

MCS 572 Project One : the game of connect-4 with MPI

Message Passing for distributed game trees

The goal of the project is to use MPI to distribute a game tree to determine the next move of the computer in the game of four in a row, also called connect-4.

In the game of connect-4 two players have each a set of tokens. Each set has a different color: green and red. Without a graphical user interface, the colors green and red are replaced by the characters X and O. Identifying the players by the color of their tokens we say that green plays against red, or X against O. The players take turn in dropping a token in any of the not yet full slots of the vertical columns of a 7-by-7 board. After each drop the token ends up at the lowest empty square in that column on the board. The game is won by green (respectively red) as soon as four green (respectively red) tokens are aligned, either horizontally, vertically, diagonally, or along the antidiagonal. Players never run out of tokens, but the game ends in a tie when the board is full without four tokens of the same color aligned.

The user of the program plays against the computer. At the most basic level, the computer selects an open slot at random and is easy to beat. For a more interesting game, the computer evaluates the boards and calculates several moves ahead to make its move. Below we outline some heuristic rules for the computer to make its next move.

At level one, if the computer looks only one move ahead, there are seven possibilities (assuming all columns are not yet full). For each of the seven possible moves of the computer, a win of the computer is marked by +1, a win of the user by -1, and all other cases by 0. Note that at the first level, a win of the user cannot happen, but this first level is also the base case in the general recursion, when the current level has reached its maximum depth and the last move could have been a move by the user.

When the computer considers d moves ahead, the recursion starts at level k equal to zero and runs to d . At each level k , a board with the computer as winner receives value (+1) 7^{d-k} while a loss (a win of the user) is marked as (-1) 7^{d-k} . Doing so, moves in the near future get stronger positive or negative values than later moves. For boards where there is no winner, the recursion proceeds recursively to the next level up, provided the board is not yet full. Full boards evaluate to zero. The recursive evaluation returns the sum of the values of all boards in case $k > 0$. In case $k = 0$, before returning, the recursion computes the index of the column which yielded the maximum value. This index is then the computed move.

An example of a Python script implementing the game is posted as the solution of the second project of MCS 275 in Spring 2010. (available at <http://www.math.uic.edu/~jan/mcs275/projs.html>). C code for a game tree for connect-4 is at <http://www.math.uic.edu/~jan/mcs360s03/projects.html>. Getting a functional C program working is needed for your solution to the project to be relevant and useful. The coding essence of the project lies in the implementation of message passing in assignment one.

Assignment One: define the message passing

Write a C program to define the message passing for a program using MPI with 8 nodes in a static work load assignment. The root node assigns to the i th compute node to compute the value of dropping a token in slot i , for $i = 1, 2, \dots, 7$.

The root node sends the level to the node, i.e.: how many moves ahead must be computed. Compute node i sends an integer back to the root node, expressing the value of dropping a token in the i th column. In determining the next move of the computer, the root node selects the column with the highest valued received by the compute nodes.

Assignment Two: examine the scalability

As we ask the computer to look more moves ahead, with each increase of the level, the number of game boards is multiplied by 7. With 8 processors, we may hope for an optimal speedup, provided there is sufficient memory and the communication overhead is dominated by the computation time.

Determine experimentally the relation between the communication and the computation cost to determine the scalability of the program developed in the first assignment. In your cost considerations, take the memory usage into account because memory is the most likely bottleneck for a serial program.

Extrapolating on your experimental results for 8 processors, in your asymptotic analysis, you may assume the number of processors increases with a factor of 7.

Assignment Three: discuss load balancing

In some scenarios, computing the value of one move is immediate, whereas for other moves there are many more possibilities. Describe a situation where static work load assignment does not perform very well.

Develop an algorithm for dynamic load balancing to apply for any number of processors. Decide on the granularity of the size of each job and motivate your choice. Indicate the application of the dynamic load balancing code we have discussed in class to this problem.

The deadline is Monday 20 February 2012 at noon

On the deadline, bring to class the answers to the assignments on paper, describing your design decisions made while developing the code and the analysis of communication and computation costs. In addition, also send me by email the programs you wrote.

You may work individually or in pairs on this project. For each pair, please submit only one solution. If you have questions or difficulties with the assignments, feel free to come to my office for help.