Table of Contents

Congruent Modulo Bitboards was introduced by Trevor Fenner and Mark Levene in the ICGA Journal, Vol. 31, No. 1 in 2008 [1]. While their Perfect Hashing approach provides great mathematical insights in Congruent Modulo arithmetic, their final conclusion in comparison with Hashing Dictionaries, Rotated Bitboards and Magic Bitboards was criticized by the obvious comparison with Kindergarten Bitboards [2].

Modulo vs. Multiplication

BitScan broaches the issue of Perfect Hashing with Modulo versus Multiplication as well:

- Bitscan by Modulo
- De Bruijn Multiplication

So does the SWAR-Popcount, when it is about to finally add byte-wise populations:

- Casting out
- Multiplication

Modulo

Congruence relation

Fenner and Levene use masked lines (not necessarily excluding the sliding piece), that is bitboards with N=8 active bits with k={7,8,9} bits apart, starting with bit zero

-

Based on Congruence relation

-

or equivalently

-

they deduced two general perfect hashing functions. The case N

```
masked occupany  %  256-1           =  A-H
. . . . . . . H     . . . . . . . .     . . . . . . . .
. . . . . . G .     . . . . . . . .     . . . . . . . .
. . . . . F . .     . . . . . . . .     . . . . . . . .
. . . . E . . .     . . . . . . . .     . . . . . . . .
. . . . . . . .  %  . . . . . . . .  =  . . . . . . . .
. . C . . . . .     . . . . . . . .     . . . . . . . .
. B . . . . . .     . . . . . . . .     . . . . . . . .
A . . . . . . .     1 1 1 1 1 1 1 1     A B C . E F G H
```

Reciprocal Multiplication

The 64-bit modulo by a constant can be done most efficiently by reciprocal fixed point multiplication, this is how Microsoft Visual C++ 2005 compiler implements the mod constant for x86-64 processors. One 64*64=128 bit multiplication, one shift, one further 32-bit multiplication , one subtraction. Of course using 64-bit division to get the remainder burns even more cycles.

```
Code:
% 514
 mov    r11d, r10 ; masked diagonal
 mov    rax, ff00ff00ff00ff01H
 mul    r10
 shr    rdx, 9
 imul   edx, 514 ; 00000202H
 sub    r11d, edx

% 257
 mov    r11d, r10 ; masked diagonal
 mov    rax, ff00ff00ff00ff01H
 mul    r10
 shr    rdx, 8
 imul   edx, 257 ; 00000101H
 sub    r11d, edx
```

Multiplication

A Kindergarten like approach might look like this (not considering inner six bits):

```
U64 arrDiagonalAttacks[256][64];  // 128 K

U64 diagonalAttacks(U64 occ, enumSquare sq) {
   occ = (diagonalMask[sq] & occ) * C64(0x0101010101010101) >> 56;
   return arrDiagonalAttacks[occ][sq];
}
```

and uses one 64*64=64-bit multiplication, with this x86-64 assembly for calculating an eight-bit occupied index:

```
 mov    rax, 0101010101010101H
 imul   rdx, rax
 shr    rdx, 56
```

Even Kindergarten File-Attacks are cheaper and faster, not to mention Magic Bitboards, which covers two lines of a rook or bishop in one run.

Fenner's and Levene's conclusion

Quote from their paper pp 11
3.5. Comparison with other Methods

- As reported in Hyatt (2007), the rotated and magic bitboard methods are of comparable performance, and Tannous (2007) claims just a small improvement of the direct lookup method over rotated bitboards. It is easy to see that, in terms of the number of computer operations, the efficiency of our method will be similar to that of direct lookup. Thus we are justified in claiming that the computational efficiency of our method is comparable to the others.

Their conclusion was based on following statement of Robert Hyatt [4] **...**

- I had reported this earlier. Magic was no faster than rotated. I switched because of two things...

- 1. Magic is simpler, and simpler is better as I get older.
- 2. Magic gives you the opportunity to update the occupied_squares and then generate moves easily.

- To do this with rotated bitboards first requires that all rotated bitboards be updated in addition to the normal occupied_squares bitboard. This is faster, if you use the feature (I don't yet, but well might at times).

... and this claim of Sam Tannous [5]:

- The results shown indicate that directly looking up the attacking moves for sliding pieces in hash tables improves the move generation speeds from 10% to 15% depending on computer architecture. Further efficiencies can be expected in a full implementation where the overhead of maintaining rotated bitboards is eliminated. The implementation and test code is made available in an Open-Source, interactive, chess programming module called “Shatranj” (Tannous, 2006).

Publications

- Zbigniew J. Czech, George Havas, Bohdan S. Majewski (**1997**). *Perfect Hashing*. Theoretical

Computer Science, Vol. 182, Nos. 1-2, pp. 1-143
- Trevor Fenner, Mark Levene (**2008**). *Move Generation with Perfect Hashing Functions.* ICGA Journal, Vol. 31, No. 1, pp. 3-12. pdf

Forum Posts

- Nice Math - Strange Conclusions by Gerd Isenberg, CCC, April 29, 2008
- Low memory usage attack bitboard generation by crystalclear, Winboard Forum, October 06, 2011

External Links

- Congruence relation from Wikipedia
- Linear congruence theorem from Wikipedia
- Modular arithmetic from Wikipedia
- Modulo operation from Wikipedia

References

1. ^ Trevor Fenner, Mark Levene (**2008**). *Move Generation with Perfect Hashing Functions.* ICGA Journal, Vol. 31, No. 1, pp. 3-12. pdf
2. ^ Nice Math - Strange Conclusions by Gerd Isenberg, CCC, April 29, 2008
3. ^ Low memory usage attack bitboard generation by crystalclear, Winboard Forum, October 06, 2011
4. ^ Re: BitBoard Tests Magic v Non-Rotated 32 Bits v 64 Bits by Robert Hyatt, CCC August 25, 2007
5. ^ Sam Tannous (**2007**). *Avoiding Rotated Bitboards with Direct Lookup*. ICGA Journal, Vol. 30, No. 2, pp. 85-91, pdf

What links here?

Page	Date Edited
Congruent Modulo Bitboards	Jun 26, 2013
Efficient Generation of Sliding Piece Attacks	Nov 5, 2016
General Setwise Operations	Feb 25, 2018
Hash Table	Jan 1, 2018
ICGA Journal	Dec 21, 2017
Mark Levene	Jun 2, 2015
Sliding Piece Attacks	May 27, 2016
Trevor Fenner	Jun 2, 2015

Up one Level