Exploring Reversi Endgames via Parallelization

Devanshu Pandey
University at Buffalo
CSE633-Parallel Computing
December 2, 2010
Reversi and the endgame

- Demo
- Branching factor – 4 – (Please note that this is something I erroneously picked for my project based on scarce empirical observation). You can find an average of 6-7 moves at each point.
  - Important to my implementation
- Initial central idea – Vary the size of the endgame
- Later implemented Monte Carlo Tree Search
- Parallelizing alpha-beta pruning too difficult
  - Parallel Variant Splitting
Implementation

- Python
- MPI work
  - mpi4py
  - MPICH II
- Why?
  - Python’s ease of use
  - Testing MPICH II
The Board

- An 8 x 8 array
- State based implementation
  - Initializing states
  - e_count
- The possible moves function
  - Inefficient?
The parallel algorithm

- Master-Slave implementation
- Root starts the game, initializes the board state
- Calls the possible moves function.
- An average of 4 possible moves, one move per processor
Tasks for every slave PE

- Monte Carlo Tree Search
  - Play random moves till game ends.
- Store the win ratio (\# wins/ \# times move was played) in a transposition table
  - Transposition table:
    - An array of length 64
    - For position \((r, c)\), index = \(8r + c\)
- In the end, return the transposition tables to the master
- “Zip” the tables together to get final values
Problems

- Full blown tree shaped architecture
  - My implementation – truncated
    - CCR problems
    - Bad implementation

- Code translation
  - Time hog
  - Led to inefficient time and space usage
Time (ms) v/s Endgame size

- Depth 5
- Depth 10
- Depth 15
- Complete game

- Serial Code
- 5 Processors
- 21 Processors
Win Ratio v/s Size of Endgame

- Serial Code
- 5 Processors
- 21 Processors

Depth 5 | Depth 10 | Depth 15 | Complete game

12/21/2010
Future work

- C implementation
- Achieve time and space efficiency
- MCTS on CUDA
References
