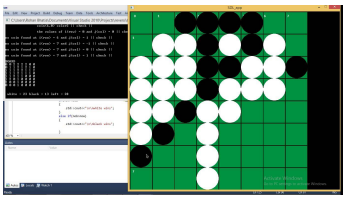
CS3211 project 2 - OthelloX

March 3, 2017

Due 5pm on Thursday April 13, and worth 30% of your course mark.

Project 2: Develop an openMPI parallelized program to assess the best choice for a next move in the new game OthelloX, given a current board position. A likely technique would be to use the minimax algorithm, given a specific board evaluator. The emphasis is not so much on your algorithm or its results, but instead on your focus on parallelization. You will outline your techniques used, and contribution of each technique with respect to the speedup results you achieved. You will reflect on the behaviour of your program as the problem scales. We expect this write-up to take no more than 10-20 pages.

1 The problem



In project 2, we want you to develop an openMPI parallelized program to assess the best choice for a next move in OthelloX, given a current board position. OthelloX (pronounced like ocelots) is the game Othello, played on an $m \times n$ board instead of an 8×8 square. A likely algorithmic technique would be to use the minimax algorithm, given a specific board evaluator, alpha-beta pruning and so on. You can see a description of Othello here:

```
https://en.wikipedia.org/wiki/Computer_Othello
```

And you can find some description of possible evaluation techniques here:

```
http://radagast.se/othello/howto.html
```

Your program should accept an input board file, an evaluation parameter file, and should return a set of best next moves along with information about the number of boards assessed, depth of moves, and a boolean indicating if your program covered the entire search space. A possible example run might look like this:

```
-bash-4.2$ mpirun  ./othellox initialbrd.txt evalparams.txt -np 100
   Best moves: { d3,c4,e6,f5 }
   Number of boards assessed: 1234567
   Depth of boards: 6
   Entire space: false
   Elapsed time in seconds: 212.3
   ....: ....
-bash-4.2$
```

In this example, the `othellox` program is given 100 processes, and evaluates (in this case) the four best moves for the dark player as the options (d,3), (c,4), (e,6) and (f,5). Note that with any given board there may be multiple moves that (in the end) are the best move. The starting position is found in the file `initialbrd.txt`, and some evaluation parameters are found in `evalparams.txt`. When the program finishes, it also reports various parameters, including the minimal set given above.

The board might be `brd[a..h,1..8]`. The input board file format for specifying boards and positions is as follows:

```
-bash-4.2$ cat initialbrd.txt
   Size: 8,8             # x,y
   White: { d4,e5 }      # This is the standard starting position
   Black: { d5,e4 }      # for Othello, but you can set any position
-bash-4.2$
```

The evaluation parameter file format format will be dependant on your choices of board evaluation parameters, but will follow the same basic format as follows:

```
-bash-4.2$ cat evalparams.txt
   MaxDepth: 10          # Maximum depth of following moves
   MaxBoards: 10000000   # maximum number of boards to evaluate
   CornerValue: 42       # Parameter for the board evaluator
   EdgeValue: 4          # Parameter for the board evaluator
   ...: ...
-bash-4.2$
```

In this example, the first two lines give limits on the overall evaluation strategy. Other parameters are used to "tune" the board evaluator.

You might find it easier to start by coming up with a sequential board evaluator for OthelloX, so you can have a benchmark against which you can test the correctness of your parallel version.

1.1 Your task

Your task is initally to implement `othellox`, and then to explore the scalability of the (multi-dimensional) problem space, and the results with your parallelized implementation. Note that you have various ways of expanding the problem size - for example the size of the board could be changed, and the depth of the search could be increased. We expect that you could search the complete space for 6×6 boards and perhaps even 8×10, but 26×26 may be intractable.

Your writeup should have components that report on any techniques you used to improve your code that had a significant effect (board storage technique for example), along with any quantitative evaluation you did for the code technique. You should describe what you did to address the various challenges of parallel computation (for example aggregation, load balancing, communication/computation). You should tabulate and chart the actual results you get when changing the size(s) of the problem, and reflect on the reasons for the results that you see.

2 Assessment

The assessment below is only a guideline, and is indicative of the project assessment. However, the assessment for individual projects may deviate from this in some ways, dependant on the form of the delivered project. On **Thursday 13th April 17:00**, the project is due. It will be worth **30/100** of your final mark. The assessment will be done as follows:

- **8/30** Implementation: An assessment of the implementation and associated code quality, and code documentation.
- **10/30** Discussion: An assessment of the discussion of the techniques you used to improve your code, along with justfications.
- **12/30** Analysis: An assessment of your writeup with respect to the analysis, which should include clear tabulations and graphs of the behaviour of your parallel system, and clear reflection and analysis of the results obtained.

In general, better assessments will be given to writeups which encapsulate more complex ideas.

2.1 Finally...

Your final completed project sources should be uploaded on or before Thursday 13th April 17:00, along with a readable **hardcopy** delivered to Hugh.

You are allowed to discuss the problems with your friends, and to study any background material with them, but the project *should be your own work*. **Copying** and **cheating** will be grounds for failing the project.