Beta I Writeup

Chengkai Zhang, Haoran Xu, Yinzhan Xu, Yuzhou Gu

November 18, 2016

1 Profiling Data

We tested the original program on the command ”go depth 8” and recorded the profiling. Below is the result for the most costly 6 function.

scout_search	pawnpin	h_squares_attackable	eval	square_of	make_move
14.59%	13.34%	9.71%	8.74%	6.18%	6.04%

2 The changes so far

1. We used a uint64_t to store the cells on board that are lasered. This replacement saves some scans of the whole board, and also saved some memory space. Also, in eval.c, the laser was computed several times on the same board; since we are using a bitmap, we can simply use a bitmap to store the laser and use this bitmap for all the computation. It gives about 20% speedup.

2. We also use a bitmap to store the cells on the board that are white pieces and another bitmap to store the cells that are black pieces (it is supplementary and the original representation is still stored). This helps to reduce some blind scan of the whole board. It gives about 15% speedup.

3. We change ARR_WIDTH from 16 to 10. This decreases ARR_SIZE from 256 to 100.

4. The pcentral function in eval.c repeats calculation a lot of times. We precompute the results and stores the result in a constant table.

5. We use constant table to remove many divisions in the code.

6. We changed small functions to inline functions or macros.

7. We found that it is not necessary to use int in some places. We replaced them with appropriate smaller types such as uint8_t and uint16_t.

8. In scoutsearch function, there is a incremental search function called. We found that this function is very slow so we decide to replace it with a more efficient sorting algorithm. First we tried quick sort but that does not help. Then we discover that mostly only the smallest several items will be used, so we decide to find the smallest element each time

with the iteration. Using bruteforce to find it does not make the code faster, so we instead maintained a range tree to maintain the smallest element in the array. It gives about 10% speedup.

9. subpv in .. is only used to store the best moves up the search depth, but we really only need the first move. Thus, we decide to delete the array and replace it with a variable. This saves the memory and thus improves the speed. It gives about 20% speedup.

10. We modified some logic in eval.c but keeping the result as the same as the original. For example, we merged the case for king and for pawn in the switch struct, and then minus score for king out of the loop. Such improvements give a 20% speedup.

3 Optimizations we plan to make

1. Parallelize the code. Chengkai Zhang and Haoran Xu peer programming.

2. Scout_search and make_move now are the most costly function, so we may be able to optimize them in more detail. Chengkai Zhang and Haoran Xu peer programming.

3. During beta1, we find that make searchNode smaller can give us noticeable speed up, so we can still to use this trick to improve, Yuzhou Gu and Yinzhan Xu peer programming.

4. We can also do some strategical change. Yuzhou Gu and Yinzhan Xu peer programming.