实验报告

软硕161 徐毅
2016213585

一、研究背景

本次实验选题是著名的“八皇后问题”[1]。问题的描述是：给定一个8*8的国际象棋棋盘，最多能放置多少个皇后使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。这个问题是在1848年由Max Bezze提出，1850年Franz Nauck求出了部分解，之后包括像高斯这样的大数学家都在研究。如果采用暴力的方法需要遍历C(64, 8) = 4,426,165,368种可能性，实际不可行，而且最终结果只有92个解。一种优化方法是，因为每一行只会出现一个皇后，那只需要考虑8^8=16,777,216种情况，进一步来看，每一列也只会出现一个皇后，因此只需要考虑8!=40,320中情况。随着计算机的出现，这40,320种情况可以很容易的遍历求解。

上述问题可以做更深入的讨论，如果将棋盘绕某一个轴进行对称变换或者旋转变换，那么可以认为这些新产生的摆法和原来的是同一种，那在原来的基础上求一共有多少种独立的摆法，对于“八皇后问题”独立解的个数是12个。进一步来拓展八皇后问题，如果给定一个n*n的棋盘，那么最多有多少种摆法以及它们之中一共有多少种独立的摆法。

二、相关工作

现在学术界的研究的热点问题在于对于“n皇后问题”，如何快速求解个数。到1999年为止，学术界求出了在n<=23的情况下，所有解和独立解的个数。Matthias R. Engelhardt等人 [2]在2007年提出一种基于“群论”[3]的优化方法，不需要枚举n!种可能性，能够做到在可以接收的时间范围内计算n=30的解个数。Zsuzsanna Szaniszlo 等人[4]在2009年对判断一个摆法是否合法提出了优化方法，基本思想是将一个摆法看做是一个特殊的矩阵，利用矩阵的一些性质来判断，提高了计算的效率。

三、实现思路

3.1 串行实现

最基本的想法是给定一行的所有位置，依次试着放一个皇后，每放完后检查是否满足要求，如果满足要求，考虑下一行皇后的位置；如果不满足要求，考虑这一行中的下一个位置。依次进行下去，具体编程实现的过程中可以采用递归+回溯的方法实现。

```
int chessboardSize;// 棋盘大小
int[] chessboard;
long count;// 解的个数

// 针对第level行，遍历这一行所有的可能摆放的位置
public void caculateQueensWithoutParallel(int level){
    for(int pos = 0; pos < chessboardSize; pos++){
        if(checkVaildPostition(chessboard, level, pos)){
            chessboard[level] = pos;
            if(level == chessboardSize - 1){
                count++;
                chessboard[level] = -1;
                return;
            }
            caculateQueensWithoutParallel(level+1);
            chessboard[level] = -1;
        }
    }
}
// 判断在第level行，第pos列加入一个新的皇后是否合理
public  boolean checkVaildPostition(int[] board, int level, int pos){
    for(int i = 0; i < level; i++){
        if((board[i] == pos) || (i - level == board[i] - pos) || (i - level == pos - board[i])) return false;
    }
    return true;
}
// 程序入口，从第一行皇后的位置开始遍历
caculateQueensWithoutParallel(0);
```

函数caculateQueensWithoutParallel从第一行的所有皇后的位置开始遍历，直到level == chessboardSize - 1才算是找到了一个可行的摆法。

3.2 并行实现

串行方法方法里面最开始需要遍历第一行中所有n个可能的位置，在设计并行方法的时候，可以将第一步转化为n个子任务，相当于每一个任务在一开始已经确定了第一行皇后摆放的位置，要计算的是剩下n-1行中可能出现的摆法的个数，每个子任务的计算规模是n-1。因为这些任务之间互不影响，因此可以做到并行化。理想情况下并行程序计算n*n棋盘解的个数的时间应该等于串行计算(n-1)*(n-1)棋盘解的时间。

四、实验

4.1 实验环境

OS：Ubuntu 14.04

CPU：Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz * 8

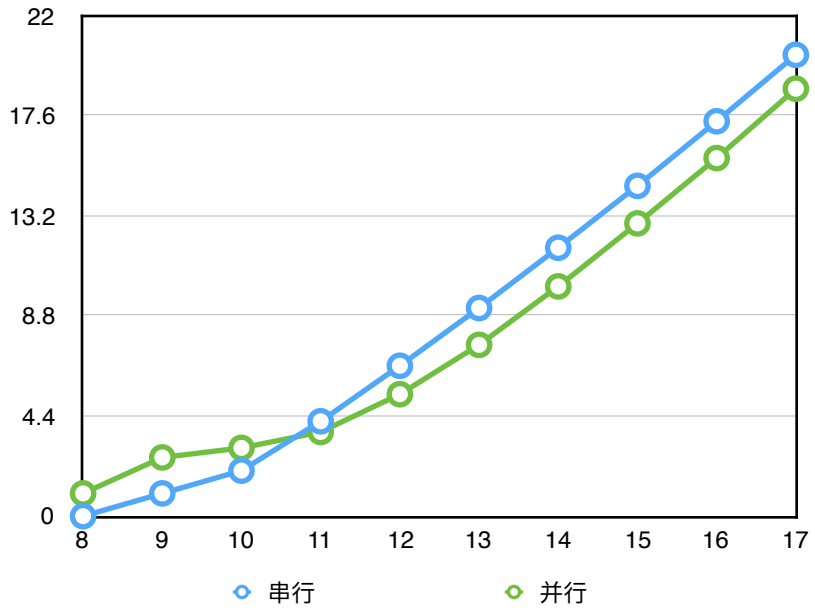
Memory：32GB

4.2 串行与并行结果对比

对于“n皇后问题”，n的规模从8到17，本次实验采集了串行方法和并行方法所需要的时间(单位：毫秒)，其中并行方法中使用了最大并行度为16的线程池。

	8	9	10	11	12	13	14	15	16	17
串行	1	2	4	18	98	569	3556	23553	168267	1269343
并行	2	6	8	13	41	185	1096	7485	54742	353520

针对采集到的结果，以2为底取了对数绘制下面的这张图。



从图中可以看出来随着问题规模的扩大，并行方法的计算时间明显小于串行方法，在n=17的情况下，串行方法花费的时间是1269343ms，并行方法是454520ms，前者是后者的2.79倍。虽然起到了一定的加速作用，但是没有拉开明显的差距，主要原因还在于计算主要花费的时间在判断每一种枚举出来的摆法是否合理上面。

对于n*n的棋盘，可以看做是n个(n-1)*(n-1)棋盘的子任务，理论上可以将原任务拆分n个子任务。那么理论上并行计算n*n的棋盘所花费的时间是串行计算(n-1)*(n-1)棋盘的时间。实际编写并行程序的时候，会给一个任务分配1-2个(n-1)*(n-1)棋盘的计算量，因此实际并行计算n*n的棋盘所花费的时间是串行计算(n-1)*(n-1)棋盘的时间的两倍，实验数据也很好的体现了这一点，符合理论的预期。

五、扩展问题

上面的实验分析只是针对所有可能的解，如果考虑到对称性和旋转，有些解就可能重复了。因此如果需要求独立解的个数，就要去掉一些重复的情况。首先对于这些重复的情况，给出一个关于同构的定义：

如果一种摆法可以通过若干次对称和旋转变成另外一种摆法，那么称这两种摆法是同构的

讨论对于给定的一个解，它可以通过多次对称和旋转产生不同的解。旋转可以有转0，90，180，270四种情况，对称可以按不对称，水平方向对称，垂直方向对称，先水平后垂直四种情况，出掉明显的重复情况，一共最多出现8种同构体。

在编写串行程序的时候，每得到一种可行的摆法，需要和已有的所有摆法进行比较，确定这种摆法和已有的摆法不是同构的。这一步操作会非常的费时。

在设计并行程序的时候，针对第一行皇后摆放的位置进行了优化。因为可以进行对称操作，所有第一行的皇后摆在(n/2+2) - (n)的位置所产生的摆法都可以由摆在1-(n/2+1)的位置通过对称得到，因此可以将问题的规模缩小一半，考虑第一行的皇后摆在1-(n/2+1)的位置时所有独立解的个数即可。可以讲原始任务拆成n/2+1个子任务，然后对于每个任务得到的结果，需要两两合并，去除同构解。这里的合并去重操作成为了性能的瓶颈。主要原因是，对于一个解的集合S和一个独立解p，如何快速判断p和S中的每一个元素都不同构，现在的方式是基于遍历的方式，效率很低，未来需要研究一种高效的比较办法。

六、总结

本次实验针对“n皇后问题”完全解和独立解问题，设计了串行计算和并行计算方法，比较了两者之间的性能。通过实验比较可以发现，虽然并行方法可以将原问题转变成若干个子问题，但是加速效果仍然不是非常的理想，主要原因有两个，第一个原因是拆分完的子问题仍然需要大量的计算时间。在本次实验里面，当n比较大的时候，计算n或n-1规模的棋盘所需要的时间都很长，并行方法并没有真正起到加速的效果，因此设计串行方法的时候要足够高效。现在的一种想法是对于一个串行计算的任务再将其并行化，但是这样带来的问题是，普通的计算机可能没有这么多的计算资源，并行的结果可能会退化为串行。除此之外，还有更重要的原因，当两个子任务做完之后，需要将结果合并。对于完全解的问题，合并可能只是简单的将结果个数相加，但是对于独立解问题，需要去除重复，这一步往往会很耗时。合并操作十分频繁的话，并行的效果就会打折扣，这里需要做权衡。

从串行到并行的转化过程中，需要合理分配计算资源。最理想的情况是，要计算的数据相互之间完全独立，这样就可以并行最大化。但实际问题中，往往数据之间有一定的关系，计算到最后需要有一个合并的过程，合并的过程往往成为计算的瓶颈。因此，在以后编写并行程序的时

候，要特别注意是否需要将子任务的结果合并，如果需要，需要提前想好一个高效的合并方法，否则实际运行的结果往往和预期会有很大的差距。

参考文献

1. Eight queens puzzle. https://en.wikipedia.org/wiki/Eight_queens_puzzle
2. Engelhardt M R. A group-based search for solutions of the n-queens problem[J]. Discrete Mathematics, 2007, 307(21): 2535-2551..
3. Group theory. https://en.wikipedia.org/wiki/Group_theory
4. Szaniszlo Z, Tomova M, Wyels C. The N-queens Problem on a symmetric Toeplitz matrix[J]. Discrete Mathematics, 2009, 309(4): 969-974.
5. https://oeis.org/A002562
6. https://oeis.org/A000170