The Ideal Hash Function and Pseudorandom Number Generator

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

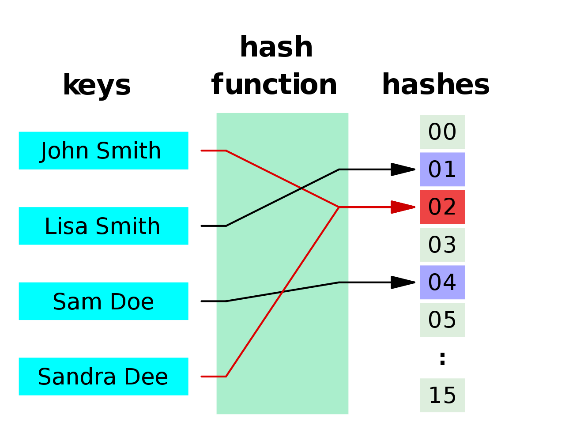
1. Ministry of Education Key Laboratory of Contemporary Anthropology, Collaborative Innovation Center for Genetics and Development, School of Life Sciences, Shanghai, China.
2. Human Phenome Institute, Fudan University, Shanghai, China.
3. Fraunhofer Institute for Digital Medicine MEVIS, Bremen, Germany.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

Abstract

**Introduction**

A hash function is a function to convert arbitary 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, eg: fast algorithms, hash table, file checksum, duplication/collision detection, password storage, unique ID generation, proof-of-work, etc.

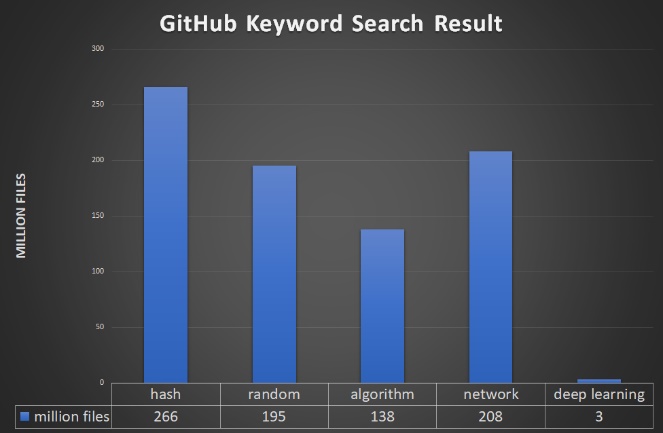
***Figure 1: Hash function explained***



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 determinstic computer, thus has wide applications, eg: randomized algorithm, statistical sampling, simulation, gaming, cryptography, lottery, music and art, etc.

The importance of an algorithm can be roughly accessed by the number of GitHub files that contains the name of the algorithm (figure 2). Due to their importance and openess, numerous hash functions (https://github.com/rurban/smhasher) and PRNGs (https://github.com/lemire/testingRNG) have been designed in last decades.

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



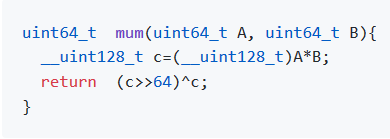
Despite the richness of hash functions and PRNGs, we still desire the ideal hash function and PRNG for non-crypographic applications. According to practical demand, the ideal hash function and PRNG should be (1) uniformly distributed (2) very fast (3) portable (4) elegent. After one year of development, we are honored to announce wyhash (https://github.com/wangyi-fudan/wyhash), the ideal hash function and PRNG.

**Method**

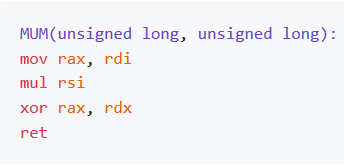
***Mix Function:***

wyhash is based on a mix function call MUM that take two 64-bit integer A and B and produces 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 seed-masked-MUM: MUM(A^seed^p0, B^seed^p1), where p0 and p1 are random prime masks containing 32 1s. The seed-masked-MUM can protect the hasher, randomize biased real data and produce an avalanche effect. We observed experimentally that just two rounds of seed-masked-MUM suffice to pass all statistical tests.

***wyhash Hash Function***

wyhash hash function based on seed-masked-MUM contains two parts: A quick-return part and a batch part. The quick-return part processes the first K=(length mod 64) bytes and try to return if possible while the batch part processes the rest of the data as 64-byte blocks. Then code is compact and elegant with 21 lines of C code as follow.



***wyrand PRNG***

Our PRNG is named wyrand. 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.



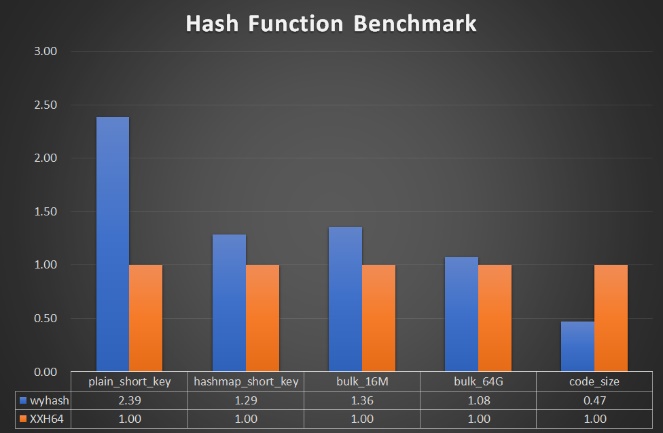
**Result**

We perform statistical quality test on wyhash by SMHasher. Wyhash passed all quality tests. (suppelemental 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).

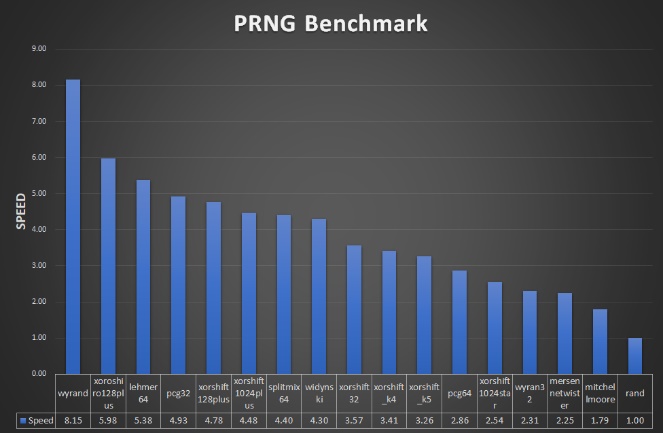
We benchmark the speed of wyhash against the popular XXH64 hash function. Four benchmarks were carried: the plain short key hashing, the hashmap short key hashing, the 16MB bulk key hashing and the 64GB bulk key hashing. We employ /usr/shared/dict/words as testing corpus and repeat 10000 times for short key hashing. We also list the compiled code size based on SMHasher. Relative speed is shown in figure 3.

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



We benchmarked the speed of wyrand with other 16 PRNGs in testingRNG suite. We employed inlined benchmark code instead of testingRNG. Relative speed is shown in figure 4.

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

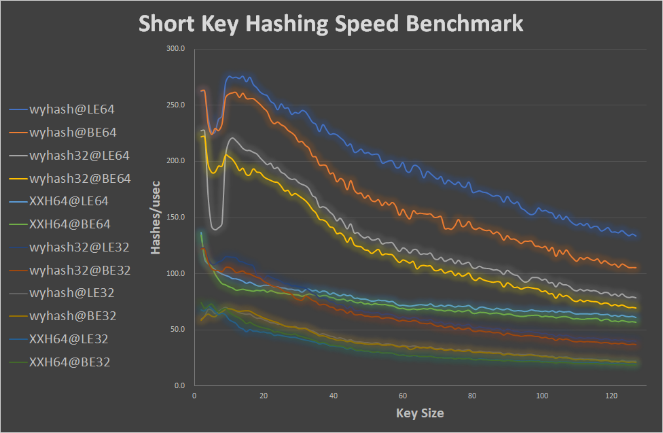


Portability Claim: wyhash and wyrand is portable on both 64-bit/32-bit/big endian/little endian machine. However, it is most efficient on 64-bit little endian machine which is most common today.

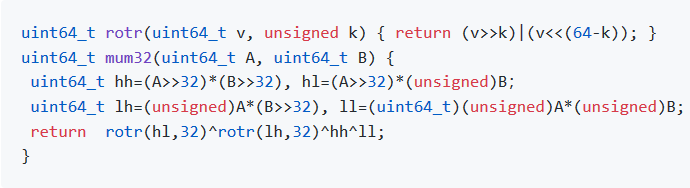
**Discussion**

The MUM function is very fast on 64-bit machine but is slow on 32-bit machine. (Supplemental Figure 1)

***Supplemental Figure1: Benchmark on different achitecture***



Yi Wang proposed an alternative MUM32 function to be efficient of 32-bit machine. It is adopted by Microsoft MixedReality-Sharing project (https://github.com/microsoft/MixedReality-Sharing).



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

Currently, wyhash had make certain impacts during its expousure on GitHub. There are 179 stars, 13 repository, 442 files asscociated with it. It becomes the default hash function for V and Zig languages.

**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.