
Subject: New hash folding function...

Posted by [mirek](#) on Sun, 20 Jan 2008 15:24:46 GMT

[View Forum Message](#) <> [Reply to Message](#)

Do not ask me why, but today I have got a mood to experiment with hasing once again.

The primary motivation was ARM - because it lacks DIV instruction, therefore modulo computations there would be quite slow.

Of course, U++ associativity depends on specific hashing and `_did_` (until today) depended on division to extract maximum entrophy from 32-bit hash code to map it to hashing space.

Therefore I was today experimenting with alternatives. In the end I have found this strange operation:

```
unsigned HashFold(unsigned h)
{
    return (h >> 23) - (h >> 9) - (h >> 15) - h;
}
```

which "readies" the hash code to be masked with the mapping space range (which can be done in 2^n steps in U++). Suprisingly, this method seems superior to former U++ method, which was: range size is a prime number and mapping is done by modulo prime number.

Moreover, despite many more operations involved, it is still 3x faster than modulo (division is quite slow even on modern CPU, because it is done in 2 bit steps, means >16 cycles to get single result; AFAIK, this should be reduced on Penryn cpu, which divides by 4 bit steps).

As result, there is about 10% improvement in associative benchmarks, like `idmapBench` (funny, even in 2007 U++ it was pretty fast, e.g. compared to STL:

[http://www.ultimatepp.org/www\\$upweb\\$vsstd\\$en-us.html](http://www.ultimatepp.org/www$upweb$vsstd$en-us.html)

and now it is about 60% faster than that

This is the code and results I used to test this new invention:

```
#include <Core/Core.h>
```

```
using namespace Upp;
```

```
int q = 0;
```

```
#define N 100 * 1000 * 1000
```

```

void Benchmark()
{
{
    RTIMING("XOR-folding");
    for(int h = 0; h < N; h++) {
        q += 255 & ((h >> 24) ^ (h >> 16) ^ (h >> 8) ^ h);
    }
}
{
    RTIMING("%");
    for(int i = 0; i < N; i++) {
        q += i % 257;
    }
}
{
    RTIMING("&");
    for(int i = 0; i < N; i++) {
        q += i & 255;
    }
}
}

```

```

unsigned HashFold(unsigned h)
{
    return (h >> 23) - (h >> 9) - (h >> 15) - h;
}

```

```

template <class T>
void TestCollisions(const char *desc, const Index<T>& data)
{
    VectorMap<unsigned, int> hash1, hash2, hash3;
    int m1 = 0, m2 = 0, m3 = 0;
    int a = Pow2Bound(data.GetCount()) - 1;
    int mod = PrimeBound(data.GetCount());
    for(int i = 0; i < data.GetCount(); i++) {
        unsigned h = GetHashValue(data[i]);
        m1 = max(m1, ++hash1.GetAdd(h % mod, 0));
        m2 = max(m2, ++hash2.GetAdd(a & HashFold(h), 0));
    }
    LOG(desc << ", unique " << data.GetCount());
    LOG("mod unique: " << hash1.GetCount() << ", worst: " << m1 << " (% " << mod << ")");
    LOG("fold unique: " << hash2.GetCount() << ", worst: " << m2 << " (& " << a << ")");
}

```

```

void TestCollisions(String desc, const char *file)
{
    FileIn in(file);
    Index<String> data;

```

```

while(!in.IsEof())
    data.FindAdd(in.GetLine());
TestCollisions(desc + " " + file, data);
}

CONSOLE_APP_MAIN {
    Benchmark();
    int q = 0;
    for(int i = 0; i < 100000; i++)
        if((HashFold(i) & 1023) == 0) {
            LOG(i << " - " << i - q);
            q = i;
        }
    TestCollisions("", "e:/bookmarks.html");
    TestCollisions("", "e:/test.txt");
    Index<String> x;
    for(int i = 0; i < 1000 * 100; i++)
        x.FindAdd(FormatIntBase(i, 2));
    TestCollisions("Bin100", x);
    x.Clear();
    for(int i = 0; i < 1000 * 10; i++)
        x.FindAdd(FormatIntBase(i, 2));
    TestCollisions("Bin10", x);
    x.Clear();
    for(int i = 0; i < 1000 * 100; i++)
        x.FindAdd(FormatIntBase(i, 16));
    TestCollisions("Hex100", x);
    x.Clear();
    for(int i = 0; i < 1000 * 10; i++)
        x.FindAdd(FormatIntBase(i, 16));
    TestCollisions("Hex10", x);

    for(int n = 10; n <= 1000 * 1000; n = 10 * n) {
        x.Clear();
        for(int i = 0; i < n; i++)
            x.FindAdd(FormatIntBase(i, 10));
        TestCollisions("Dec" + AsString(n), x);
    }
    x.Clear();
    FileIn in("d:/uppsrc/CtrlLib/ArrayCtrl.cpp");
    for(;;) {
        int c = in.Get();
        if(c < 0) break;
        if(isalpha(c) || c == '_') {
            String id;
            id.Cat(c);
            c = in.Get();
            while(c >= 0 && (isalnum(c) || c == '_')) {

```

```

    id.Cat(c);
    c = in.Get();
}
x.FindAdd(id);
}
else
if(isdigit(c))
do c = in.Get();
while(c >= 0 && (isalnum(c) || c == '.'));
}
TestCollisions("cpp ids", x);

```

```

Index<int> y;
for(int i = 0; i < 100000; i++)
y.FindAdd(i);
TestCollisions("i100000", y);
y.Clear();
for(int i = 0; i < 30000; i++)
y.FindAdd(rand());
TestCollisions("i rand", y);
}

```

```

e:/bookmarks.html, unique 84
mod unique: 64, worst: 3 (% 127)
fold unique: 62, worst: 3 (& 127)
e:/test.txt, unique 22949
mod unique: 16260, worst: 10 (% 32771)
fold unique: 16384, worst: 8 (& 32767)
Bin100, unique 100000
mod unique: 60556, worst: 10 (% 131071)
fold unique: 66805, worst: 7 (& 131071)
Bin10, unique 10000
mod unique: 7263, worst: 7 (% 16381)
fold unique: 7490, worst: 5 (& 16383)
Hex100, unique 100000
mod unique: 32654, worst: 25 (% 131071)
fold unique: 64290, worst: 10 (& 131071)
Hex10, unique 10000
mod unique: 6611, worst: 7 (% 16381)
fold unique: 6679, worst: 8 (& 16383)
Dec10, unique 10
mod unique: 10, worst: 1 (% 17)
fold unique: 10, worst: 1 (& 15)
Dec100, unique 100
mod unique: 65, worst: 3 (% 127)
fold unique: 63, worst: 3 (& 127)

```

Dec1000, unique 1000
mod unique: 572, worst: 5 (% 1021)
fold unique: 533, worst: 6 (& 1023)
Dec10000, unique 10000
mod unique: 6800, worst: 3 (% 16381)
fold unique: 4170, worst: 9 (& 16383)
Dec100000, unique 100000
mod unique: 29176, worst: 35 (% 131071)
fold unique: 69480, worst: 7 (& 131071)
Dec1000000, unique 1000000
mod unique: 621155, worst: 8 (% 1048573)
fold unique: 632889, worst: 8 (& 1048575)
cpp ids, unique 704
mod unique: 512, worst: 4 (% 1021)
fold unique: 519, worst: 4 (& 1023)
i100000, unique 100000
mod unique: 100000, worst: 1 (% 131071)
fold unique: 100000, worst: 1 (& 131071)
i rand, unique 19668
mod unique: 19668, worst: 1 (% 32771)
fold unique: 19640, worst: 2 (& 32767)

For now, I have replaced former modulo with this new thing. I welcome discussion about this new scheme; hashing can be quite tricky and maybe some common usage scenario can lead to pathologic cases here.. (= too many colissions).

Mirek
