The report[b7-335-c0596]

Intergalactic language translator.

Code name: hashtable

Category: ~~obsolete software~~ self-upgrading software
Origin: unknown
Location: 3rd orbit of Sun-573, Milky Way glx., 14th iteration of realm
Rarity: UNIQUE
State: lost in reverse time*

Description: given a dictionary of predefined format, grants access to the interface of the translator unit that not only outstrips our current most advanced systems, but disburses much less time, than theoretically possible.
Further notes show progress of hashtable through it's registered evolution.

Hardware:

Intel® Core™ i5-5200U CPU @ 2.20GHz × 4

Compilation line: `g++ -Wall -Wextra -std=c++17 -lm -O1 -march=native -masm=intel`

Hashtable info:

- Open addressing
- FNV64 hash
- Automatic rehashing
- Hash preservation
- Parallel computations possibilities

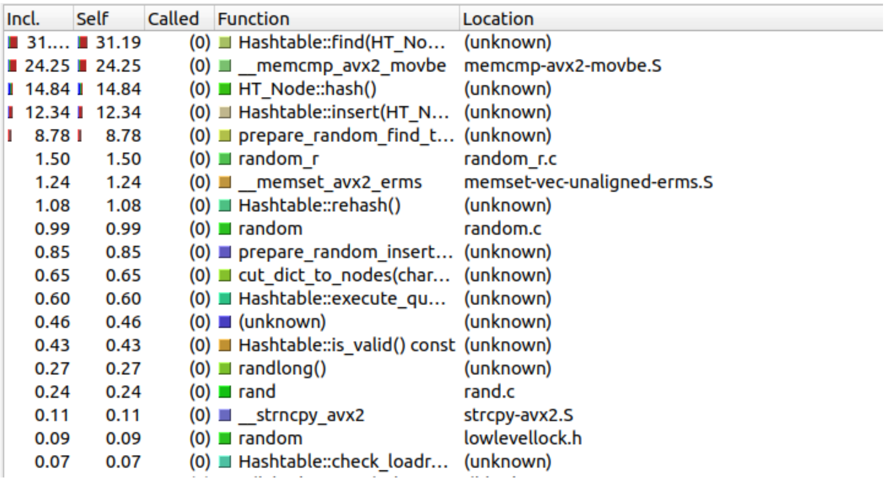
Tests info:

- 10 tests run per function
- 10^6 random pairs to find
- 10^5 random pairs to insert
- All the tests are generated and loaded to the memory before actual testing, so random doesn't affect our measurements

ITERATION 01

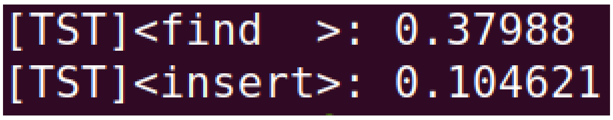
Simple hashtable, no asm optimization. All architectural optimizations were encoded straight from the beginning.

Cashgrind:



Incl.	Self	Called	Function	Location
31....	31.19	(0)	Hashtable::find(HT_No...	(unknown)
24.25	24.25	(0)	__memcmp_avx2_movbe	memcmp-avx2-movbe.S
14.84	14.84	(0)	HT_Node::hash()	(unknown)
12.34	12.34	(0)	Hashtable::insert(HT_N...	(unknown)
8.78	8.78	(0)	prepare_random_find_t...	(unknown)
1.50	1.50	(0)	random_r	random_r.c
1.24	1.24	(0)	__memset_avx2_erms	memset-vec-unaligned-erms.S
1.08	1.08	(0)	Hashtable::rehash()	(unknown)
0.99	0.99	(0)	random	random.c
0.85	0.85	(0)	prepare_random_insert...	(unknown)
0.65	0.65	(0)	cut_dict_to_nodes(char...	(unknown)
0.60	0.60	(0)	Hashtable::execute_qu...	(unknown)
0.46	0.46	(0)	(unknown)	(unknown)
0.43	0.43	(0)	Hashtable::is_valid() const	(unknown)
0.27	0.27	(0)	randlong()	(unknown)
0.24	0.24	(0)	rand	rand.c
0.11	0.11	(0)	__strncpy_avx2	strcpy-avx2.S
0.09	0.09	(0)	random	lowlevellock.h
0.07	0.07	(0)	Hashtable::check_loadr...	(unknown)

Real time:

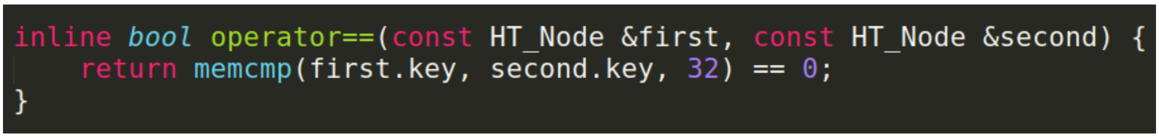


```
[TST]<find  >: 0.37988
[TST]<insert>: 0.104621
```

Observations:

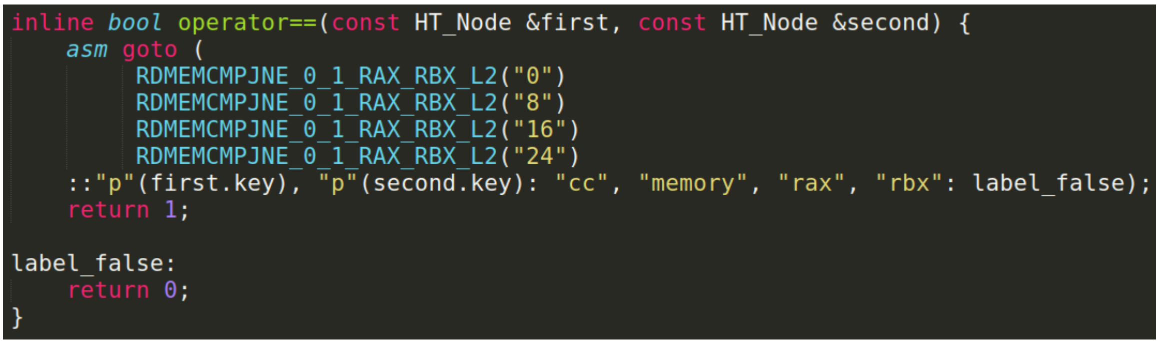
As Hashtable::find just iterates through memory one cell by one, it leaves no place for possible optimization. Valuable insight is that it actually calls memcmp all the time, and this is the right place for optimization, as we know, that keys of hashtable are of fixed 64-chars length. It's obvious that Node::hash also takes much time, but we can't optimize them together, if we want to see real benefit of our work.

So, let's replace



```
inline bool operator==(const HT_Node &first, const HT_Node &second) {
    return memcmp(first.key, second.key, 32) == 0;
}
```

With

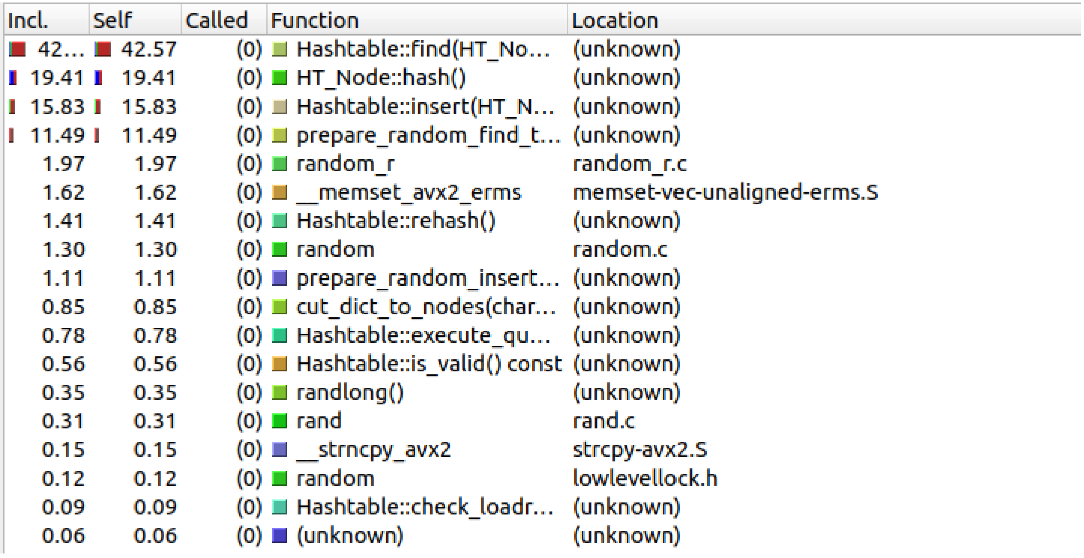


```
inline bool operator==(const HT_Node &first, const HT_Node &second) {
    asm goto (
            RDMEMCMPJNE_0_1_RAX_RBX_L2("0")
            RDMEMCMPJNE_0_1_RAX_RBX_L2("8")
            RDMEMCMPJNE_0_1_RAX_RBX_L2("16")
            RDMEMCMPJNE_0_1_RAX_RBX_L2("24")
    ::"p"(first.key), "p"(second.key): "cc", "memory", "rax", "rbx": label_false);
    return 1;

label_false:
    return 0;
}
```

ITERATION 02

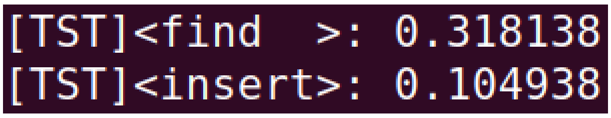
Hashtable with inline asm optimization

Cashgrind:



Incl.	Self	Called	Function	Location
42…	42.57	(0)	Hashtable::find(HT_No…	(unknown)
19.41	19.41	(0)	HT_Node::hash()	(unknown)
15.83	15.83	(0)	Hashtable::insert(HT_N…	(unknown)
11.49	11.49	(0)	prepare_random_find_t…	(unknown)
1.97	1.97	(0)	random_r	random_r.c
1.62	1.62	(0)	__memset_avx2_erms	memset-vec-unaligned-erms.S
1.41	1.41	(0)	Hashtable::rehash()	(unknown)
1.30	1.30	(0)	random	random.c
1.11	1.11	(0)	prepare_random_insert…	(unknown)
0.85	0.85	(0)	cut_dict_to_nodes(char…	(unknown)
0.78	0.78	(0)	Hashtable::execute_qu…	(unknown)
0.56	0.56	(0)	Hashtable::is_valid() const	(unknown)
0.35	0.35	(0)	randlong()	(unknown)
0.31	0.31	(0)	rand	rand.c
0.15	0.15	(0)	__strncpy_avx2	strcpy-avx2.S
0.12	0.12	(0)	random	lowlevellock.h
0.09	0.09	(0)	Hashtable::check_loadr…	(unknown)
0.06	0.06	(0)	(unknown)	(unknown)

Real time:

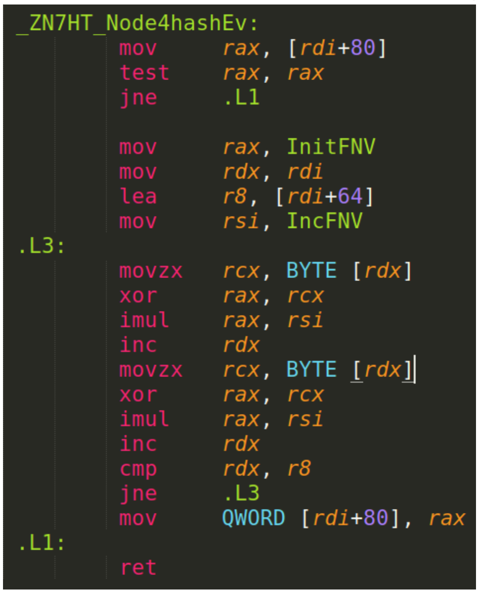


```
[TST]<find  >: 0.318138
[TST]<insert>: 0.104938
```

Observations:

We have already received Ded’s coefficient 306 by inserting writing 4 lines of asm. But as we are aimed at speed, and not at score, so let’s further enhance hashtable. Find is still an untouchable function, so let’s fully assemblize Node::hash().

Instead of 10 lines of C code we get



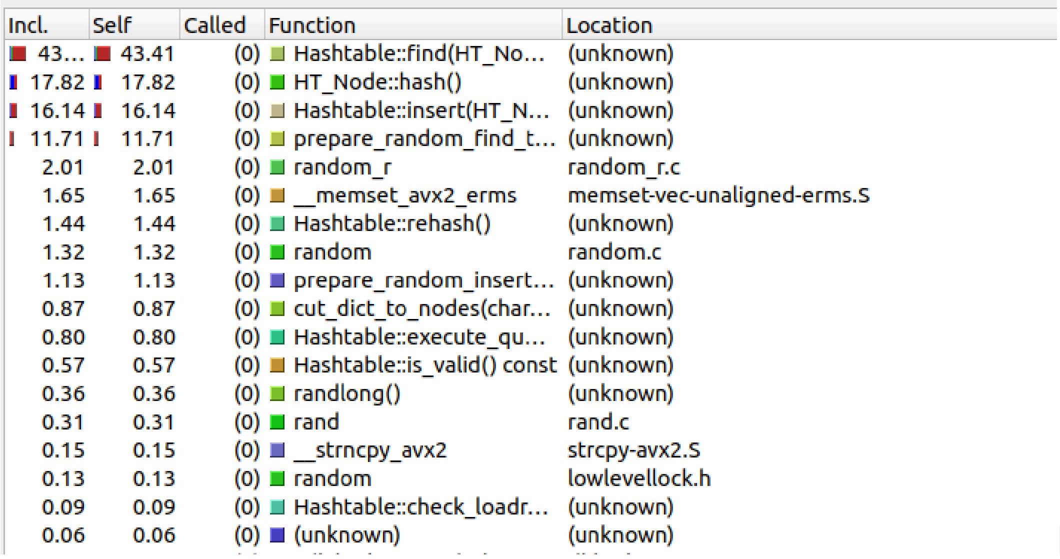
```
_ZN7HT_Node4hashEv:
        mov     rax, [rdi+80]
        test    rax, rax
        jne     .L1

        mov     rax, InitFNV
        mov     rdx, rdi
        lea     r8, [rdi+64]
        mov     rsi, IncFNV
.L3:
        movzx   rcx, BYTE [rdx]
        xor     rax, rcx
        imul    rax, rsi
        inc     rdx
        movzx   rcx, BYTE [rdx]
        xor     rax, rcx
        imul    rax, rsi
        inc     rdx
        cmp     rdx, r8
        jne     .L3
        mov     QWORD [rdi+80], rax
.L1:
        ret
```

It turned out to be exactly 2 bytes to be resolved in one iteration, which correlates with the duality of our perishable world.

ITERATION 03

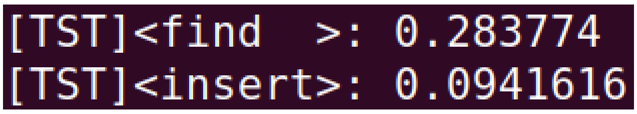
Hashtable with both inline asm and externally linked .o file with hashing function

Cashgrind:



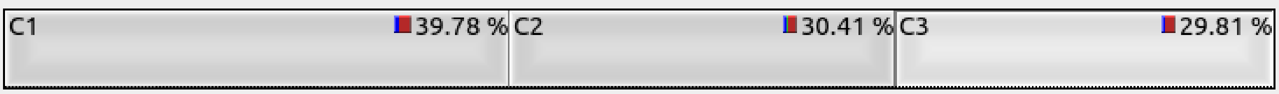
Incl.	Self	Called	Function	Location
43…	43.41	(0)	Hashtable::find(HT_No…	(unknown)
17.82	17.82	(0)	HT_Node::hash()	(unknown)
16.14	16.14	(0)	Hashtable::insert(HT_N…	(unknown)
11.71	11.71	(0)	prepare_random_find_t…	(unknown)
2.01	2.01	(0)	random_r	random_r.c
1.65	1.65	(0)	__memset_avx2_erms	memset-vec-unaligned-erms.S
1.44	1.44	(0)	Hashtable::rehash()	(unknown)
1.32	1.32	(0)	random	random.c
1.13	1.13	(0)	prepare_random_insert…	(unknown)
0.87	0.87	(0)	cut_dict_to_nodes(char…	(unknown)
0.80	0.80	(0)	Hashtable::execute_qu…	(unknown)
0.57	0.57	(0)	Hashtable::is_valid() const	(unknown)
0.36	0.36	(0)	randlong()	(unknown)
0.31	0.31	(0)	rand	rand.c
0.15	0.15	(0)	__strncpy_avx2	strcpy-avx2.S
0.13	0.13	(0)	random	lowlevellock.h
0.09	0.09	(0)	Hashtable::check_loadr…	(unknown)
0.06	0.06	(0)	(unknown)	(unknown)

Real time:



```
[TST]<find  >: 0.283774
[TST]<insert>: 0.0941616
```

Observations: difference in cashgrind can be upsetting, but actually we scratched out 2% performance boost by small asm adjustments:



C1 39.78 % C2 30.41 % C3 29.81 %

C1 – iteration 01
C2 – iteration 02
C3 – iteration 03

Results

We speed up hashtable. Noice.

Test type	C1	C2	C3	Speedup coef.
Find	0.379	0.318	0.283	1.33
Insert	0.104	0.104	0.094	1.10

Max Ded'd coef: 308
Overall Ded's coef: 178

* During further iterations hashtable started to consume a negative amount of time to translate new samples and, as a result, started traveling in the reverse time direction. Declared lost.