    #endif
}

void FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla_ARM(const char *str, size_t wrflen, uint32_t seed, void *output) {
    uint8x16_t chunkA;
    uint8x16_t chunkB;
    uint8x16_t stateMIX;
    uint64_t hashLH;
    uint64_t hashRH;

    // Initialize stateMIX with seed XOR wrflen
    stateMIX = vreinterpretq_u8_u32(vdupq_n_u32((uint32_t)wrflen ^ seed));
}

```

```

if (wrflen > 8) {
    uint8x16_t stateA = vreinterpretq_u8_u64(vdupq_n_u64(0x9e3779b97f4a7c15));
    stateA = vsetq_lane_u64(0x6c62272e07bb0142, vreinterpretq_u64_u8(stateA), 1);
    uint8x16_t stateB = vreinterpretq_u8_u64(vdupq_n_u64(0x9e3779b97f4a7c15));
    stateB = vsetq_lane_u64(0x6c62272e07bb0142, vreinterpretq_u64_u8(stateB), 1);
    uint8x16_t stateC = vreinterpretq_u8_u64(vdupq_n_u64(0x9e3779b97f4a7c15));
    stateC = vsetq_lane_u64(0x6c62272e07bb0142, vreinterpretq_u64_u8(stateC), 1);

    size_t Cycles, NDhead;
    if (wrflen > 16) {
        Cycles = ((wrflen - 1) >> 5) + 1;
        NDhead = wrflen - (Cycles << 4);
        if (NDhead < 0 || NDhead + 16 > wrflen) {
            // Fallback to single-chunk processing for safety
            Cycles = wrflen >> 4; // Process 16-byte chunks
            for (; Cycles--; str += 16) {
                chunkA = vld1q_u8((const uint8_t *)str);
                stateA = vaeseq_u8(stateA, chunkA);
                stateC = vaeseq_u8(stateC, chunkA);
                stateB = vaeseq_u8(stateB, chunkA);
                stateC = vaeseq_u8(stateC, chunkA);
            }
        } else if (Cycles & 1) {
            #pragma nounroll
            for (; Cycles--; str += 16) {
                chunkA = vld1q_u8((const uint8_t *)str);
                stateA = vaeseq_u8(stateA, chunkA);
                chunkB = vld1q_u8((const uint8_t *)str + NDhead);
                stateC = vaeseq_u8(stateC, chunkB);
                stateB = vaeseq_u8(stateB, chunkB);
                stateC = vaeseq_u8(stateC, chunkB);
            }
        } else {
            Cycles = Cycles >> 1;
            uint8x16_t stateA2 = vreinterpretq_u8_u64(vdupq_n_u64(0x9e3779b97f4a7c15));
            stateA2 = vsetq_lane_u64(0x6c62272e07bb0142, vreinterpretq_u64_u8(stateA2), 1);
            uint8x16_t stateB2 = vreinterpretq_u8_u64(vdupq_n_u64(0x9e3779b97f4a7c15));
            stateB2 = vsetq_lane_u64(0x6c62272e07bb0142, vreinterpretq_u64_u8(stateB2), 1);
            uint8x16_t stateC2 = vreinterpretq_u8_u64(vdupq_n_u64(0x9e3779b97f4a7c15));
            stateC2 = vsetq_lane_u64(0x6c62272e07bb0142, vreinterpretq_u64_u8(stateC2), 1);
            #pragma nounroll
            for (; Cycles--; str += 32) {
                chunkA = vld1q_u8((const uint8_t *)str);
                uint8x16_t chunkA2 = vld1q_u8((const uint8_t *)str + 16);
                stateA = vaeseq_u8(stateA, chunkA);
                stateA2 = vaeseq_u8(stateA2, chunkA2);
                chunkB = vld1q_u8((const uint8_t *)str + NDhead);
                uint8x16_t chunkB2 = vld1q_u8((const uint8_t *)str + NDhead + 16);
                stateC = vaeseq_u8(stateC, chunkA);
                stateB = vaeseq_u8(stateB, chunkB);
                stateC2 = vaeseq_u8(stateC2, chunkB2);
                stateC = vaeseq_u8(stateC, chunkB);
                stateC2 = vaeseq_u8(stateC2, chunkA2);
                stateB2 = vaeseq_u8(stateB2, chunkB2);
                stateC2 = vaeseq_u8(stateC2, chunkB2);
                stateMIX = vaeseq_u8(stateMIX, stateA2);
                stateMIX = vaeseq_u8(stateMIX, stateB2);
                stateMIX = vaeseq_u8(stateMIX, stateC2);
            }
        }
    } else { // 9..16 bytes
        NDhead = wrflen - (1 << 3);
        hashLH = (*(uint64_t *)str);
        hashRH = (*(uint64_t *)str + NDhead);
        chunkA = vreinterpretq_u8_u64(vdupq_n_u64(hashLH));
        stateA = vaeseq_u8(stateA, chunkA);
        chunkB = vreinterpretq_u8_u64(vdupq_n_u64(hashRH));
        stateC = vaeseq_u8(stateC, chunkA);
        stateB = vaeseq_u8(stateB, chunkB);
        stateC = vaeseq_u8(stateC, chunkB);
    }
    stateMIX = vaeseq_u8(stateMIX, stateA);
    stateMIX = vaeseq_u8(stateMIX, stateB);
    stateMIX = vaeseq_u8(stateMIX, stateC);
} else {
    hashLH = _PADr_KAZE(*(uint64_t *)(str + 0), (8 - wrflen) << 3);
    hashRH = _PAD_KAZE(*(uint64_t *)(str + 0), (8 - wrflen) << 3);
    chunkA = vreinterpretq_u8_u64(vdupq_n_u64(hashLH));
    chunkB = vreinterpretq_u8_u64(vdupq_n_u64(hashRH));
    stateMIX = vaeseq_u8(stateMIX, chunkA);
    stateMIX = vaeseq_u8(stateMIX, chunkB);
}

#endif eXdupe
    vst1q_u8((uint8_t *)output, stateMIX); // Store full 128-bit state
#else
    uint64_t result[2];
    vst1q_u8((uint8_t *)result, stateMIX);
    uint64_t hash64 = fold64(result[0], result[1]);
    *(uint32_t *)output = (uint32_t)hash64 ^ wrflen;
#endif
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <string>\n", argv[0]);
        return 1;
    }

    FNV1A_Pippin_128bit_AES_TriXZi_Mikayla (FNV1A-Deathship-Dualheaded a.k.a. FNV1A-Totenköpfe a.k.a. FNV1A-Totenschiff), Update: 2025-Aug-19, Kaze(sanmayce@sanmayce.com), 37/46
}

```

```

const char *input = argv[1];
size_t len = strlen(input);
uint32_t seed = 42;

// Test 32-bit hash
uint32_t hash32;
FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla_ARM(input, len, seed, &hash32);
printf("32-bit hash: 0x%08x\n", hash32);

// Test 128-bit hash (eXdupe mode)
#ifndef eXdupe
    hashPippip128_t hash128;
    FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla_ARM(input, len, seed, &hash128);
    printf("128-bit hash: 0x%016llx%016llx\n",
        (unsigned long long)hash128.high, (unsigned long long)hash128.low);
#endif

return 0;
}

```

4] Using the original SMHASHER gives this:

```
[sanmayce@leprechaun2 src]$ cmake . -DCMAKE_CXX_FLAGS="-maes -mbmi2"
CMake Warning (dev) at CMakeLists.txt:1 (project):
cmake_minimum_required() should be called prior to this top-level project()
call. Please see the cmake-commands(7) manual for usage documentation of
both commands.
```

This warning is for project developers. Use -Wno-dev to suppress it.

```
-- The C compiler identification is GNU 14.2.1
-- The CXX compiler identification is GNU 14.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
CMake Deprecation Warning at CMakeLists.txt:3 (cmake_minimum_required):
Compatibility with CMake < 3.5 will be removed from a future version of
CMake.
```

Update the VERSION argument <min> value or use a ...<max> suffix to tell CMake that the project does not need compatibility with older versions.

```
-- Configuring done (0.6s)
-- Generating done (0.0s)
-- Build files have been written to: /home/sanmayce/_KAZE/Benchmark_Sefaria_(eXdupe_xx_vs_eXdupe_Pippip)_newsh/smhasher-master_Pippip/src
[sanmayce@leprechaun2 src]$ make
[ 3%] Building CXX object CMakeFiles/SMHasherSupport.dir/AvalancheTest.o
[ 7%] Building CXX object CMakeFiles/SMHasherSupport.dir/BitSlice.o
[11%] Building CXX object CMakeFiles/SMHasherSupport.dir/Bitvec.o
[14%] Building CXX object CMakeFiles/SMHasherSupport.dir/CityTest.o
[18%] Building CXX object CMakeFiles/SMHasherSupport.dir/City.o
[22%] Building CXX object CMakeFiles/SMHasherSupport.dir/crc.o
[25%] Building CXX object CMakeFiles/SMHasherSupport.dir/DifferentialTest.o
[29%] Building CXX object CMakeFiles/SMHasherSupport.dir/Hashes.o
[33%] Building CXX object CMakeFiles/SMHasherSupport.dir/KeysetTest.o
[37%] Building CXX object CMakeFiles/SMHasherSupport.dir/lookup3.o
[40%] Building CXX object CMakeFiles/SMHasherSupport.dir/md5.o
[44%] Building CXX object CMakeFiles/SMHasherSupport.dir/MurmurHash1.o
[48%] Building CXX object CMakeFiles/SMHasherSupport.dir/MurmurHash2.o
[51%] Building CXX object CMakeFiles/SMHasherSupport.dir/MurmurHash3.o
[55%] Building CXX object CMakeFiles/SMHasherSupport.dir/Platform.o
[59%] Building CXX object CMakeFiles/SMHasherSupport.dir/Random.o
[62%] Building CXX object CMakeFiles/SMHasherSupport.dir/sha1.o
[66%] Building CXX object CMakeFiles/SMHasherSupport.dir/SpeedTest.o
/home/sanmayce/_KAZE/Benchmark_Sefaria_(eXdupe_xx_vs_eXdupe_Pippip)_newsh/smhasher-master_Pippip/src/SpeedTest.cpp: In function 'int64_t
timehash(pfHash, const void*, int, int)':
/home/sanmayce/_KAZE/Benchmark_Sefaria_(eXdupe_xx_vs_eXdupe_Pippip)_newsh/smhasher-master_Pippip/src/SpeedTest.cpp:153:29: warning: ISO C++17
does not allow 'register' storage class specifier [-Wregister]
153 |     volatile register int64_t begin,end;
|           ^
/home/sanmayce/_KAZE/Benchmark_Sefaria_(eXdupe_xx_vs_eXdupe_Pippip)_newsh/smhasher-master_Pippip/src/SpeedTest.cpp:153:35: warning: ISO C++17
does not allow 'register' storage class specifier [-Wregister]
153 |     volatile register int64_t begin,end;
|           ^
```

```
[ 70%] Building CXX object CMakeFiles/SMHasherSupport.dir/Spooky.o
[ 74%] Building CXX object CMakeFiles/SMHasherSupport.dir/SpookyTest.o
[ 77%] Building CXX object CMakeFiles/SMHasherSupport.dir/Stats.o
[ 81%] Building CXX object CMakeFiles/SMHasherSupport.dir/SuperFastHash.o
[ 85%] Building CXX object CMakeFiles/SMHasherSupport.dir/Types.o
[ 88%] Building C object CMakeFiles/SMHasherSupport.dir/PMurHash.o
[ 92%] Linking CXX static library libSMHasherSupport.a
[ 92%] Built target SMHasherSupport
[ 96%] Building CXX object CMakeFiles/SMHasher.dir/main.o
[100%] Linking CXX executable SMHasher
[100%] Built target SMHasher
[sanmayce@leprechaun2 src]$ ./SMHasher FNV
```

--- Testing FNV (Fowler-Noll-Vo hash, 32-bit)

[[[Sanity Tests]]]

Verification value 0xDEE2D53C : Passed!
Running sanity check 1.....PASS
Running sanity check 2.....PASS

[[[Speed Tests]]]

Bulk speed test - 262144-byte keys
Alignment 0 - 7.962 bytes/cycle - 22779.34 MiB/sec @ 3 ghz
Alignment 1 - 7.953 bytes/cycle - 22753.36 MiB/sec @ 3 ghz
Alignment 2 - 7.953 bytes/cycle - 22753.68 MiB/sec @ 3 ghz
Alignment 3 - 7.953 bytes/cycle - 22753.23 MiB/sec @ 3 ghz
Alignment 4 - 7.953 bytes/cycle - 22754.36 MiB/sec @ 3 ghz
Alignment 5 - 7.953 bytes/cycle - 22754.73 MiB/sec @ 3 ghz
Alignment 6 - 7.953 bytes/cycle - 22754.27 MiB/sec @ 3 ghz
Alignment 7 - 7.953 bytes/cycle - 22753.06 MiB/sec @ 3 ghz

Small key speed test - 1-byte keys - 34.14 cycles/hash
Small key speed test - 2-byte keys - 34.35 cycles/hash
Small key speed test - 3-byte keys - 35.80 cycles/hash
Small key speed test - 4-byte keys - 32.58 cycles/hash
Small key speed test - 5-byte keys - 32.98 cycles/hash
Small key speed test - 6-byte keys - 36.07 cycles/hash
Small key speed test - 7-byte keys - 37.20 cycles/hash
Small key speed test - 8-byte keys - 35.36 cycles/hash
Small key speed test - 9-byte keys - 35.92 cycles/hash
Small key speed test - 10-byte keys - 36.37 cycles/hash
Small key speed test - 11-byte keys - 37.59 cycles/hash
Small key speed test - 12-byte keys - 36.57 cycles/hash
Small key speed test - 13-byte keys - 38.80 cycles/hash
Small key speed test - 14-byte keys - 38.73 cycles/hash
Small key speed test - 15-byte keys - 38.00 cycles/hash
Small key speed test - 16-byte keys - 38.91 cycles/hash
Small key speed test - 17-byte keys - 39.40 cycles/hash
Small key speed test - 18-byte keys - 38.53 cycles/hash
Small key speed test - 19-byte keys - 38.84 cycles/hash
Small key speed test - 20-byte keys - 38.24 cycles/hash
Small key speed test - 21-byte keys - 40.52 cycles/hash
Small key speed test - 22-byte keys - 37.58 cycles/hash
Small key speed test - 23-byte keys - 38.29 cycles/hash
Small key speed test - 24-byte keys - 39.34 cycles/hash
Small key speed test - 25-byte keys - 38.95 cycles/hash
Small key speed test - 26-byte keys - 38.48 cycles/hash
Small key speed test - 27-byte keys - 37.74 cycles/hash
Small key speed test - 28-byte keys - 38.48 cycles/hash
Small key speed test - 29-byte keys - 38.80 cycles/hash
Small key speed test - 30-byte keys - 38.55 cycles/hash
Small key speed test - 31-byte keys - 37.62 cycles/hash

[[[Differential Tests]]]

Testing 8303632 up-to-5-bit differentials in 64-bit keys -> 32 bit hashes.
1000 reps, 8303632000 total tests, expecting 1.93 random collisions.....
2 total collisions, of which 2 single collisions were ignored

Testing 11017632 up-to-4-bit differentials in 128-bit keys -> 32 bit hashes.
1000 reps, 11017632000 total tests, expecting 2.57 random collisions.....
3 total collisions, of which 3 single collisions were ignored

Testing 2796416 up-to-3-bit differentials in 256-bit keys -> 32 bit hashes.
1000 reps, 2796416000 total tests, expecting 0.65 random collisions.....
0 total collisions, of which 0 single collisions were ignored

[[[Avalanche Tests]]]

Testing 32-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.564667%
Testing 40-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.639333%
Testing 48-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.760667%
Testing 56-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.577333%
Testing 64-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.714000%
Testing 72-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.621333%
Testing 80-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.662000%
Testing 88-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.766667%
Testing 96-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.654000%
Testing 104-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.726000%
Testing 112-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.765333%
Testing 120-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.646000%
Testing 128-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.868000%
Testing 136-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.658667%
Testing 144-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.613333%
Testing 152-bit keys -> 32-bit hashes, 300000 reps..... worst bias is 0.724667%

[[[Keyset 'Cyclic' Tests]]]

Keyset 'Cyclic' - 8 cycles of 4 bytes - 10000000 keys
Testing collisions - Expected 11641.53, actual 11529.00 (0.99x)
Testing distribution - Worst bias is the 20-bit window at bit 24 - 0.035%

Keyset 'Cyclic' - 8 cycles of 5 bytes - 10000000 keys
Testing collisions - Expected 11641.53, actual 11618.00 (1.00x)
Testing distribution - Worst bias is the 20-bit window at bit 5 - 0.044%

Keyset 'Cyclic' - 8 cycles of 6 bytes - 10000000 keys
Testing collisions - Expected 11641.53, actual 11441.00 (0.98x)
Testing distribution - Worst bias is the 20-bit window at bit 17 - 0.022%

Keyset 'Cyclic' - 8 cycles of 7 bytes - 10000000 keys
Testing collisions - Expected 11641.53, actual 11330.00 (0.97x)
Testing distribution - Worst bias is the 20-bit window at bit 3 - 0.032%

Keyset 'Cyclic' - 8 cycles of 8 bytes - 10000000 keys
Testing collisions - Expected 11641.53, actual 11671.00 (1.00x)
Testing distribution - Worst bias is the 20-bit window at bit 29 - 0.014%

[[[Keyset 'TwoBytes' Tests]]]

Keyset 'TwoBytes' - up-to-4-byte keys, 652545 total keys
Testing collisions - Expected 49.57, actual 44.00 (0.89x)
Testing distribution - Worst bias is the 16-bit window at bit 7 - 0.106%

Keyset 'TwoBytes' - up-to-8-byte keys, 5471025 total keys
Testing collisions - Expected 3484.56, actual 3782.00 (1.09x)
Testing distribution - Worst bias is the 20-bit window at bit 7 - 0.139%

Keyset 'TwoBytes' - up-to-12-byte keys, 18616785 total keys
Testing collisions - Expected 40347.77, actual 40448.00 (1.00x)
Testing distribution - Worst bias is the 20-bit window at bit 5 - 0.019%

Keyset 'TwoBytes' - up-to-16-byte keys, 44251425 total keys
Testing collisions - Expected 227963.15, actual 227565.00 (1.00x)
Testing distribution - Worst bias is the 20-bit window at bit 29 - 0.005%

Keyset 'TwoBytes' - up-to-20-byte keys, 86536545 total keys
Testing collisions - Expected 871784.70, actual 866762.00 (0.99x)
Testing distribution - Worst bias is the 20-bit window at bit 7 - 0.004%

[[[Keyset 'Sparse' Tests]]]

Keyset 'Sparse' - 32-bit keys with up to 6 bits set - 1149017 keys
Testing collisions - Expected 153.70, actual 159.00 (1.03x)
Testing distribution - Worst bias is the 17-bit window at bit 31 - 0.129%

Keyset 'Sparse' - 40-bit keys with up to 6 bits set - 4598479 keys
Testing collisions - Expected 2461.72, actual 2530.00 (1.03x)
Testing distribution - Worst bias is the 19-bit window at bit 0 - 0.052%

Keyset 'Sparse' - 48-bit keys with up to 5 bits set - 1925357 keys
Testing collisions - Expected 431.55, actual 429.00 (0.99x)
Testing distribution - Worst bias is the 18-bit window at bit 0 - 0.060%

Keyset 'Sparse' - 56-bit keys with up to 5 bits set - 4216423 keys
Testing collisions - Expected 2069.66, actual 2245.00 (1.08x)
Testing distribution - Worst bias is the 19-bit window at bit 27 - 0.102%

Keyset 'Sparse' - 64-bit keys with up to 5 bits set - 8303633 keys
Testing collisions - Expected 8026.87, actual 8685.00 (1.08x)
Testing distribution - Worst bias is the 20-bit window at bit 1 - 0.091%

Keyset 'Sparse' - 96-bit keys with up to 4 bits set - 3469497 keys
Testing collisions - Expected 1401.34, actual 1379.00 (0.98x)
Testing distribution - Worst bias is the 19-bit window at bit 12 - 0.049%

Keyset 'Sparse' - 256-bit keys with up to 3 bits set - 2796417 keys
Testing collisions - Expected 910.36, actual 944.00 (1.04x)
Testing distribution - Worst bias is the 19-bit window at bit 21 - 0.067%

Keyset 'Sparse' - 2048-bit keys with up to 2 bits set - 2098177 keys
Testing collisions - Expected 512.50, actual 455.00 (0.89x)
Testing distribution - Worst bias is the 18-bit window at bit 0 - 0.067%

[[[Keyset 'Combination Lowbits' Tests]]]

Keyset 'Combination' - up to 8 blocks from a set of 8 - 19173960 keys
Testing collisions - Expected 42799.01, actual 42823.00 (1.00x)
Testing distribution - Worst bias is the 20-bit window at bit 31 - 0.012%

[[[Keyset 'Combination Highbits' Tests]]]

Keyset 'Combination' - up to 8 blocks from a set of 8 - 19173960 keys
Testing collisions - Expected 42799.01, actual 42747.00 (1.00x)
Testing distribution - Worst bias is the 20-bit window at bit 16 - 0.014%

[[[Keyset 'Combination 0x8000000' Tests]]]

Keyset 'Combination' - up to 20 blocks from a set of 2 - 2097150 keys
Testing collisions - Expected 512.00, actual 514.00 (1.00x)
Testing distribution - Worst bias is the 18-bit window at bit 25 - 0.073%

[[[Keyset 'Combination 0x0000001' Tests]]]

Keyset 'Combination' - up to 20 blocks from a set of 2 - 2097150 keys
Testing collisions - Expected 512.00, actual 554.00 (1.08x)
Testing distribution - Worst bias is the 18-bit window at bit 25 - 0.081%

[[[Keyset 'Combination Hi-Lo' Tests]]]

Keyset 'Combination' - up to 6 blocks from a set of 15 - 12204240 keys
Testing collisions - Expected 17339.30, actual 17455.00 (1.01x)
Testing distribution - Worst bias is the 20-bit window at bit 16 - 0.026%

[[[Keyset 'Window' Tests]]]

Keyset 'Windowed' - 64-bit key, 20-bit window - 64 tests, 1048576 keys per test
Window at 0 - Testing collisions - Expected 128.00, actual 115.00 (0.90x)
Window at 1 - Testing collisions - Expected 128.00, actual 118.00 (0.92x)
Window at 2 - Testing collisions - Expected 128.00, actual 118.00 (0.92x)
Window at 3 - Testing collisions - Expected 128.00, actual 123.00 (0.96x)

```

Window at  4 - Testing collisions - Expected 128.00, actual 128.00 ( 1.00x)
Window at  5 - Testing collisions - Expected 128.00, actual 119.00 ( 0.93x)
Window at  6 - Testing collisions - Expected 128.00, actual 108.00 ( 0.84x)
Window at  7 - Testing collisions - Expected 128.00, actual 108.00 ( 0.84x)
Window at  8 - Testing collisions - Expected 128.00, actual 120.00 ( 0.94x)
Window at  9 - Testing collisions - Expected 128.00, actual 117.00 ( 0.91x)
Window at 10 - Testing collisions - Expected 128.00, actual 136.00 ( 1.06x)
Window at 11 - Testing collisions - Expected 128.00, actual 139.00 ( 1.09x)
Window at 12 - Testing collisions - Expected 128.00, actual 137.00 ( 1.07x)
Window at 13 - Testing collisions - Expected 128.00, actual 128.00 ( 1.00x)
Window at 14 - Testing collisions - Expected 128.00, actual 150.00 ( 1.17x)
Window at 15 - Testing collisions - Expected 128.00, actual 133.00 ( 1.04x)
Window at 16 - Testing collisions - Expected 128.00, actual 143.00 ( 1.12x)
Window at 17 - Testing collisions - Expected 128.00, actual 146.00 ( 1.14x)
Window at 18 - Testing collisions - Expected 128.00, actual 124.00 ( 0.97x)
Window at 19 - Testing collisions - Expected 128.00, actual 123.00 ( 0.96x)
Window at 20 - Testing collisions - Expected 128.00, actual 107.00 ( 0.84x)
Window at 21 - Testing collisions - Expected 128.00, actual 124.00 ( 0.97x)
Window at 22 - Testing collisions - Expected 128.00, actual 111.00 ( 0.87x)
Window at 23 - Testing collisions - Expected 128.00, actual 114.00 ( 0.89x)
Window at 24 - Testing collisions - Expected 128.00, actual 133.00 ( 1.04x)
Window at 25 - Testing collisions - Expected 128.00, actual 129.00 ( 1.01x)
Window at 26 - Testing collisions - Expected 128.00, actual 124.00 ( 0.97x)
Window at 27 - Testing collisions - Expected 128.00, actual 152.00 ( 1.19x)
Window at 28 - Testing collisions - Expected 128.00, actual 141.00 ( 1.10x)
Window at 29 - Testing collisions - Expected 128.00, actual 136.00 ( 1.06x)
Window at 30 - Testing collisions - Expected 128.00, actual 154.00 ( 1.20x)
Window at 31 - Testing collisions - Expected 128.00, actual 124.00 ( 0.97x)
Window at 32 - Testing collisions - Expected 128.00, actual 123.00 ( 0.96x)
Window at 33 - Testing collisions - Expected 128.00, actual 145.00 ( 1.13x)
Window at 34 - Testing collisions - Expected 128.00, actual 136.00 ( 1.06x)
Window at 35 - Testing collisions - Expected 128.00, actual 129.00 ( 1.01x)
Window at 36 - Testing collisions - Expected 128.00, actual 128.00 ( 1.00x)
Window at 37 - Testing collisions - Expected 128.00, actual 131.00 ( 1.02x)
Window at 38 - Testing collisions - Expected 128.00, actual 136.00 ( 1.06x)
Window at 39 - Testing collisions - Expected 128.00, actual 141.00 ( 1.10x)
Window at 40 - Testing collisions - Expected 128.00, actual 144.00 ( 1.13x)
Window at 41 - Testing collisions - Expected 128.00, actual 143.00 ( 1.12x)
Window at 42 - Testing collisions - Expected 128.00, actual 133.00 ( 1.04x)
Window at 43 - Testing collisions - Expected 128.00, actual 132.00 ( 1.03x)
Window at 44 - Testing collisions - Expected 128.00, actual 124.00 ( 0.97x)
Window at 45 - Testing collisions - Expected 128.00, actual 130.00 ( 1.02x)
Window at 46 - Testing collisions - Expected 128.00, actual 113.00 ( 0.88x)
Window at 47 - Testing collisions - Expected 128.00, actual 109.00 ( 0.85x)
Window at 48 - Testing collisions - Expected 128.00, actual 135.00 ( 1.05x)
Window at 49 - Testing collisions - Expected 128.00, actual 130.00 ( 1.02x)
Window at 50 - Testing collisions - Expected 128.00, actual 140.00 ( 1.09x)
Window at 51 - Testing collisions - Expected 128.00, actual 126.00 ( 0.98x)
Window at 52 - Testing collisions - Expected 128.00, actual 117.00 ( 0.91x)
Window at 53 - Testing collisions - Expected 128.00, actual 130.00 ( 1.02x)
Window at 54 - Testing collisions - Expected 128.00, actual 143.00 ( 1.12x)
Window at 55 - Testing collisions - Expected 128.00, actual 135.00 ( 1.05x)
Window at 56 - Testing collisions - Expected 128.00, actual 140.00 ( 1.09x)
Window at 57 - Testing collisions - Expected 128.00, actual 117.00 ( 0.91x)
Window at 58 - Testing collisions - Expected 128.00, actual 122.00 ( 0.95x)
Window at 59 - Testing collisions - Expected 128.00, actual 127.00 ( 0.99x)
Window at 60 - Testing collisions - Expected 128.00, actual 144.00 ( 1.13x)
Window at 61 - Testing collisions - Expected 128.00, actual 154.00 ( 1.20x)
Window at 62 - Testing collisions - Expected 128.00, actual 133.00 ( 1.04x)
Window at 63 - Testing collisions - Expected 128.00, actual 127.00 ( 0.99x)
Window at 64 - Testing collisions - Expected 128.00, actual 115.00 ( 0.90x)

```

[[[Keyset 'Text' Tests]]]

```

Keyset 'Text' - keys of form "Foo[XXXX]Bar" - 14776336 keys
Testing collisions - Expected 25418.13, actual 25173.00 ( 0.99x)
Testing distribution - Worst bias is the 20-bit window at bit 6 - 0.024%

```

```

Keyset 'Text' - keys of form "FooBar[XXXX]" - 14776336 keys
Testing collisions - Expected 25418.13, actual 25359.00 ( 1.00x)
Testing distribution - Worst bias is the 20-bit window at bit 4 - 0.019%

```

```

Keyset 'Text' - keys of form "[XXXX]FooBar" - 14776336 keys
Testing collisions - Expected 25418.13, actual 25387.00 ( 1.00x)
Testing distribution - Worst bias is the 20-bit window at bit 17 - 0.017%

```

[[[Keyset 'Zeroes' Tests]]]

```

Keyset 'Zeroes' - 65536 keys
Testing collisions - Expected 0.50, actual 0.00 ( 0.00x)
Testing distribution - Worst bias is the 12-bit window at bit 25 - 0.135%

```

[[[Keyset 'Seed' Tests]]]

```

Keyset 'Seed' - 1000000 keys
Testing collisions - Expected 116.42, actual 118.00 ( 1.01x)
Testing distribution - Worst bias is the 17-bit window at bit 25 - 0.112%

```

```

Input vcode 0x9f50b392, Output vcode 0xb426a819, Result vcode 0x00000001
Verification value is 0x00000001 - Testing took 1589.963537 seconds

```

```
[sammayce@leprechaun2 src]$
```

To reproduce, the lazy insertion of Pippip in place of FNV:
https://github.com/Sammayce/DoubleDeuce/releases/download/r7/smhasher-master_Pippip.tar.gz

5]
Benchmark_Sefaria_.eXdupe_xx_vs_eXdupe_Pippip.tar.gz 21.2 MB 2025-07-14T16:51:47Z

FastestHashBooklet_may15.pdf 6.16 MB 2025-05-19T04:31:57Z
 hashBBS_ver6.tar.gz 149 KB 2025-07-11T06:12:28Z
 hashBBS_ver7.tar.gz 212 KB 2025-07-18T10:09:18Z
 Latin_Wordlists_Extravaganza.tar.gz 2.52 MB 2025-05-15T02:00:06Z
 Pippip.h 13.2 KB 2025-07-18T10:09:28Z
 Pippip_AES_TriXZi.h 24.5 KB 2025-05-19T04:32:03Z
 Pippip_AES_TriXZi.h 26 KB 2025-07-11T06:06:50Z
 smasher-master_Pippip.tar.gz 280 KB 2025-07-22T13:00:55Z

6]
<https://images-rg.pixai.art/images/orig/59100b47-221a-4afa-9b11-cddec5f89c9>

7]
Note1: Testmachine: Latitude 7370 2x4 GB (Rank: 2) LPDDR3 1867 MT/s, Intel m7-6Y75 "Skylake", TDP: 4.5 W (MAX 7 W), 14 nm, Max Turbo: 3,100 MHz

L1d: 64 KiB (2 instances)
 L1i: 64 KiB (2 instances)
 L2: 512 KiB (2 instances)
 L3: 4 MiB (1 instance)

Note2: Compiler used: GCC 14.2.1 (-O3 -mavx2 -maes -mbmi2)

Note3: The user was superuser, CPU was set to performance mode, niceness was maximum: "nice -n -20"

Note4: The generated GCC assembler for Pippip:

FNV1A_Pippip_Yurii_000_128bit_AES_TriXZi_Mikayla:

```

.LFB6794:
    .cfi_startproc
    xorl %esi, %edx
    movq %rdi, %rax
    vnmovd%edx, %xmm6
    vpbroadcastd %xmm6, %xmm0
    vnmovdqa %xmm0, %xmm1
    cmpq $8, %rsi
    jbe .L100
    cmpq $16, %rsi
    ja .L109
    vnmovdqa .LC8(%rip), %xmm3
    vnmovddup (%rdi), %xmm4
    vnmovddup -8(%rdi,%rsi), %xmm2
    vaesenc %xmm4, %xmm3, %xmm4
    vaesenc %xmm2, %xmm3, %xmm3
    vaesenc %xmm2, %xmm4, %xmm2
.L104:
    vaesenc %xmm4, %xmm1, %xmm1
    vaesenc %xmm3, %xmm1, %xmm1
    vaesenc %xmm2, %xmm1, %xmm0
    jmp .L106
    .p2align 4,,10
    .p2align 3
.L100:
    movl $8, %edx
    movq (%rdi), %rax
    subl %esi, %edx
    sall $3, %edx
    shlx %rdx, %rax, %rax
    shrx %rdx, %rax, %rdx
    vnmovq%rdx, %xmm7
    vnmovq%rax, %xmm3
    vpunpcklqdq %xmm7, %xmm7, %xmm1
    vaesenc %xmm1, %xmm0, %xmm0
    vpunpcklqdq %xmm3, %xmm3, %xmm1
    vaesenc %xmm1, %xmm0, %xmm0
.L106:
    vnmovq%xmm0, %rdx
    vpxtrq $1, %xmm0, %rax
    mulx %rax, %rax, %rdx
    xorq %rdx, %rax
    xorl %eax, %esi
    movl %esi, (%rcx)
    ret
    .p2align 4,,10
    .p2align 3
.L109:
    leaq -1(%rsi), %rdi
    movq %rsi, %r8
    shrq $5, %rdi
    addq $1, %rdi
    movq %rdi, %rdx
    salq $4, %rdx
    subq %rdx, %r8
    testb$1, %dil
    je .L102
    vnmovdqa .LC8(%rip), %xmm2
    addq %rax, %rdx
    vnmovdqa %xmm2, %xmm3
    vnmovdqa %xmm2, %xmm4
    .p2align 6
    .p2align 4
    .p2align 3
.L103:
    vnmovdqu (%rax), %xmm8
    vnmovdqu (%rax,%r8), %xmm5
    addq $16, %rax
    vaesenc %xmm8, %xmm4, %xmm4
    vaesenc %xmm0, %xmm2, %xmm0
    vaesenc %xmm5, %xmm3, %xmm3
    vaesenc %xmm5, %xmm0, %xmm2
    cmpq %rdx, %rax
    jne .L103

```

```

jmp .L104
.p2align 4,,10
.p2align 3
.L102:
    vmovdqa    .LC8(%rip), %xmm1
    shrq %rdi
    leaq 512(%rax), %rdx
    leaq 512(%rax,%r8), %rax
    salq $5, %rdi
    vmovdqa    %xmm1, %xmm9
    vmovdqa    %xmm1, %xmm10
    vmovdqa    %xmm1, %xmm2
    addq %rdx, %rdi
    vmovdqa    %xmm1, %xmm3
    vmovdqa    %xmm1, %xmm4
    .p2align 4
    .p2align 3
.L105:
    vmovdqu    -512(%rdx), %xmm5
    prefetcht0 (%rdx)
    addq $32, %rdx
    vmovdqu    -528(%rdx), %xmm7
    vmovdqu    -512(%rax), %xmm8
    vmovdqu    -496(%rax), %xmm6
    prefetcht0 (%rax)
    addq $32, %rax
    vaesenc    %xmm5, %xmm4, %xmm4
    vaesenc    %xmm7, %xmm1, %xmm1
    vaesenc    %xmm5, %xmm2, %xmm5
    vaesenc    %xmm7, %xmm10, %xmm10
    vaesenc    %xmm8, %xmm3, %xmm3
    vaesenc    %xmm8, %xmm5, %xmm2
    vaesenc    %xmm6, %xmm9, %xmm9
    vaesenc    %xmm6, %xmm1, %xmm1
    cmpq %rdi, %rdx
    jne .L105
    vaesenc    %xmm10, %xmm0, %xmm0
    vaesenc    %xmm9, %xmm0, %xmm0
    vaesenc    %xmm1, %xmm0, %xmm1
    jmp .L104
.cfi_endproc
...

```

Note5: Hashing 'Fedora-Workstation-Live-42-1.1.x86_64.iso' (2,398,523,392 bytes) at each position for orders 2..8192, since the hashing takes few hours, the Turbo Modes are quite irrelevant as long as the cooling is not excellent i.e. they cannot be sustained for thus long making the benchmark even more non-synthetic. What makes it truly non-synthetic is the immediate lookoping (into 512MB hashtable housing 2^{32} slots) after hashing i.e. hashing+lookoping delivers real stats. The ASCII roster (UniqueHashes vs HashSlots = 4294967296/1694706087= 2.5 Slots-Per-Hash):

Hasher	BB_Size	Collisions	Speed
MeowHash	2	2,398,457,855	0.033 GiB/s
MeowHash	3	2,381,779,035	0.041 GiB/s
MeowHash	4	1,045,894,389	0.054 GiB/s
MeowHash	5	772,914,994	0.068 GiB/s
MeowHash	6	770,389,494	0.081 GiB/s
MeowHash	7	769,232,476	0.095 GiB/s
MeowHash	8	768,110,713	0.108 GiB/s
MeowHash	9	767,118,627	0.121 GiB/s
MeowHash	10	766,291,293	0.135 GiB/s
MeowHash	11	765,541,660	0.148 GiB/s
MeowHash	12	764,857,298	0.162 GiB/s
MeowHash	13	764,173,915	0.175 GiB/s
MeowHash	14	763,560,894	0.188 GiB/s
MeowHash	15	763,024,555	0.202 GiB/s
MeowHash	16	762,529,548	0.217 GiB/s
MeowHash	18	761,592,246	0.240 GiB/s
MeowHash	20	760,685,816	0.267 GiB/s
MeowHash	22	759,866,125	0.293 GiB/s
MeowHash	24	759,052,727	0.319 GiB/s
MeowHash	26	758,383,552	0.346 GiB/s
MeowHash	28	757,729,795	0.373 GiB/s
MeowHash	30	757,164,291	0.399 GiB/s
MeowHash	32	756,684,587	0.428 GiB/s
MeowHash	48	754,169,138	0.636 GiB/s
MeowHash	64	752,384,928	0.835 GiB/s
MeowHash	80	751,619,232	1.021 GiB/s
MeowHash	96	751,073,797	1.223 GiB/s
MeowHash	128	750,129,922	1.599 GiB/s
MeowHash	256	747,573,687	2.092 GiB/s
MeowHash	512	743,510,444	3.337 GiB/s
MeowHash	1024	736,579,233	5.831 GiB/s
MeowHash	2048	724,612,116	9.855 GiB/s
MeowHash	8192	703,809,114	19.043 GiB/s
Pippip_AES_TriXZi_Mikayla	2	2,398,457,856	0.081 GiB/s
Pippip_AES_TriXZi_Mikayla	3	2,381,779,284	0.107 GiB/s
Pippip_AES_TriXZi_Mikayla	4	1,045,900,603	0.144 GiB/s
Pippip_AES_TriXZi_Mikayla	5	772,898,928	0.168 GiB/s
Pippip_AES_TriXZi_Mikayla	6	770,348,762	0.216 GiB/s
Pippip_AES_TriXZi_Mikayla	7	769,211,357	0.252 GiB/s
Pippip_AES_TriXZi_Mikayla	8	768,105,495	0.287 GiB/s
Pippip_AES_TriXZi_Mikayla	9	767,144,693	0.324 GiB/s
Pippip_AES_TriXZi_Mikayla	10	766,297,023	0.357 GiB/s
Pippip_AES_TriXZi_Mikayla	11	765,541,930	0.390 GiB/s
Pippip_AES_TriXZi_Mikayla	12	764,910,240	0.426 GiB/s
Pippip_AES_TriXZi_Mikayla	13	764,162,803	0.460 GiB/s
Pippip_AES_TriXZi_Mikayla	14	763,563,327	0.495 GiB/s
Pippip_AES_TriXZi_Mikayla	15	763,033,074	0.529 GiB/s

Pippip_AES_TriXZi_Mikayla	16	762,491,793	0.563 GiB/s
Pippip_AES_TriXZi_Mikayla	18	761,595,592	0.487 GiB/s
Pippip_AES_TriXZi_Mikayla	20	760,700,964	0.541 GiB/s
Pippip_AES_TriXZi_Mikayla	22	759,883,069	0.594 GiB/s
Pippip_AES_TriXZi_Mikayla	24	759,027,321	0.645 GiB/s
Pippip_AES_TriXZi_Mikayla	26	758,356,861	0.701 GiB/s
Pippip_AES_TriXZi_Mikayla	28	757,771,038	0.754 GiB/s
Pippip_AES_TriXZi_Mikayla	30	757,179,626	0.808 GiB/s
Pippip_AES_TriXZi_Mikayla	32	756,695,416	0.861 GiB/s
Pippip_AES_TriXZi_Mikayla	48	754,175,971	1.065 GiB/s
Pippip_AES_TriXZi_Mikayla	64	752,375,921	1.418 GiB/s
Pippip_AES_TriXZi_Mikayla	80	751,645,188	1.748 GiB/s
Pippip_AES_TriXZi_Mikayla	96	751,045,916	2.074 GiB/s
Pippip_AES_TriXZi_Mikayla	128	750,128,912	2.456 GiB/s
Pippip_AES_TriXZi_Mikayla	256	747,607,556	3.400 GiB/s
Pippip_AES_TriXZi_Mikayla	512	743,504,333	5.969 GiB/s
Pippip_AES_TriXZi_Mikayla	1024	736,557,317	6.432 GiB/s
Pippip_AES_TriXZi_Mikayla	2048	724,638,132	9.513 GiB/s
Pippip_AES_TriXZi_Mikayla	8192	703,775,221	15.524 GiB/s

! 1st BEST ! 703,775,221 Pippip, 2nd 703,804,152 wyhash, 3rd

703,809,114 MeowHash

wyhash f4	2	2,398,457,855	0.064 GiB/s
wyhash f4	3	2,381,778,657	0.081 GiB/s
wyhash f4	4	1,045,897,660	0.108 GiB/s
wyhash f4	5	772,917,697	0.136 GiB/s
wyhash f4	6	770,366,681	0.163 GiB/s
wyhash f4	7	769,225,103	0.190 GiB/s
wyhash f4	8	768,106,881	0.218 GiB/s
wyhash f4	9	767,141,409	0.245 GiB/s
wyhash f4	10	766,302,477	0.272 GiB/s
wyhash f4	11	765,508,539	0.299 GiB/s
wyhash f4	12	764,867,479	0.325 GiB/s
wyhash f4	13	764,166,956	0.353 GiB/s
wyhash f4	14	763,560,575	0.380 GiB/s
wyhash f4	15	763,001,839	0.407 GiB/s
wyhash f4	16	762,506,359	0.434 GiB/s
wyhash f4	18	761,567,771	0.460 GiB/s
wyhash f4	20	760,704,713	0.511 GiB/s
wyhash f4	22	759,835,040	0.563 GiB/s
wyhash f4	24	759,047,896	0.613 GiB/s
wyhash f4	26	758,376,755	0.664 GiB/s
wyhash f4	28	757,717,840	0.715 GiB/s
wyhash f4	30	757,145,469	0.765 GiB/s
wyhash f4	32	756,681,334	0.816 GiB/s
wyhash f4	48	754,180,681	1.031 GiB/s
wyhash f4	64	752,405,740	1.321 GiB/s
wyhash f4	80	751,631,401	1.448 GiB/s
wyhash f4	96	751,044,437	1.453 GiB/s
wyhash f4	128	750,151,266	1.775 GiB/s
wyhash f4	256	747,566,422	3.255 GiB/s
wyhash f4	512	743,522,720	3.446 GiB/s
wyhash f4	1024	736,572,233	5.649 GiB/s
wyhash f4	2048	724,645,282	8.048 GiB/s
wyhash f4	8192	703,804,152	13.492 GiB/s

komihash 5.27	2	2,398,457,855	0.063 GiB/s
komihash 5.27	3	2,381,778,799	0.079 GiB/s
komihash 5.27	4	1,045,890,763	0.106 GiB/s
komihash 5.27	5	772,928,968	0.133 GiB/s
komihash 5.27	6	770,365,858	0.159 GiB/s
komihash 5.27	7	769,223,502	0.186 GiB/s
komihash 5.27	8	768,113,101	0.208 GiB/s
komihash 5.27	9	767,155,875	0.234 GiB/s
komihash 5.27	10	766,313,745	0.261 GiB/s
komihash 5.27	11	765,510,212	0.287 GiB/s
komihash 5.27	12	764,870,377	0.320 GiB/s
komihash 5.27	13	764,148,805	0.348 GiB/s
komihash 5.27	14	763,555,628	0.374 GiB/s
komihash 5.27	15	763,015,920	0.401 GiB/s
komihash 5.27	16	762,510,149	0.415 GiB/s
komihash 5.27	18	761,593,945	0.467 GiB/s
komihash 5.27	20	760,671,514	0.518 GiB/s
komihash 5.27	22	759,855,108	0.570 GiB/s
komihash 5.27	24	759,025,887	0.613 GiB/s
komihash 5.27	26	758,369,256	0.663 GiB/s
komihash 5.27	28	757,723,926	0.714 GiB/s
komihash 5.27	30	757,153,333	0.763 GiB/s
komihash 5.27	32	756,657,717	0.661 GiB/s
komihash 5.27	48	754,159,954	0.941 GiB/s
komihash 5.27	64	752,370,095	0.872 GiB/s
komihash 5.27	80	751,615,711	1.069 GiB/s
komihash 5.27	96	751,066,774	1.244 GiB/s
komihash 5.27	128	750,118,970	1.643 GiB/s
komihash 5.27	256	747,573,705	1.824 GiB/s
komihash 5.27	512	743,510,955	3.102 GiB/s
komihash 5.27	1024	736,543,186	5.031 GiB/s
komihash 5.27	2048	724,615,372	6.576 GiB/s
komihash 5.27	8192	703,811,325	10.268 GiB/s

XXH3_64bits 0.8.3	2	2,398,457,856	0.061 GiB/s
XXH3_64bits 0.8.3	3	2,381,778,999	0.078 GiB/s
XXH3_64bits 0.8.3	4	1,045,936,508	0.105 GiB/s
XXH3_64bits 0.8.3	5	772,905,350	0.131 GiB/s
XXH3_64bits 0.8.3	6	770,379,004	0.158 GiB/s
XXH3_64bits 0.8.3	7	769,224,813	0.184 GiB/s
XXH3_64bits 0.8.3	8	768,097,053	0.210 GiB/s
XXH3_64bits 0.8.3	9	767,136,024	0.244 GiB/s
XXH3_64bits 0.8.3	10	766,299,459	0.258 GiB/s

XXH3_64bits 0.8.3	11	765,554,050	0.298 GiB/s
XXH3_64bits 0.8.3	12	764,867,211	0.325 GiB/s
XXH3_64bits 0.8.3	13	764,149,701	0.353 GiB/s
XXH3_64bits 0.8.3	14	763,582,925	0.380 GiB/s
XXH3_64bits 0.8.3	15	763,030,121	0.407 GiB/s
XXH3_64bits 0.8.3	16	762,480,991	0.434 GiB/s
XXH3_64bits 0.8.3	18	761,593,374	0.398 GiB/s
XXH3_64bits 0.8.3	20	760,711,097	0.442 GiB/s
XXH3_64bits 0.8.3	22	759,853,204	0.485 GiB/s
XXH3_64bits 0.8.3	24	759,039,604	0.529 GiB/s
XXH3_64bits 0.8.3	26	758,362,363	0.573 GiB/s
XXH3_64bits 0.8.3	28	757,722,911	0.616 GiB/s
XXH3_64bits 0.8.3	30	757,156,842	0.661 GiB/s
XXH3_64bits 0.8.3	32	756,668,408	0.706 GiB/s
XXH3_64bits 0.8.3	48	754,160,728	0.944 GiB/s
XXH3_64bits 0.8.3	64	752,390,144	1.254 GiB/s
XXH3_64bits 0.8.3	80	751,664,567	1.133 GiB/s
XXH3_64bits 0.8.3	96	751,070,153	1.357 GiB/s
XXH3_64bits 0.8.3	128	750,135,964	1.721 GiB/s
XXH3_64bits 0.8.3	256	747,561,495	3.168 GiB/s
XXH3_64bits 0.8.3	512	743,521,669	3.733 GiB/s
XXH3_64bits 0.8.3	1024	736,574,831	6.118 GiB/s
XXH3_64bits 0.8.3	2048	724,635,104	9.620 GiB/s
XXH3_64bits 0.8.3	8192	703,819,509	17.015 GiB/s

8]

// Pippipify.c, ver. 2, written by chatGPT, 2025-Jul-26

```
// Pippipify.c - Streaming + One-go Pippip hash (128-bit AES-based)
// Compile: gcc -maes -msse4.1 -O3 Pippipify.c -o Pippipify.elf
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <immintrin.h>
#include <fcntl.h>
#include <unistd.h>
```

```
#define BLOCK_SIZE 4096
```

```
typedef struct {
    __m128i stateA;
    __m128i stateB;
    __m128i stateC;
    __m128i stateMIX;
    size_t total_len;
} PippipStreamState;
```

```
static const uint64_t CONST1 = 0x9e3779b97f4a7c15ULL;
static const uint64_t CONST2 = 0x6c62272e07bb0142ULL;
```

```
static inline uint64_t fold64(uint64_t A, uint64_t B) {
    __uint128_t r = (__uint128_t)A * B;
    return (uint64_t)r ^ (uint64_t)(r >> 64);
}
```

```
void PippipStream_Init(PippipStreamState *st, uint32_t seed) {
    __m128i base = _mm_set_epi64x(CONST2, CONST1);
    st->stateA = base;
    st->stateB = base;
    st->stateC = base;
    st->stateMIX = _mm_set1_epi32(seed);
    st->total_len = 0;
}
```

```
void PippipStream_Update(PippipStreamState *st, const void *data, size_t len) {
    const char *ptr = (const char *)data;
    st->total_len += len;
```

```
    while (len >= 16) {
        __m128i chunk = _mm_loadu_si128((const __m128i *)ptr);
        st->stateA = _mm_aesenc_si128(st->stateA, chunk);
        st->stateB = _mm_aesenc_si128(st->stateB, chunk);
        st->stateC = _mm_aesenc_si128(st->stateC, chunk);
        ptr += 16;
        len -= 16;
    }
}
```

```
if (len > 0) {
    char tail[16] = {0};
    memcpy(tail, ptr, len);
    __m128i chunk = _mm_loadu_si128((const __m128i *)tail);
    st->stateA = _mm_aesenc_si128(st->stateA, chunk);
    st->stateB = _mm_aesenc_si128(st->stateB, chunk);
    st->stateC = _mm_aesenc_si128(st->stateC, chunk);
}
```

```
void PippipStream_Final(PippipStreamState *st, uint32_t seed, void *output128) {
    st->stateMIX = _mm_aesenc_si128(st->stateMIX, st->stateA);
    st->stateMIX = _mm_aesenc_si128(st->stateMIX, st->stateB);
    st->stateMIX = _mm_aesenc_si128(st->stateMIX, st->stateC);
    __m128i lenvec = _mm_set1_epi32((int)st->total_len ^ seed);
    st->stateMIX = _mm_aesenc_si128(st->stateMIX, lenvec);
    _mm_storeu_si128((__m128i *)output128, st->stateMIX);
}
```

```

// One-go version
void PippipHash_OneGo(const void *data, size_t len, uint32_t seed, void *output128) {
    PippipStreamState st;
    PippipStream_Init(&st, seed);
    PippipStream_Update(&st, data, len);
    PippipStream_Final(&st, seed, output128);
}

void print_hash128(const uint8_t *h) {
    for (int i = 0; i < 16; ++i)
        printf("%02x", h[i]);
    printf("\n");
}

int main(int argc, char **argv) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    const char *fname = argv[1];
    uint8_t streamHash[16], onegoHash[16];
    PippipStreamState streamState;
    PippipStream_Init(&streamState, 1234);

    int fd = open(fname, O_RDONLY);
    if (fd < 0) {
        perror("open");
        return 1;
    }

    char *buffer = malloc(BLOCK_SIZE);
    if (!buffer) return 1;

    ssize_t r;
    size_t total = 0;
    while ((r = read(fd, buffer, BLOCK_SIZE)) > 0) {
        PippipStream_Update(&streamState, buffer, r);
        total += r;
    }
    close(fd);

    PippipStream_Final(&streamState, 1234, streamHash);

    FILE *fp = fopen(fname, "rb");
    if (!fp) return 1;
    char *fulldata = malloc(total);
    fread(fulldata, 1, total, fp);
    fclose(fp);

    PippipHash_OneGo(fulldata, total, 1234, onegoHash);
    free(fulldata);
    free(buffer);

    printf("Streaming hash: "); print_hash128(streamHash);
    printf("One-go hash:   "); print_hash128(onegoHash);
}

return 0;
}
***
```