**A Fast Hash Function and Pseudorandom Number Generator**

Yi Wang1,2, Chang Sun1,2, Diego Barrios Romero3, Daniel Lemire4, Li Jin1,2\*

1. Ministry of Education Key Laboratory of Contemporary Anthropology, Collaborative Innovation Center for Genetics and Development, School of Life Sciences, Shanghai, China.
2. Human Phenome Institute, Fudan University, Shanghai, China.
3. Fraunhofer Institute for Digital Medicine MEVIS, Bremen, Germany.
4. Université du Québec (TÉLUQ) Montreal, Canada

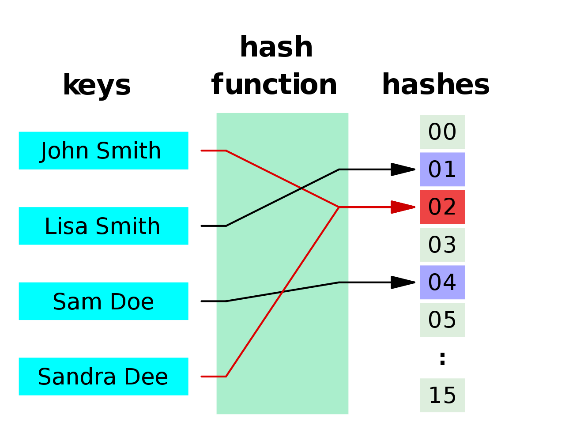
**Abstract**

Hash function and pseudorandom number generator (PRNG) are two fundamental functions in computer science. They have numerous applications and were estimated as important as algorithm and network. Even though hundreds of hash function and PRNG have been proposed in last decades, there are still rooms for improvement. We propose wyhash and wyrand as a new generation hash function and PRNG, which are of high quality, fast, portable and elegant. They passed SMHasher, PractRand and BigCrush tests. Benchmark and practice usage show that replacing exist hash function and PRNG with wyhash lead to significant speedup. wyhash and wyrand are completely free at https://github.com/wangyi-fudan/wyhash

**Introduction**

A hash function is a function to convert arbitrary data to a fixed-length integer. The input data was called the “keys” and the output integer was called the “hashes”. (figure1) Hash is a cornerstone of computer science and has numerous applications: fast algorithms, hash table, file checksum, duplication/collision detection, password storage, unique ID generation, proof-of-work, etc (1).

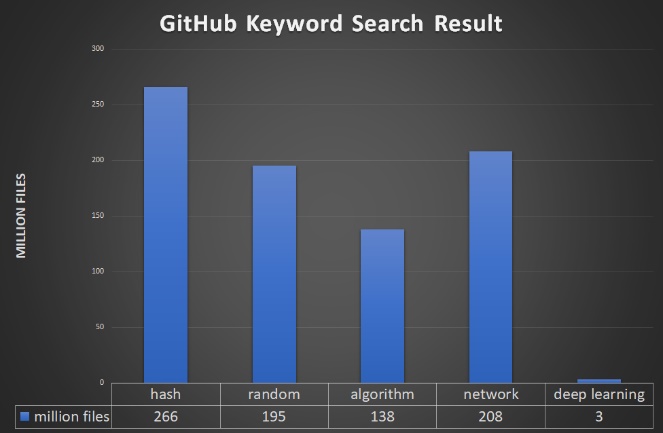
***Figure 1: Hash function explained***



A pseudo-random number generator (PRNG) is an algorithm that can generate a stream of numbers which appears random. The PRNG-generated sequence is not truly random, because it is completely determined by an initial value, called the “seed”. PRNG brings “randomness” to a deterministic computer, thus has wide applications: randomized algorithm, statistical sampling, simulation, gaming, cryptography, lottery, music and art, etc(1).

To introduce the importance of hash and PRNG to the public, we show the number of GitHub (1) files that contains the several keywords (figure 2). Hash and PRNG are roughly as important as algorithm and network in the computer world. Due to their importance and openness, numerous hash functions (2) and PRNGs (3) have been designed in last decades.

***Figure 2: Million of GitHub files that contain keywords***

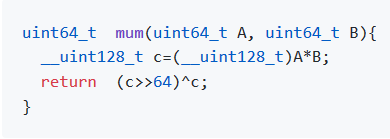


Despite the richness of hash functions and PRNGs, we still desire better hash function (especially for short keys) and PRNG for non-cryptographic applications. According to practical demand, an ideal hash function and PRNG should be (1) uniformly distributed (2) very fast (3) portable to different machine archtectures (4) small and elegant. Keeping these goals in mind, with one year of continual development, we are honored to announce wyhash, a new generation hash function and PRNG.

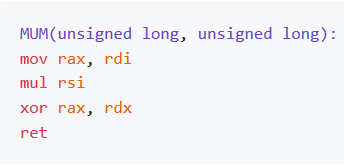
**Method**

***Mix Function:***

wyhash is based on a mix function call MUM that mix two 64-bit integer A and B to produce a 64-bit integer C: MUM(A,B) => C. @vnmakarov released the original version of MUM on Mother’s Day.



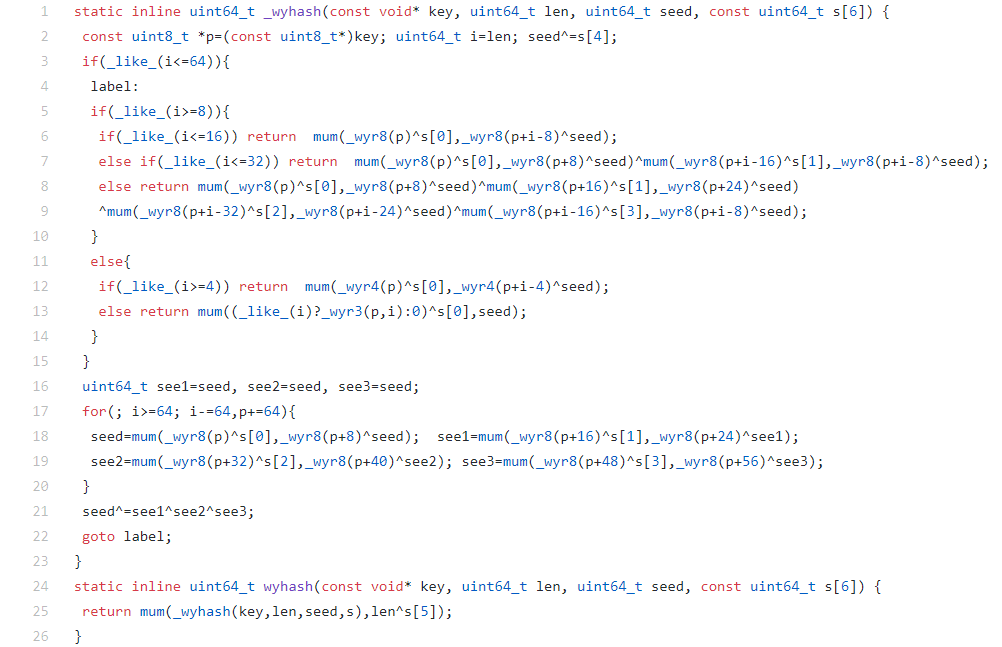
Despite the nominal 128-bit multiplication, the actual instructions on 64-bit machines are as simple as follow:



Our further improvements on MUM is the masked-MUM: MUM(A^secret, B^seed), where secret is predefined 64-bit integer with 32 1bits. The masked-MUM can protect the hasher, randomize biased real data and produce an avalanche effect. We observed experimentally that just two rounds of masked-MUM suffice to pass all statistical tests.

***\_wyhash and wyhash Hash Function***

\_wyhash hash function is based on masked-MUM and contains two parts: A return part and a batch part. The return part processes the tail bytes (<=64) while the batch part processes most of the data as 64-byte blocks. The wyhash function finalizes \_wyhash’s return value with a MUM. We expose \_wyhash to the user because \_wyhash contains the same amount of information as wyhash but faster. Then code is compact with only 26 lines of C code as follow, where \_wyr\* functions are inlined data read functions using memcpy:



***wyrand PRNG***

Our PRNG is named wyrand is even simpler. It keeps a 64-bit internal status and updates it by adding a 64-bit prime. The internal status is mixed with masked itself by MUM function to produce a pseudorandom number. It is obvious that its cycle length is 2^64 as p0 is a large prime.



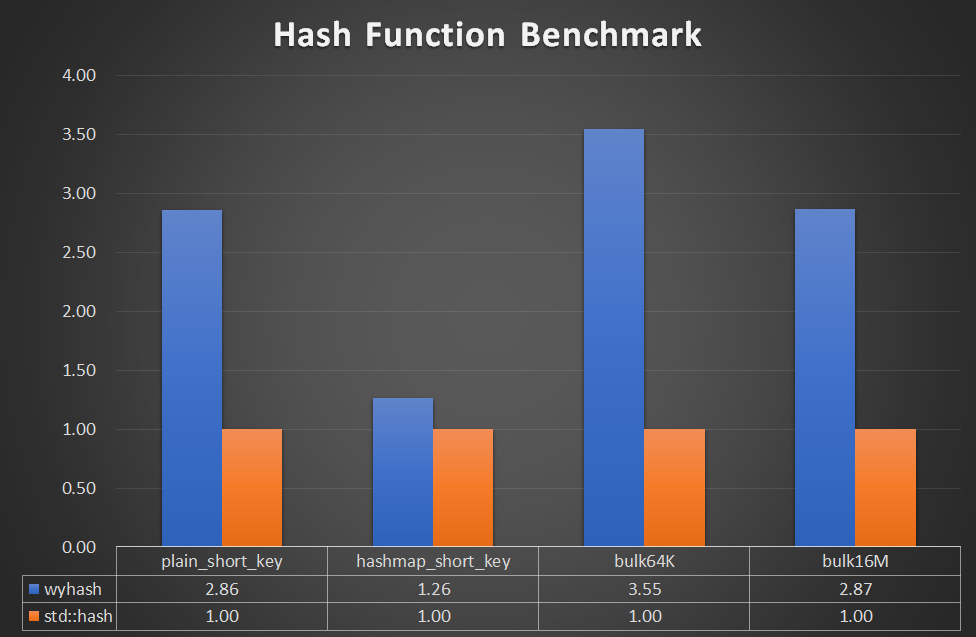
**Result**

We perform statistical quality test on wyhash by SMHasher. wyhash passed all quality tests. (supplemental material).

We performed statistical quality test of wyrand by PractRand and BigCrush via testingRNG suite. wyrand passed all tests despite of few suspicious false positives due to multiple testing (supplemental material).

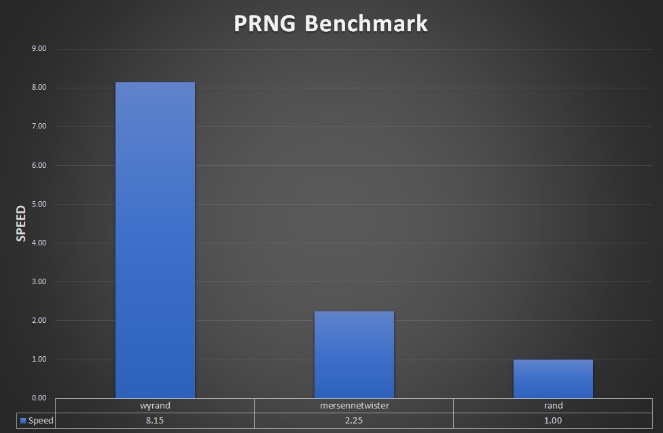
We benchmark the speed of wyhash against the C++ library hash function std::hash. Four benchmarks were carried: the plain short key hashing, the hashmap short key hashing, the 64KB bulk key hashing and the 16MB bulk key hashing. We employ /usr/shared/dict/words as testing corpus. Relative speed is shown in figure 3. SMHasher also lists benchmark of 150 hash functions, in which wyhash is the fastest one without quality problem.

***Figure 3: Hash function benchmark***



We benchmarked the speed of wyrand with rand and Mersenne Twister in testingRNG suite. We employed inlined benchmark code instead of testingRNG. Relative speed is shown in figure 4. Notably, wyrand is 8X faster than C library function rand and 3.5X faster than Mersenne Twister.

***Figure 4: PRNG Speed Benchmark***

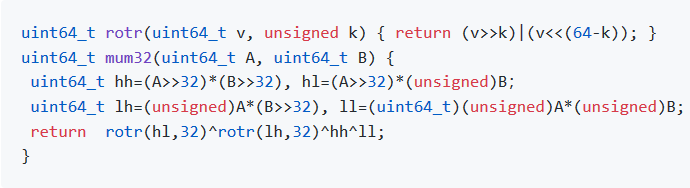


Portability Claim: wyhash and wyrand is portable on both 64-bit/32-bit/big endian/little endian machine. However, it is most efficient on 64-bit little endian machine which is most common today.

License Claim: wyhash and wyrand is an open source project and was distributed under UNLICENSE (<https://unlicense.org/>) which means completely free for the public.

**Discussion**

The MUM function is very fast on 64-bit machine but is slow on 32-bit machine. Yi Wang proposed an alternative MUM32 function to be efficient of 32-bit machine.



As @leo-yuriev pointed out, MUM function without xoring mask is vulnerable, as MUM(0, X)=0 for any X. As a solution to this problem, we evolved to the masked-MUM. By keeping the mask as secrets or randomized value, MUM can not be cracked trivially in non-cryptographic applications.

wyhash use memcpy to access memory safely. It does not depend on the “read through” method that read across memory bound. It does not do unaligned memory access which is unsafe on some machines. One key technique that make wyhash fast is the fast reading of tail memory (<8 byte). It is credit to Yann Collet.

The benchmark process, in fact, is complex. There are several benchmark suits available. There are common pitfalls in these suites. First, they employ function pointers, which disables powerful function inlining. Without function inlining, the speed is compromised. Second, they use artificial data, which is either predictable or unrealistic. To avoid these pitfalls, we developed our own simple benchmark code (supplement material) and benchmark it using /user/shared/dict/word.

Currently, wyhash had make certain impacts during its one-year exposure on GitHub. There are 181 stars, 13 repository, 442 files associated with it. wyhash is adopted by Microsoft MixedReality-Sharing project team who reported “much better” (https://github.com/microsoft/MixedReality-Sharing/issues/115). wyhash becomes the default hash function for V and Zig languages. The V language team reported “incredible” speed up (<https://github.com/vlang/v/pull/3591>).

**Acknowledgements**

We sincerely thank the following names due to their contributions to wyhash development: Reini Urban, Dietrich Epp, Joshua Haberman, Tommy Ettinger, Otmar Ertl, cocowalla, leo-yuriev, Diego Barrios Romero, paulie-g, dumblob, Yann Collet, ivte-ms, hyb, James Z.M. Gao.

**References**

1. https://github.com/
2. <https://github.com/rurban/smhasher>
3. <https://github.com/lemire/testingRNG>