WCSC31 Qugiy アピール文書 2

森　大慶

2021 年 5 月 18 日

1　はじめに

とても運が良いことに、Qugiy は今大会で決勝進出することができ、また独創賞まで頂いてしまいました。ビット演算で利きを求めるアレで賞を頂いたので、この文書では選手権前に提出したアピール文書では触れていなかった飛車、角の利きについて書こうと思います。

2　開発動機

正月、暇だったのでネットサーフィンをしていると AWAKE というコンピュータ将棋が題材の映画が取り上げられている記事を見つけました。数年前に一度将棋プログラムを作ろうとして途中でやめてしまったのを思い出し、今回はルール通り将棋が指せるところまで書ききるぞ！という気持ちで開発をはじめました。

3　飛車、角の利き

飛車、角の利き関数と、テーブルの初期化コードを掲載します。以下のソースコードは Qugiy のビットボード定義を想定しています（が、やねうら王でもそのままで動いた気がします）。Qugiy での定義は選手権前に提出したアピール文書を参照してください。PseudoAttacks の初期化も混じっていますが無視してください。

```
/*
    SQUARE_BIT[i]: マス i に対応するビットが立ったビットボード
    part<N>()    : N=0 ならビットボードの下位 64bit を，1 なら上位 64bit を取得するメソッド
*/

// テーブルの定義
alignas(32) uint64_t BISHOP_MASK[2][81][4];
alignas(16) uint64_t ROOK_MASK[2][81][2];

// 角の利き
inline Bitboard bishopAttacks(int i, const Bitboard& occupancy) {
    const __m256i shuffle = _mm256_set_epi8(
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
```

```
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
    );
    const __m256i bishop_mask_lo = _mm256_load_si256((__m256i*)BISHOP_MASK[0][i]);
    const __m256i bishop_mask_hi = _mm256_load_si256((__m256i*)BISHOP_MASK[1][i]);

    __m256i occ2 = _mm256_broadcastsi128_si256(occupancy.xmm());
    __m256i rocc2 = _mm256_shuffle_epi8(occ2, shuffle);
    __m256i hi = _mm256_unpackhi_epi64(occ2, rocc2);
    __m256i lo = _mm256_unpacklo_epi64(occ2, rocc2);
    hi = _mm256_and_si256(hi, bishop_mask_hi);
    lo = _mm256_and_si256(lo, bishop_mask_lo);
    __m256i t1 = _mm256_add_epi64(hi, _mm256_cmpeq_epi64(lo, _mm256_setzero_si256()));
    __m256i t0 = _mm256_add_epi64(lo, _mm256_set1_epi64x(-1ULL));
    t1 = _mm256_and_si256(_mm256_xor_si256(t1, hi), bishop_mask_hi);
    t0 = _mm256_and_si256(_mm256_xor_si256(t0, lo), bishop_mask_lo);
    __m256i a2 = _mm256_shuffle_epi8(_mm256_unpackhi_epi64(t0, t1), shuffle);
    a2 = _mm256_or_si256(a2, _mm256_unpacklo_epi64(t0, t1));
    return _mm_or_si128(_mm256_castsi256_si128(a2), _mm256_extracti128_si256(a2, 1));
}

// 飛車の利き
inline Bitboard rookAttacks(int i, const Bitboard& occupancy) {
    const __m128i shuffle = _mm_set_epi8(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15);
    const __m128i rook_mask_lo = _mm_load_si128((__m128i*)ROOK_MASK[0][i]);
    const __m128i rook_mask_hi = _mm_load_si128((__m128i*)ROOK_MASK[1][i]);

    __m128i rocc = _mm_shuffle_epi8(occupancy.xmm(), shuffle);
    __m128i hi = _mm_unpackhi_epi64(occupancy.xmm(), rocc);
    __m128i lo = _mm_unpacklo_epi64(occupancy.xmm(), rocc);
    hi = _mm_and_si128(hi, rook_mask_hi);
    lo = _mm_and_si128(lo, rook_mask_lo);
    __m128i t1 = _mm_add_epi64(hi, _mm_cmpeq_epi64(lo, _mm_setzero_si128()));
    __m128i t0 = _mm_add_epi64(lo, _mm_set1_epi64x(-1ULL));
    t1 = _mm_xor_si128(t1, hi);
    t0 = _mm_xor_si128(t0, lo);
    t1 = _mm_and_si128(t1, rook_mask_hi);
    t0 = _mm_and_si128(t0, rook_mask_lo);
    __m128i updown = _mm_shuffle_epi8(_mm_unpackhi_epi64(t0, t1), shuffle);
    updown = _mm_or_si128(updown, _mm_unpacklo_epi64(t0, t1));
```

```
    return lanceAttacks<BLACK>(i, occupancy)
         | lanceAttacks<WHITE>(i, occupancy)
         | Bitboard(updown);
}

// 角のテーブル初期化
void init_BISHOP_MASK() {
    for (int i = 0; i < 9; ++i) {
        for (int j = 0; j < 9; ++j) {
            Bitboard rightup(_mm_setzero_si128());
            for (int k = 1; k <= std::min(i, 8 - j); ++k)
                rightup |= SQUARE_BIT[(i - k) + (j + k) * 9];
            Bitboard leftup(_mm_setzero_si128());
            for (int k = 1; k <= std::min(8 - i, 8 - j); ++k)
                leftup |= SQUARE_BIT[(i + k) + (j + k) * 9];
            Bitboard rightdown(_mm_setzero_si128());
            for (int k = 1; k <= std::min(i, j); ++k)
                rightdown |= SQUARE_BIT[(i - k) + (j - k) * 9];
            Bitboard leftdown(_mm_setzero_si128());
            for (int k = 1; k <= std::min(8 - i, j); ++k)
                leftdown |= SQUARE_BIT[(i + k) + (j - k) * 9];

            // ついでに PseudoAttacks も初期化
            BISHOP_PSEUDO_ATTACKS[i + j * 9] = leftup | rightup | rightdown | leftdown;

            rightdown = _mm_shuffle_epi8(rightdown.xmm(), _mm_set_epi8(
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15));
            leftdown  = _mm_shuffle_epi8(leftdown.xmm() , _mm_set_epi8(
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15));

            BISHOP_MASK[1][i + j * 9][0] = rightup  .part<1>();
            BISHOP_MASK[1][i + j * 9][1] = leftdown .part<1>();
            BISHOP_MASK[1][i + j * 9][2] = leftup   .part<1>();
            BISHOP_MASK[1][i + j * 9][3] = rightdown.part<1>();

            BISHOP_MASK[0][i + j * 9][0] = rightup  .part<0>();
            BISHOP_MASK[0][i + j * 9][1] = leftdown .part<0>();
            BISHOP_MASK[0][i + j * 9][2] = leftup   .part<0>();
            BISHOP_MASK[0][i + j * 9][3] = rightdown.part<0>();
        }
```

```
    }
}

// 飛車のテーブル初期化
void init_ROOK_MASK() {
    for (int i = 0; i < 9; ++i) {
        for (int j = 0; j < 9; ++j) {
            Bitboard up(_mm_setzero_si128());
            for (int k = 1; k <= 8 - j; ++k)
                up |= SQUARE_BIT[i + (j + k) * 9];

            Bitboard down(_mm_setzero_si128());
            for (int k = 1; k <= j; ++k)
                down |= SQUARE_BIT[i + (j - k) * 9];

            // ついでに PseudoAttacks も初期化
            Bitboard left(_mm_setzero_si128());
            for (int k = 1; k <= 8 - i; ++k)
                left |= SQUARE_BIT[(i + k) + j * 9];

            Bitboard right(_mm_setzero_si128());
            for (int k = 1; k <= i; ++k)
                right |= SQUARE_BIT[(i - k) + j * 9];

            ROOK_PSEUDO_ATTACKS [i + j * 9] = left | up | right | down;
            LANCE_PSEUDO_ATTACKS[i + j * 9][BLACK] = right;
            LANCE_PSEUDO_ATTACKS[i + j * 9][WHITE] = left;

            down = _mm_shuffle_epi8(down.xmm(), _mm_set_epi8(
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15));

            ROOK_MASK[1][i + j * 9][0] = up  .part<1>();
            ROOK_MASK[1][i + j * 9][1] = down.part<1>();

            ROOK_MASK[0][i + j * 9][0] = up  .part<0>();
            ROOK_MASK[0][i + j * 9][1] = down.part<0>();
        }
    }
}
```

4 実験結果

Qugiy には Magic Bitboard を実装していないため、やねうら王に実装して速度比較をしました。

■置換表 1GB, 1 **スレッド**, depth=19

	time_e1	nodes_e1	nps_e1	time_e2	nodes_e2	nps_e2	nps_diff
0	17970	22460409	1249883	17376	22460409	1292611	42728
1	17867	22460409	1257088	17425	22460409	1288976	31888
2	17883	22460409	1255964	17403	22460409	1290605	34641
3	17828	22460409	1259838	17392	22460409	1291421	31583
4	17890	22460409	1255472	17370	22460409	1293057	37585
5	17886	22460409	1255753	17320	22460409	1296790	41037
6	17862	22460409	1257440	17330	22460409	1296042	38602
7	17855	22460409	1257933	17302	22460409	1298139	40206
8	17895	22460409	1255122	17319	22460409	1296865	41743
9	17851	22460409	1258215	17357	22460409	1294025	35810

	time_e1	nodes_e1	nps_e1	time_e2	nodes_e2	nps_e2	nps_diff
count	10	10	10	10	10	10	10
mean	17879	22460409	1256271	17359	22460409	1293853	37582
std	38	0	2676	41	0	3043	3998
min	17828	22460409	1249883	17302	22460409	1288976	31583
25%	17857	22460409	1255542	17322	22460409	1291718	34933
50%	17875	22460409	1256526	17364	22460409	1293541	38094
75%	17889	22460409	1257810	17388	22460409	1296603	40829
max	17970	22460409	1259838	17425	22460409	1298139	42728

```
Result of 10 runs
==================
base           =    1256271 +/- 2676
test           =    1293853 +/- 3043
diff           =     +37582 +/- 3998

speedup        = +0.0299
P(speedup > 0) = 1.0000

Vendor ID         : AuthenticAMD
CPU Name          : AMD Ryzen 5 3600 6-Core Processor
Microarchitecture : x86_64
```

■**置換表** 1GB, 12 **スレッド**, depth=19

	time_e1	nodes_e1	nps_e1	time_e2	nodes_e2	nps_e2	nps_diff
0	6322	60220033	9525471	3721	37832468	10167285	641814
1	5275	51663910	9794106	4355	44842894	10296875	502769
2	4877	47339800	9706745	7180	70646510	9839346	132601
3	5178	51261782	9899919	5453	54992259	10084771	184852
4	6877	68141848	9908659	5109	52461532	10268454	359795
5	6803	65484524	9625830	6484	66316828	10227764	601934
6	5708	56561438	9909151	3791	38700044	10208399	299248
7	5754	55802825	9698092	5980	58420601	9769331	71239
8	4508	43599834	9671657	10472	103546544	9887943	216286
9	5503	54548347	9912474	7821	77349262	9889945	-22529

	time_e1	nodes_e1	nps_e1	time_e2	nodes_e2	nps_e2	nps_diff
count	10	10	10	10	10	10	10
mean	5680	55462434	9765210	6037	60510894	10064011	298801
std	787	7636000	139659	2076	20019892	198156	225419
min	4508	43599834	9525471	3721	37832468	9769331	-22529
25%	5202	51362314	9678266	4544	46747554	9888444	145664
50%	5606	55175586	9750426	5716	56706430	10126028	257767
75%	6180	59305384	9906474	7006	69564090	10222923	467026
max	6877	68141848	9912474	10472	103546544	10296875	641814

```
Result of 10 runs
==================
base           =    9765210 +/- 139659
test           =   10064011 +/- 198156
diff           =    +298801 +/- 225419

speedup        = +0.0306
P(speedup > 0) = 0.9994

Vendor ID         : AuthenticAMD
CPU Name          : AMD Ryzen 5 3600 6-Core Processor
Microarchitecture : x86_64
```

ちゃんと計ると 3% ぐらいの高速化みたいです。