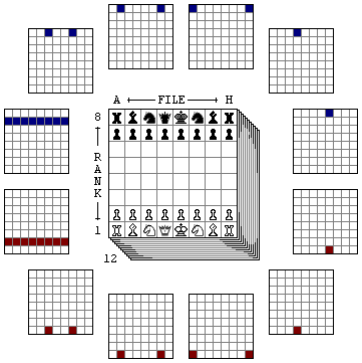
Bitboard Board-Definition (/Bitboard+Board-Definition)

Table of Contents

To represent the board we typically need one bitboard for each piece-type and color - likely encapsulated inside a class or structure, or as an array of bitboards as part of a position object. A one-bit inside a bitboard implies the existence of a piece of this piece-type on a certain square - one to one associated by the bit-position [1]:



To be aware of their scalar 64-bit origin, we use so far a type defined unsigned integer U64 in our C/C++ source snippets, the scalar 64-bit long in Java. Feel free to define a distinct type or wrap U64 into classes for better abstraction and type-safety during compile time.

Classical Board

Those bitboards may part of a central position object which is incrementally updated while making or unmaking moves.

Structure

```
class CBoard
{
   U64 whitePawns;
   U64 whiteKnights;
   U64 whiteBishops;
   U64 whiteRooks;
   U64 whiteQueens;
   U64 whiteKing;

   U64 blackPawns;
   U64 blackKnights;
   U64 blackBishops;
   U64 blackRooks;
   U64 blackQueens;
   U64 blackKing;
   ...
};
```

Array

For better generalization and to avoid branches, one may encapsulate arrays of bitboards. For instance, inside the Beowulf sources (sample from moves.c) one finds a lot of branches on side to move to either fetch white or black piece bitboards, as already criticized by Vincent Diepeveen in 2001 [2] ...

```
  switch (side) {
   case WHITE: tsq = B->whiteRooks; break;
   case BLACK: tsq = B->blackRooks; break;
  }
```

.. where an indexed access with appropriate defined {0,1} color range for the side to move would avoid those branches, per piece-kind, ...

```
  tsq = B->rooks[side];
```

... or over all piece-kinds, ...

```
  tsq = B->pieceBB[nWhiteRook + side];
```

... for instance, on x86 or x86-64, utilizing its addressing modes with base- and scalable index register , plus displacement:

```
   ; rsi pointer to structure, rcx side (0 == White, 1 == Black)
   mov  rax, qword ptr [rsi + rookOffset + 8*rcx]
```

Likely one also keeps some often used redundant union sets like white and black pieces, occupancy or their complement, the empty squares.

```
class CBoard
{
   U64 pieceBB[14];
   U64 emptyBB;
   U64 occupiedBB;
   ...
public:
   enum enumPiece
   {
      nWhite,     // any white piece
```

```
      nBlack,     // any black piece
      nWhitePawn, // white Pawn
      nBlackPawn, // white Pawn
      ...
   };

   U64 getPieceSet(PieceType pt) const {return pieceBB[pt];}
   U64 getWhitePawns() const {return pieceBB[nWhitePawn];}
   ...
   U64 getPawns(ColorType ct) const {return pieceBB[nWhitePawn + ct];}
   ...
};
```

On initialization and update of the bitboards, see general setwise operations.

Denser Board

A common alternative to reduce the size of the board structure is to keep two color bitboards and six color independent piece bitboards, which are the union of black and white respective pieces, i.e. all queens. This space saving requires a cheap intersection of a color and a piece bitboard to get the required pieces of that color only.

```
class CBoard
{
   U64 pieceBB[8];
public:
   enum enumPiece
   {
      nWhite,     // any white piece
      nBlack,     // any black piece
      nPawn,
      nKnight,
      nBishop,
      nRook,
      nQueen,
      nKing
   };

   U64 getPieceSet(PieceType pt) const {return pieceBB[pieceCode(pt)] & pieceBB[colorCode(pt)];}
   U64 getWhitePawns() const {return pieceBB[nPawn] & pieceBB[nWhite];}
   ...
   U64 getPawns(ColorType ct) const {return pieceBB[nPawn] & pieceBB[ct];}
   ...
};
```

See also

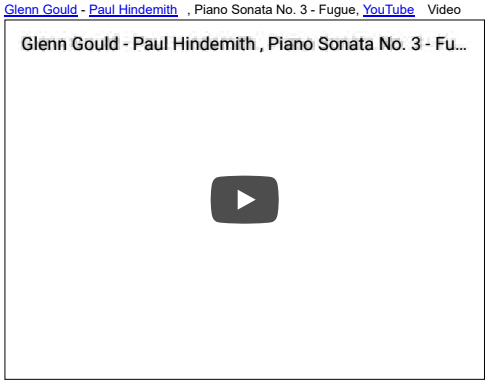
- Color Flipping
- Quad-Bitboards

Forum Posts

- Bit Board Bonkers?? by Dave, rec.games.chess.computer, July 28, 1997
- Bitboard board representation by Eric Oldre, CCC, January 13, 2005
- BitBoard representations of the board by Uri Blass, CCC, October 14, 2007
- Decision concerning board representation by Piotr Lopusiewicz, CCC, May 05, 2013

External Links

- Glenn Gould - Paul Hindemith , Piano Sonata No. 3 - Fugue, YouTube Video



References

1. ^ Bitwise Optimization in Java: Bitfields, Bitboards, and Beyond by Glen Pepicelli, 2005, O'Reilly's OnJava.com
2. ^ On Beowulf - long post by Vincent Diepeveen, CCC, April 04, 2001

What links here?

Page	Date Edited
AlphaZero	Feb 10, 2018
Array	Dec 1, 2016
Bitboard Board-Definition	Jun 23, 2014
Bitboards	Nov 14, 2017
Checks and Pinned Pieces (Bitboards)	Aug 14, 2013
Chess 0.5	Nov 20, 2016
Color Flipping	May 17, 2017
Counter	Nov 13, 2017
Demolito	Mar 1, 2018
Efficient Generation of Sliding Piece Attacks	Nov 5, 2016
General Setwise Operations	Feb 25, 2018
Incremental Updates	Sep 6, 2017
Initial Position	Jan 18, 2018
King Pattern	Nov 15, 2013

Page	Date Edited
Knight Pattern	Feb 23, 2015
Lachex	Jan 7, 2016
Make Move	Mar 2, 2016
Material Tables	May 5, 2017
Occupancy	Sep 19, 2016
Pieces	Feb 19, 2018
Quad-Bitboards	Jan 30, 2017
Rodent	Jan 11, 2018
Rotated Bitboards	Mar 7, 2017
RuyDos	Feb 17, 2018
SEE - The Swap Algorithm	Jun 5, 2017
Simona Tancig	Nov 7, 2012
Sliding Piece Attacks	May 27, 2016
Spector	Nov 11, 2016
Square Attacked By	Jan 20, 2018
Sungorus	Apr 11, 2014
Teki	Mar 29, 2018
Tunguska	Sep 16, 2017
WyldChess	Mar 10, 2018
X-ray Attacks (Bitboards)	Mar 31, 2015

Up one Level

(https://www.wikispaces.com/user/view/Bretwa)

One piece, one bitboard ?
Bretwa (https://www.wikispaces.com/user/view/Bretwa) Aug 7, 2012

Hello,

I have a question about piece representation with bitboard. Actually it seems that the approach, one kind of piece for each color <=> one bitboard is very common on Internet. But this approach implies that a "nextbitset" call has to be done, which can require quite a lot of cpu time. It's also require a mask to eliminate the other piece on the bitboard. If we use one bitboard for a piece (exception for pawns) we don't have to do this. For example we would have a bitboard for the left white rook, another bitboard for the right white rook. And we create special treatment for pawn promotion. Is this relevant for a faster moves generation ? Thanks beforehand.

Bretwa


(https://www.wikispaces.com/user/view/GerdIsenberg)

GerdIsenberg (https://www.wikispaces.com/user/view/GerdIsenberg) *Aug 7, 2012*
Hi Betwa,
what you propose is the purpose of piece lists with square indices given for each individual piece. Keeping single populated bitboards, despite low data density, still requires a bitscan. With bitboards you like to intersect for instance all white rooks with rook attacks of the black king, to determine whether a rook gives check.

Also, the inner loop over target squares takes much more time in move generation than the outer loop over source squares of a rook, bishop or knight.

Gerd

(https://www.wikispaces.com/user/view/Bretwa)

Bretwa (https://www.wikispaces.com/user/view/Bretwa) *Aug 7, 2012*
Hi Gerd,

I'm sorry but I do not understand why we still need a bitscan with single populated bitboards. For example if we generate white rooks moves in pseudocode we have :

if(white_left_rook != 0) *optional*
{
long left_rook_mask = rook_mask_list[white_left_rook]; a hashmap of course
long left_bitboard_with_mask = left_rook_mask AND current_bitboard;
long white_left_rook_moves = rook_moves_list[left_bitboard_with_mask];
}

We do same for right rook and a special case for pawn subpromotion with another bitboard. If we use traditional two rooks bitboard we have something like :

int first_rook_index = get_next_bit_index(white_rook_bitboard, 0);
long first_rook_mask = rook_mask_list[first_rook_index];
long first_rook_bitboard_with_mask = first_rook_mask AND current_bitboard;
long white_first_rook_moves = rook_moves_list[first_rook_bitboard_with_mask];

int last_rook_index = get_next_bit_index(white_rook_bitboard, first_rook_index + 1);
long last_rook_mask = rook_mask_list[last_rook_index];
long last_rook_bitboard_with_mask = last_rook_mask AND current_bitboard;
long white_last_rook_moves = rook_moves_list[last_rook_bitboard_with_mask];

Of course we can create a buckle for pawn subpromotion. To my point of view it appears that we don't need a bitscan with the first approach. Concerning the
check topic, we just have to make OR between left and right rooks bitboards, this seems to be a little drawback regarding the gain of not using

get_next_bit_index. Where do you think is the mistake ? Hashmap could have a strong negative impact ?

Bretwa


(https://www.wikispaces.com/user/view/GerdIsenberg)

GerdIsenberg (https://www.wikispaces.com/user/view/GerdIsenberg) *Aug 7, 2012*
Sure you can use a hashmap, but hashing the single populated bitboard for a dense index is exactly what the bitscan is about - minimal perfect hashing for instance ala DeBruijn multiplication. It seems cheaper to me instead of keeping single populated bitboards, to keep and incrementally maintain (make, unmake) the logarithm dualis of the power of two value inside a piece list, that is the square index itself.

Gerd

(https://www.wikispaces.com/user/view/Bretwa) Bretwa (https://www.wikispaces.com/user/view/Bretwa) *Aug 8, 2012*

Ok Gerd, but I thought to another advantage of using single populated bitboards. It's about the evaluation function. If we use several pieces on a single bitboard, we will have to compute the hamming weight of this bitboard, then multiply by the piece value. According to Wikipedia, this need 4/5 operations with the many 0 method. If we use single populated bitboards, we just have to check if the bitboard is not null and add the piece value to the total evaluation. The evaluation function is called many times, so this could be a great advantage, no ?

Bretwa

(https://www.wikispaces.com/user/view/GerdIsenberg) GerdIsenberg (https://www.wikispaces.com/user/view/GerdIsenberg) *Aug 9, 2012*

I fear not. Material is calculated by incremental update during make/unmake anyway, i.e. only updated after captures/promotions. You don't use popcount for that, except perhaps in initialization. Keeping explicit single populated bitboards inside a piece-list as board representation makes no sense.

Gerd

(https://www.wikispaces.com/user/view/Bretwa) Bretwa (https://www.wikispaces.com/user/view/Bretwa) *Aug 9, 2012*

Ok thanks for this advice. Calculating material during the tree construction should be faster, even if the type of the captured piece has to be known explicilty.

Bretwa