Algorithms:

In this program, we implemented the Minimax algorithm by considering the maximum number of flips each step can achieve. If the number of steps we need to think ahead is 0, then the one with the maximum number of flips is picked; if the number of steps we need to think ahead is 1, then we calculate the maximum number of flips in this step subtracted by the maximum number of flips in the next step and pick the maximum number. For arbitrary number of steps we need to think ahead (depth), we can do it iteratively followed by the procedure described above.

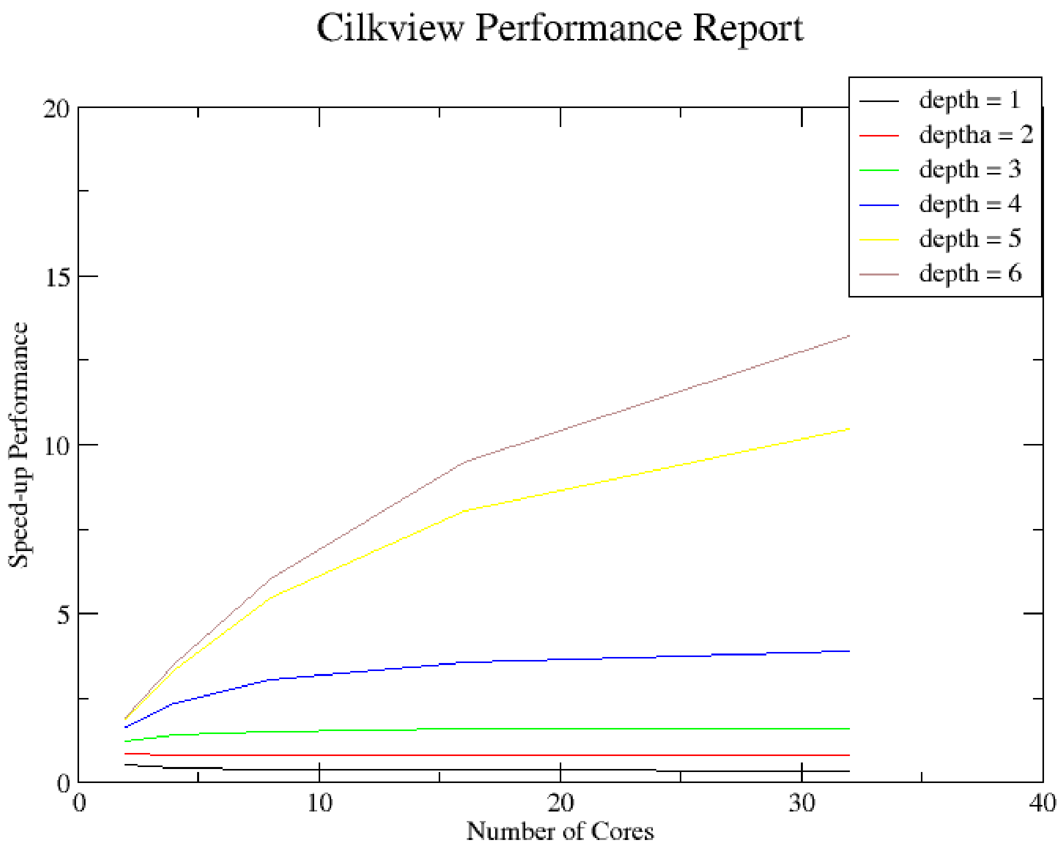
For each depth we evaluate 64 search of each move on the board, it is a loop. At each time, we only consider about the legal and acceptable move. If depth=0, we need to consider 64 moves; if depth=1, we need to consider 64^2 moves. As the depth goes further, the number of moves we need to consider about grows exponentially, which means the computer time would also grow exponentially. This is why we need to do parallelism.

Parallelism Scheme:

As described above, the most important time lag lies in the part of searching the board for the best move. As the depth increases, this search time will also increase exponentially. The implementation of parallelism here is on this part. I use the cilk_for instead of the for for that loop:

```
cilk_for (int row=8; row >=1; row--)
```

At each loop, the "CEnumerateLegalMove" function will iteratively call itself to perform the selection of best move. Also, the last step where we are in the outer loop is not selected as the part for parallelism. I do not consider the other part of possible parallelism since this loop can already occupy the 8 cores provided fully. Furthermore, other part for possible parallelism may not get an optimistic result, and may add up to more overheads. At each end of the cilk_for function, the synchronization is implicitly implemented. The experimental results with the Cilkview are as follows:



We can see that as the depth increases, the speed-up performance gets better, which means the parallelism scheme is favored by larger depth.

When the search depth is different, the algorithm implemented will utilize the parallelism with different percentage. We can see from this result that the depth = 6 has larger speed up estimate than depth = 1.

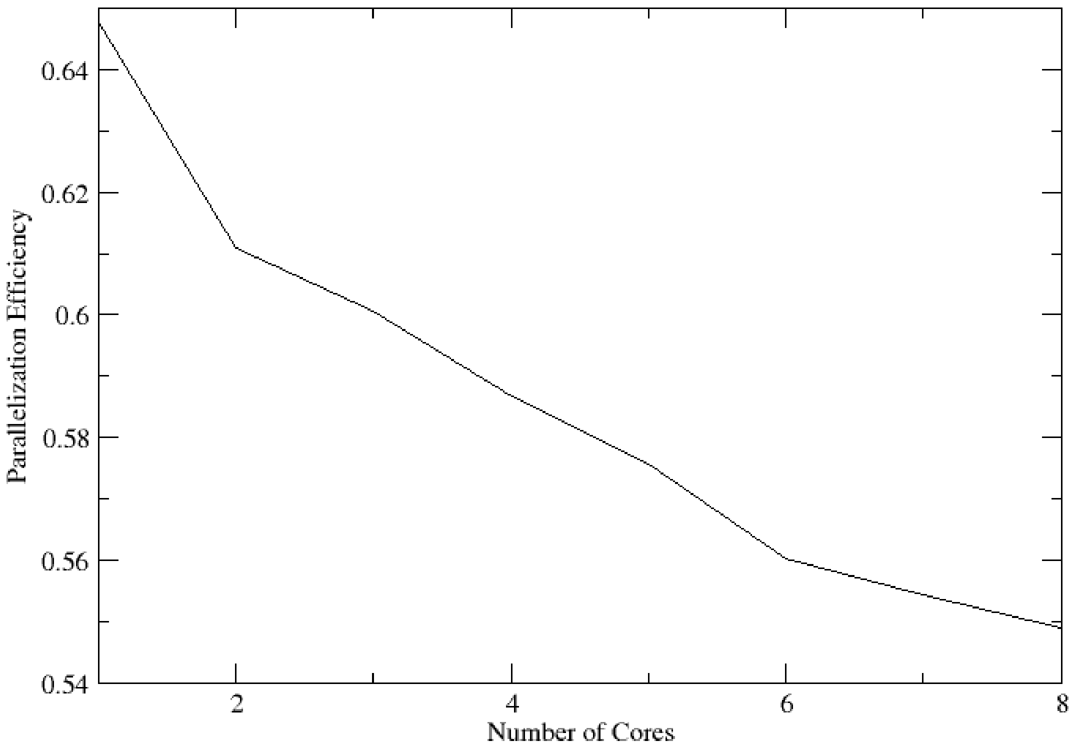
For the depth = 1 to 4, we can see that the cilkview does not show a linear speed up over 1 processor version. However, when depth = 5 or 6, the linear speed-up is obvious.

According to the later experiments, as shown in the following plot, we can see that the experiments in reality achieve similar results, with a relatively stable efficiency, around 0.60. So the practical results also show an approximately linear speed-up with depth = 6.

The parallelism seems to be good for until 8 cores. We can invoke more cores if we want, however, the biggest number of parallelism in cilk_for is 8, since the row=8, so we needn't more than 8 cores. We can do the parallelism for the inner loop, hence resulting in 64 threads available for the parallelism. In that way, we can use more cores to create a better parallelism.

In this parallelism, I do not consider the inner loop in order to avoid more overheads, this will sacrifice the parallelism to some extent.

The Efficiency Plot is shown below:



From the above plot we can see that the parallelization Efficiency drops steadily with the increase of number of cores, but overall the efficiency is maintained around 0.6. This result indicates that the speed-up performance would grow linearly with the increase of number of cores as the parallelization efficiency is relative stable.

Result with Cilkview

depth = 1:
Cilkview Scalability Analyzer V1.1.0, Build 8503
1) Parallelism Profile
Cilkview Scalability Analyzer V1.1.0, Build 8503
1) Parallelism Profile
Work : 14,595,804 instructions
Span : 8,032,171 instructions
Burdened span : 24,833,865 instructions
Parallelism : 1.82
Burdened parallelism : 0.59
Number of spawns/syncs: 2,870
Average instructions / strand : 1,695
Strands along span : 2,461
Average instructions / strand on span : 3,263
Total number of atomic instructions : 28
Frame count : 108592
2) Speedup Estimate
2 processors: 0.51 - 1.82
4 processors: 0.41 - 1.82
8 processors: 0.38 - 1.82
16 processors: 0.36 - 1.82
32 processors: 0.35 - 1.82
depth = 2:
Cilkview Scalability Analyzer V1.1.0, Build 8503
1) Parallelism Profile
Work : 73,320,550 instructions
Span : 13,597,359 instructions
Burdened span : 55,832,617 instructions
Parallelism : 5.39
Burdened parallelism : 1.31
Number of spawns/syncs: 16,562
Average instructions / strand : 1,475
Strands along span : 6,529
Average instructions / strand on span : 2,082
Total number of atomic instructions : 28
Frame count : 729752
2) Speedup Estimate
2 processors: 0.87 - 2.00
4 processors: 0.82 - 4.00
8 processors: 0.80 - 5.39
16 processors: 0.78 - 5.39
32 processors: 0.78 - 5.39
depth = 3:
Cilkview Scalability Analyzer V1.1.0, Build 8503

1) Parallelism Profile
Work : 371,373,567 instructions
Span : 24,174,908 instructions
Burdened span : 132,749,247 instructions
Parallelism : 15.36
Burdened parallelism : 2.80
Number of spawns/syncs: 106,883
Average instructions / strand : 1,158
Strands along span : 17,815
Average instructions / strand on span : 1,356
Total number of atomic instructions : 28
Frame count : 3823629
2) Speedup Estimate
2 processors: 1.24 - 2.00
4 processors: 1.42 - 4.00
8 processors: 1.52 - 8.00
16 processors: 1.58 - 15.36
32 processors: 1.61 - 15.36
depth = 4:
Cilkview Scalability Analyzer V1.1.0, Build 8503
1) Parallelism Profile
Work : 4,167,802,909 instructions
Span : 95,867,245 instructions
Burdened span : 572,647,406 instructions
Parallelism : 43.47
Burdened parallelism : 7.28
Number of spawns/syncs: 1,037,386
Average instructions / strand : 1,339
Strands along span : 74,203
Average instructions / strand on span : 1,291
Total number of atomic instructions : 30
Frame count : 43760012
2) Speedup Estimate
2 processors: 1.62 - 2.00
4 processors: 2.35 - 4.00
8 processors: 3.04 - 8.00
16 processors: 3.55 - 16.00
32 processors: 3.88 - 32.00
depth = 5:
Cilkview Scalability Analyzer V1.1.0, Build 8503
1) Parallelism Profile
Work : 26,216,874,413 instructions
Span : 159,913,564 instructions
Burdened span : 1,023,539,376 instructions
Parallelism : 163.94
Burdened parallelism : 25.61

Number of spawns/syncs: 7,317,492
Average instructions / strand : 1,194
Strands along span : 135,469
Average instructions / strand on span : 1,180
Total number of atomic instructions : 30
Frame count : 270710432
2) Speedup Estimate
2 processors: 1.88 - 2.00
4 processors: 3.34 - 4.00
8 processors: 5.46 - 8.00
16 processors: 8.02 - 16.00
32 processors: 10.47 - 32.00
depth = 6:
Cilkview Scalability Analyzer V1.1.0, Build 8503
1) Parallelism Profile
Work : 298,731,501,091 instructions
Span : 1,306,279,861 instructions
Burdened span : 8,070,317,401 instructions
Parallelism : 228.69
Burdened parallelism : 37.02
Number of spawns/syncs: 85,436,463
Average instructions / strand : 1,165
Strands along span : 1,024,813
Average instructions / strand on span : 1,274
Total number of atomic instructions : 30
Frame count : 3075689861
2) Speedup Estimate
2 processors: 1.90 - 2.00
4 processors: 3.52 - 4.00
8 processors: 6.05 - 8.00
16 processors: 9.47 - 16.00
32 processors: 13.20 – 32.00