Math-454 Parallel Computing and Pthreads
Solving n-queens puzzle on GPU

Anders Asheim Hennum

June 8th

Abstract

Using GPU for scientific computing has become very popular. For many problems, the massive parallelism that GPU offers, can give magnificent speedups compared to normal CPU's. In this paper I have investigated if GPU is usable for solving puzzles. TODO: Rewrite when project is done. Add summarizing result.

1 Introduction

The puzzle I will try to solve is the n-queens puzzle [1]. It is a simple and well documented puzzle. The puzzle is to place n queens on an $n \times n$ chess board without any queen attacking another queen. The approach I will use is a random first search. This approach relies on random searches and has very variable running times. By using GPU, we can run multiple searches in parallel and by then, have a greater chance of finding a solution faster than running multiple searches on CPU sequentially.

2 The Algorithm

There are several approaches to solve the n-queens puzzle. The method used here is based on a random first tree search. I chose this because the method is very general and can be applied to a large number of problems. It also very intuitive and easy to implement.

In short words the algorithm works by placing out a queen randomly in the first column. This becomes the "root" node. It continues with placing a queen in the next column and makes sure that the row and diagonal is free and that it has not been tried before. It does this for all columns (levels in the tree) until it reaches a dead end or finishes by reaching the bottom. If a dead end is reached (no next placements possible) it back tracks to a previous column where not all possibilities is tried and continues to search from there.

Algorithm 1 Solution search for K-queens puzzle

```
initialize $K, S, N, D, i, q, iter, max\_iter$
while ( $iter < max\_iter$ ) do
    $q \leftarrow$ rand()*$K$
    if ($D[q] == 0$ and $N[i][q] == 0$) then
        N[i][q] = 1
        if (diagonalsOK($q, i, S$) == 1) then
            $S[i] = q$
            $D[q] = 1$
            i++
            if ($i == K$) then
                break
            end if
        end if
    end if
    if ((sum($N[i]$) + sum($D$)) == $K$) then
        $D[S[i-1]] = 0$
        $S[i-1] = -1$
        for each $j$ in $N[i]$ do
            $j = 0$
        end for
        $i - -$
    end if
    $iter + +$
end while
```

3 Implementation on GPU

To implement the algorithm efficiently on GPU there is three things one need to consider: random number generation, memory management and optimal grid size.

3.1 Random number generation

The CUDA toolkit has a very nice library `curand` (reference) to generate random numbers. The documentation describes how to obtain highest quality parallel pseudorandom number generation. I general, every experiment is assigned with an unique seed (time) value. Within the experiment all threads is assigned with an unique id number (by thread id and block id) and this is used as sequence number. By this we are guaranteed that all threads will get different sequences with good statistical properties.

3.2 Memory management

3.3 Optimal grid size

4 Results

TODO: Add some nice graphs and tables with running times of CPU vs GPU.

5 Conclusion

TOOD: CPU vs GPU best when solving puzzles with state-space search approach? Advantages, disadvantages? What needs to be done to solve this kind of problems on GPU's?

References

[1] `http://en.wikipedia.org/wiki/Eight_queens_puzzle`