Home * Board Representation * Bitboards * **Design Principles**

By Steffan Westcott [1]:

I should add that these techniques (Kogge-Stone) are designed to take advantage of the parallel nature of bitboards, in that they consider the entire board. Here, routines like RookMovesUp() will calculate the upward rook moves of all friendly rooks.

In general, I first identify a (bit) pattern of interest, then devise methods for recognising all instances of that pattern on the board. Pattern instances are counted as late as possible, if at all. The complexity of the patterns varies greatly. Simple ones are like OpenFiles(), UnmovedRooks(), PawnAttacks(), PawnRams(), PawnDuos(), KingIsUpRight(). Medium complexity are ones like ConnectedRooks(), RooksCanCastle(), OnKingDiagonal(), NearKingDiagonal(), OutPost(). Complex examples are Fortress(), PawnMass(), PawnStorm(), BackwardPawns(), MaterialSignature(), WeakSquareControl(), StrongSquareControl(), SpaceBehindPawnFront(), StrongKnightOutposts(), StrongFianchettoedBishops(), WeakWhiteSquares(), KingShelter(), GamePhase().

Often the more complex patterns are combinations of the simpler ones. In fact, the chess position itself can be viewed as composed of 'primitive' or 'atomic' patterns (bitboards). Most of the simpler patterns are returned as bitboards, where set bits indicate a (bit) pattern match. This is fine where simple square-centric patterns are sought, and a yes/no for each square is sufficient.

Complex patterns like KingShelter() and GamePhase() are really functions which classify (group together) general patterns spread across the whole board, eg. KingShelter() classifies the pawn structure near the king (matches against a large pattern set), MaterialSignature() returns things like BNbn to classify the material balance.

Just considering the simple patterns, if your engine deals with pattern instances in a serial fashion (strictly one at a time), the algorithm requirements are sufficiently different that an alternative may be better eg. rotated bitboard table lookups, or perhaps even a different board representation altogether. To my mind, the major reason to use bitboards in the first place is to find many pattern instances quickly. In summary, I would advise that my algorithms are no magic bullet - They are better judged in the wider context of your engine design and target architecture.

Rererences

1. ^ Bitboard algorithm design principles by Steffan Westcott, CCC, October 23, 2002

What links here?

Page	Date Edited
Bitboards	Nov 14, 2017
Chunking	Jun 12, 2017
Design Principles	Jan 17, 2018
Kogge-Stone Algorithm	Sep 17, 2016
Pattern Recognition	Sep 8, 2017
Steffan Westcott	Jan 21, 2014

Up one Level