

State of the hash functions, 2012

January 29, 2012

The state-of-the-art in non-cryptographic hash functions has advanced rapidly in the last few years. When I did some searching this week I was happy to see that new cutting-edge hash functions had been released even since last time I looked into this 6 months or a year ago.

Non-cryptographic hash functions take a string as input and compute an integer output. The desirable property of a hash function is that the outputs are evenly distributed across the domain of possible outputs, especially for inputs that are similar. Unlike a cryptographic hash function, these functions are *not* designed to withstand an effort by an attacker to find a collision. Cryptographic hash functions have this property, but are much slower: [SHA-1 is on the order of 0.09 bytes/cycle](#) whereas the newest non-cryptographic hash functions are on the order of 3 bytes/cycle. So non-cryptographic hashes are roughly 33x faster, at the cost of not being able to withstand attacks. Non-cryptographic hashes are most often used for hash tables.

As an interesting aside, [there is a debate going on in the Lua community right now](#) about what, if anything, should be done about the fact that Lua's hash function could theoretically be attacked to force its hash table implementation into its $O(n)$ worst-case lookup performance. This could let an attacker DoS you if he is feeding you input that you are putting into a Lua hash table. The Lua authors are somewhat skeptical about how realistic this attack is (and whether it would be cheaper than other DoS alternatives), but are moving ahead anyway with a plan to generate a random seed at startup that the hash function will use. This is an interesting alternative to cryptographic hash functions that should be able to give you the same collision resistance as a cryptographic hash function (presuming you have an entropy source that can give you truly random bits), but at the cost of non-reproducible output.

Since there are lots of options out there for non-cryptographic hash functions and this number keeps expanding, I thought I'd summarize my knowledge of what is out there.

Bob Jenkins' Functions

[Bob Jenkins](#) has been working on hash functions for 15 years or so. In 1997 he published an article about hash functions in Dr. Dobbs Journal; the article is available now on the web with

more content added since its original publication: [A hash function for hash Table lookup](#). In this article Bob has an extensive catalog of existing hash functions, as well as presenting his own called “lookup2.” Bob subsequently published [lookup3](#) in 2006, which for the purposes of this article I will consider the first “modern” hash function, in the sense that it is both fast (0.5 bytes/cycle, according to Bob) and free of any serious flaws.

More information about Bob’s functions can be found on Wikipedia: [Jenkins hash function](#).

Second generation: MurmurHash

In 2008 Austin Appleby published a new hash function called [MurmurHash](#). In its most recent version it is roughly 2x the speed of lookup3 (so roughly 1 byte/cycle), and it comes in both 32 and 64-bit versions. The 32-bit version uses only 32-bit math and gives you a 32-bit hash, the 64-bit version uses 64-bit math and gives a 64-bit hash. According to Austin’s analysis it has excellent properties, though Bob Jenkins says in his expanded Dr. Dobbs article “I can see [MurmurHash is] weaker than my lookup3, but I don’t by how much, I haven’t tested it.” MurmurHash quickly became popular thanks to its excellent speed and statistical properties.

Third generation: CityHash and SpookyHash

In 2011 two hash functions were released that both improve on MurmurHash due largely to greater instruction-level parallelism. Google released [CityHash](#) (written by Geoff Pike and Jyrki Alakuijala) and Bob Jenkins released a new hash of his own, [SpookyHash](#) (so named because it was released on Halloween). Both functions are on the order of 2x the speed of MurmurHash, but both functions use 64-bit math and have no 32-bit version, and CityHash depends on the CRC32 instruction that is present in SSE 4.2 (Intel Nehalem and later) for its speed. SpookyHash gives you 128-bit output, whereas CityHash has 64-bit, 128-bit, and 256-bit variants.

Which function is best/fastest?

From what I can see, all of the hash functions I mentioned in this article are good enough from a statistical perspective. One consideration is that only CityHash/SpookyHash give more than 64 bits of output, but for a hash table 32 bits of output is plenty. Other applications may have use for 128 or 256 bit output.

If you’re on 32-bit, MurmurHash looks like the clear winner since it’s the only function faster than lookup3 that has a native 32-bit version. 32-bit machines could probably compile and

run City and Spooky, but I would expect it to be much slower because the 64-bit math would have to be emulated.

On 64-bit machines it's hard to say which is best without further benchmarking. I'd be liable to prefer Spooky to City since the latter depends on the CRC32 instruction for speed which isn't available everywhere.

One other consideration is aligned vs. unaligned access. MurmurHash (unlike City or Spooky) comes in a variant that will only perform aligned reads, since on many architectures unaligned reads will crash or return the wrong data (unaligned reads are undefined behavior in C). City and Spooky both address the issue by copying the input data into aligned storage with `memcpy()`; Spooky does the `memcpy()` a block at a time (if `ALLOW_UNALIGNED_READS` is not defined), City does the `memcpy()` an integer at a time! On machines that can handle unaligned reads (like x86 and x86-64) the `memcpy` will be optimized away, but I did a test on my little ARM box and found that this:

```
#include <stdint.h>
#include <string.h>
int32_t read32_unaligned(const void *buf) {
    int32_t ret;
    memcpy(&ret, buf, 4);
    return ret;
}
```



compiles to this very inefficient code (this would be a single instruction on x86):

```
0: b500      push {lr}
2: 2204      movs r2, #4
4: b083      sub sp, #12
6: 4601      mov r1, r0
8: eb0d 0002  add.w r0, sp, r2
c: f7ff fffe  bl 0
10: 9801      ldr r0, [sp, #4]
12: b003      add sp, #12
14: bd00      pop {pc}
```

To conclude, MurmurHash still looks like the best option if you need 32-bit or aligned-only reads. CityHash and SpookyHash look to be faster on x86-64, but I would almost think of them as being specific to that architecture, since I'm not aware of other architectures that are both 64-bit and allow unaligned reads.

Please let me know of any errors in this article.

Josh Haberman
jhaberman@gmail.com

 haberman
 JoshHaberman

Parsing, performance, and low-level
programming.

