WCSC31 Qugiy アピール文書

森　大慶

2021 年 3 月 31 日

1　はじめに

Qugiy は今年の 1 月からフルスクラッチで開発をしている将棋ソフトです。やねうら王や技巧などをとても参考にしています。

探索部は従来の minimax 型のもので、評価関数は NNUE です。評価関数データは自前のではなく、水匠 3 改を利用する予定です。初出場なのでお手柔らかにお願いします。

2　Qugiy の特徴

2.1　飛び駒の利きをビット演算で計算

飛び駒の利きは、Magic Bitboard など表引きの手法で求めるのが一般的ですが、Qugiy ではビット演算で計算しています。

同じものをやねうら王に実装してベンチを取ると、Magic Bitboard のビルドと比べて 4% ほど高速化するようです（@Ryzen 5 3600）。Zen2 の CPU では PEXT 命令が遅いので、PEXT の実装との速度比較はしていません。

Magic Bitboard のように巨大なテーブルは必要ない（おかげで速くなる？）ですし、めんどくさい初期化も要らないので実装が楽なのもメリットです。

これらを全部公開するとやねうら王が 4% 速くなってしまうので、一部（香車の利き計算）だけ紹介することにします。

2.1.1　ビットボードレイアウト

Qugiy のビットボードのレイアウトと将棋盤は以下のように対応しています。

Bitboard と将棋盤の関係

9	8	7	6	5	4	3	2	1	
00	09	00	09	18	27	36	45	54	一
01	10	01	10	19	28	37	46	55	二
02	11	02	11	20	29	38	47	56	三
03	12	03	12	21	30	39	48	57	四
04	13	04	13	22	31	40	49	58	五
05	14	05	14	23	32	41	50	59	六
06	15	06	15	24	33	42	51	60	七
07	16	07	16	25	34	43	52	61	八
08	17	08	17	26	35	44	53	62	九
p[0]		p[1]							

いわゆる縦型ビットボードですが、縦型ならなんでも同じだろうと思っていたら、このレイアウトだと同一直線上にビットが 2 つ以上あるかを判定するときに p[0] と p[1] を or してから popcount... ができないという欠陥がありました。直すのがめんどくさいのでこのままにしてあります。このように、Qugiy のビットボードのレイアウトがやねうら王と異なっているため、以下のコードはそのままやねうら王に貼り付けても動きませんが、定数の部分を変えれば正しく動作します。

2.1.2 後手の香車の利き

このようなビットボードのレイアウトの場合、後手の香車の利きは次のように求められます。

```
template <Color C>
inline Bitboard lanceAttacks(int i, const Bitboard& occupancy) {
    if (C == BLACK) {
        ...
    }
    else if (C == WHITE) {
        __m128i mask = _mm_set_epi64x(0x3fdfeff7fbfdfeffULL, 0x000000000001feffULL);
        __m128i em = _mm_andnot_si128(occupancy.xmm(), mask);
        __m128i t = _mm_add_epi64(em, PAWN_ATTACKS[WHITE][i].xmm());
        return Bitboard(_mm_xor_si128(t, em));
    }
}
```

PAWN_ATTACKS[WHITE][i] は後手の歩がマス i にいるときの利きです。（後手の）香車であればたったこれだけで求めることができます。が、先手はこの方法で求められないのでトータルでどっちが速いかは知りません（適当）。4% の高速化の部分はおそらく飛車角の部分のおかげなので...

2.1.3 先手の香車の利き

先手の香車ですが、上のコードほど効率的ではありませんが一応ビット演算で求まります。コードは以下です。

```
if (C == BLACK) {
    __m128i mocc = _mm_and_si128(BLACK_LANCE_PSEUDO_ATTACKS[i].xmm(), occupancy.xmm());
    mocc = _mm_or_si128(mocc, _mm_srli_epi64(mocc, 1));
    mocc = _mm_or_si128(mocc, _mm_srli_epi64(mocc, 2));
    mocc = _mm_or_si128(mocc, _mm_srli_epi64(mocc, 4));
    mocc = _mm_srli_epi64(mocc, 1);
    return Bitboard(_mm_andnot_si128(mocc, BLACK_LANCE_PSEUDO_ATTACKS[i].xmm()));
}
```

BLACK_LANCE_PSEUDO_ATTACKS[i] はマス i にある香車の、盤上に何も駒がないときの利きを予め求めたテーブルです。

以上でビット演算による利き計算の紹介は終わりです。飛車角もこれの応用なので、ぜひ考えてみてください。

2.2 指し手生成の高速化

Qugiy の指し手生成の速度 (1 秒間での Capture(+ 歩の成り)、Quiet(− 歩の成り) の生成回数) は初期盤面で 15M 回/s、指し手生成祭りの局面で 8M 回/s、最大合法手局面で 4M 回/s ぐらいです。その主要な工夫をまとめます。

2.2.1 二歩判定の高速化

ビット演算ばかりで申し訳ないのですが、二歩判定もビット演算で高速化できました。全体の速度にはそこまで影響していない気がするのでコードを公開します。

```
inline Bitboard pawn_drop_mask(const Bitboard& pawns) {
    __m128i left = _mm_set_epi64x(0x40201008040201000ULL, 0x0000000000020100ULL);
    __m128i t = _mm_sub_epi64(left, pawns.xmm());
    t = _mm_srli_epi64(_mm_and_si128(t, left), 8);
    return Bitboard(_mm_xor_si128(left, _mm_sub_epi64(left, t)));
}
```

この関数は、歩のビットボードから、歩のいない筋を 1 で埋めたビットボードを返す関数です。香車の利き計算でのビット演算を応用して、複数の筋について同時に計算しています。同じ要領で複数の香車の利きを一発で求めたりできると思いますが、あまり高速化できる処理がなさそうなので使っていません。

なにげにお気に入りです。

2.2.2 SIMD による駒打ち高速化

駒打ちの生成において AVX2 命令で複数種類の駒を同時に処理すると、指し手生成祭りの局面での生成速度が 2 倍くらいになりました。上の二歩判定よりも圧倒的に効果があるのでおすすめです。

3 現時点での強さ

PVS、指し手のオーダリング、置換表での枝刈りなどの後ろ向き枝刈りに加えて、Futility Pruning, Null Move Pruning, ProbCut, Late Move Reduction などの前向き枝刈りは（条件がかなり適当ですが）実装が終わっています。一方マルチスレッドでの探索や、延長系はまだ実装していません。

Qugiy は floodgate に AisakaTaiga という名前で参加していて、現在のレーティングは 3000 程度です。作者は将棋のルールしか知らないのでどれくらいの強さなのかよく分かっていないのですが、レート的にはまだまだ弱いのでこれから強くしていきたいです。

4 選手権までに実装したい or 試したいもの

予告なく変更される場合があるので予めご了承ください。

1. Lazy SMP
2. ponder 対応
3. １手詰め
4. † Singular Extension †
5. Large Page？の利用
6. SIMD による指し手ソートの高速化
7. まともな時間制御
8. 宣言勝ち

5 最後に

ここまでご覧いただきありがとうございました！