INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

TOWERS OF HANOI-AN ITERATIVE SOLUTION FOR PARALLEL COMPUTATION

Bhaskar Kumar Mishra[1*], Shubham Vishnoi[2]
[1*2]School of Computing Sciences and Engineering,Vellore Institute of Technology Chennai, India.
Correspondence Author*:* bhaskar.kumar2011@vit.ac.in

Keywords: Algorithm, Iterative methods, Parallel computing, Recursive calls, Space complexity, Time complexity, Towers of Hanoi

Abstract

This paper deals with a new efficient methodology to solve the Towers of Hanoi in iterative way for parallel computation. The algorithm presented in this paper identifies the disc to be moved and the movement to be performed on each step independently, and hence each step is independent of the previous steps, for example- For 3 discs to be moved from peg A to peg C using peg B maintaining the property of discs to be aligned in ascending order on each peg, the earlier methodologies will calculate the first movement to be carried out first then the second and so on till the last step, however this algorithm can calculate the first, second, third or any step irrespective of the earlier steps.
Thus, the algorithm presented in this paper has an extra advantage that it can be implemented through parallel computation which the earlier methodology failed to provide. Thus using this algorithm can reduce the execution time of Towers of Hanoi problem considerably. Also space complexity of this algorithm is not much as compared to previous methodologies.

Introduction

The problem consists of three rods or pegs namely A, B and C and a number of disks of different sizes which can slide onto any peg. Initially, all the disks are placed as a stack on one peg, say A (Fig 1.1) in ascending order of size (the smallest at the top, the largest at the bottom).
The objective is to move the entire stack to another rod, say C conforming to the following rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.[1]

With three disks, the puzzle can be solved in seven moves as depicted in Fig 1.2. The minimum number of moves.

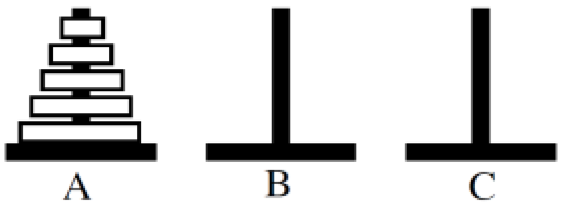

Fig 1.1- Towers of Hanoi

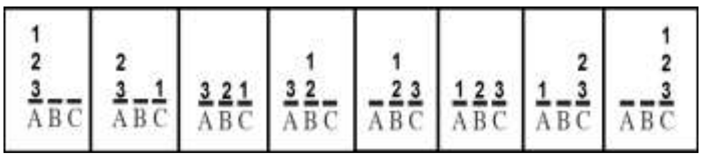

Fig 1.2 – Solution for 3 discs

required to solve a Tower of Hanoi problem is $2^n - 1$, where n is the number of disks involved. [1]
Since the prediction of a recursive solution to this problem, many attempts have been made to find a more optimized solution to the problem involving techniques like dynamic programming, iterative solutions etc. Various iterative solutions have been suggested by authors. The main idea behind all these iterative algorithms is that the disk transfer can occur in two directions using cyclic moves. For an even number of discs n, the direction of odd-numbered disks is 1→2→3→1, and for even-numbered disks is 1→3→2→1. Reversely for an odd n the directions are reversed i.e. the direction for even-numbered disks is 1→2→3→1, and for odd-numbered disks is 1→3→2→1.

In this paper, a simple version of an iterative algorithm for solution of tower of Hanoi is presented. The solution is based on the same idea used for all the iterative algorithms suggested so far i.e. cyclically moving the disks in both directions. The algorithm suggested in this paper first finds out the disk involved in the movement, i.e.at each step, it finds out the disk number to be displaced and afterwards, find out the movement to be performed i.e. the peg from which to move the disk and the peg on which the disk is to be placed.
Technically the storage complexity of this algorithm is O(1) and time complexity is O(2^n). However, each step of this algorithm can be independently computed i.e. the iterations of the loop (to find the disk number to be moved and the movement to be performed) are independent of each other and hence can be computed independently. Thus by using parallel computation, the time complexity can be reduced to a great extent. For example by using 4 CPU cores to find the solution, the running time would be reduced to 4 times the normal running time. In general, by using n cores to compute the solution, the running time would be reduced by a factor of n.

Recursive algorithm

The recursive solution to the Towers of Hanoi problem is understandable by the following simple observation:
Moving n disks from peg A to peg C is a problem consisting of the sub-problem of moving n-1 discs to peg B first, then moving 1 disc to peg C and then again solving the sub-problem of moving n-1 disks to peg C. Thus if we can find the solution of moving n-1 disks to peg B first and later to peg C, then we can find the solution to n disks. And hence the pseudo-code is depicted as under:

```
void towers(n, from_peg, to_peg, aux_peg)begin
    if n==1 then begin
        print "from_peg→to_peg"
    endif

   else if n>1 then begin
        towers(n-1,frompeg,auxpeg,topeg)
        towers(1,frompeg, topeg, auxpeg)
        towers(n-1,auxpeg,topeg,frompeg)
    endif
end
```

Hence for n>1, the first recursive call moves n-1 disks from "from_peg" to "aux_peg" using "to_peg". So after the first call, n-1 disks are in "aux_peg" in the order of size and the nth disk is on "from_peg". Next, we move the nth disk to "to_peg", after which again a recursive call moves the n-1 disks from "aux_peg" to "to_peg" using "from_peg". Thus finally the "to_peg" contains all the n disks in order of size.

Time complexity

Let the time required for n disks is T(n). There are 2 recursive calls in order to move n-1 disks and one constant time operation to move a disk from "from_peg" to "to_peg".
Thus we have:
$T(n) = 2T(n-1) + 1$ (1)
As we have, $T(n-1)=2T(n-2) + 1$
Substituting T(n-1) in (i) we have
$T(n) = 2(2T(n-2) + 1) + 1$
Following the same procedure we have,
$T(n)= 4T(n-2) + 3$
$T(n) = 4(2T(n-3) +1) +3$
$T(n)= 8T(n-3) + 7$
Thus generalizing the above sequence, we have:
$T(n) = 2^kT(n-k) + (2^K -1)$
As, $T(0) = 0$
Hence when $k\rightarrow n$ we have
$T(n) = 2^nT(0) + (2^n - 1)$
$T(n) = 2^n – 1$
Time complexity is O(2^n)

Space complexity

Space required for each recursive call is independent of n, i.e. constant. Let it be k.
When we make the 2nd recursive call, 1st recursive call is over and hence we can reuse the space of 1st call for 2nd call. Thus we have:

$T(n) = T(n-1) + k$
$T(0) = k$
$T(1) = 2k$
$T(2) = 3k$
$T(3) = 4k$

So, the space complexity is $O(n)$.
Thus, for recursive solution of Towers of Hanoi, the time complexity is exponential but the space complexity is linear.

Iterative algorithm

As we know that parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").[3]
The most important aspect of parallel computation is the task dependency. If a task A needs to be performed after the completion of task B then we cannot parallel compute these two tasks as they are dependent. Thus more the independency among the task being performed, more would be the chances of performing their computation parallel.
In recursive algorithm, we are dividing the problem into sub-problems (in order to compute solution for n disks, we are finding solution for n-1 disks), but the problem is that there is dependency between these tasks i.e. without knowing the solution for n-1 disks, we cannot compute the solution for n disks. By using the iterative algorithm suggested below, we could remove the task-dependency and thus exploit the wonderful technique of parallel computation to find the solution to our problem.
Also by using the recursive calls, there is an extra overhead of context switching which utilizes the stack memory space and as we saw above the space complexity is linear. The algorithm suggested below would do computation using iteration which is more efficient in terms of memory usage since no context switch is required.
To start with, let us first explore all the elements of this iterative solution, and then merge all these components together to formulate the algorithm.
First, we need to find the disk number to be moved at a particular step, this could be achieved keeping the following in mind:

- The disk numbered 1 is to be moved at each alternate steps starting with steps 1 and then 3,5,7,9 and so on. Thus the smallest disk is to be moved at each odd step.
- The disk numbered 2 is to be moved at the step 2 first and then 6, 10, 14 and so on. Thus the sequence of steps at which the second-smallest disk is to be moved follows an airthematic sequence with 1st element being 2 and common difference 4.
- The disk numbered 3 is to be moved at the step 4 first and then 12, 20, 28 and so on. Thus the sequence of steps at which the third-smallest disk is to be moved follows an airthematic sequence with 1st element being 4 and common difference 8.

We could easily notice that for a disk d, the first step at which it is to be moved is 2^{d-1} and then the steps follow a sequence with the common difference among them being 2^d.
Thus for each iteration we have to find a disk number d such that the iteration number (step), i belongs to one of these sequences. So we have to find a disk d such that the step number, i follow the following condition:

$i \bmod 2^d = 2^{d-1}$
where i := step number
d:= disk number
mod := modulus operator

By performing the above computation, we would have the disk number, d to be moved at a particular step. Now the next step is to find the movement to be performed pertaining to that particular step. Examples:

1. For n=4, we would have $2^4-1 = 15$ steps:
   - If step i = 5, we have
     $5 \bmod 2^1 = 1$ which is equal to 2^{1-1}
     Hence, our d=1.
   - If step i = 10, we have
     $10 \bmod 2^1 = 2$ which is not equal to 2^{1-1}
     Next we check,
     $10 \bmod 2^2 = 2$ which is equal to 2^{2-1}
     Hence, our d = 2.

- If step i = 12, we have
  12 mod 2^1 = 0 which is not equal to 2^{1-1}
  Next we check,
  12 mod 2^2 = 0 which is not equal to 2^{2-1}
  Next we check,
  12 mod 2^3 = 4 which is equal to 2^{3-1}
  Hence, our d = 3.

As we know that there are three pegs, hence the disk at any point could be at one of these three pegs namely A, B and C. Thus the movement of disk could be from one of these three pegs. Also remember the basic idea for all the iterative algorithms for Towers of Hanoi according to which the disk transfer can occur in two directions using cyclic moves. For an even number of discs n, the direction of odd-numbered disks is 1→2→3→1, and for even-numbered disks is 1→3→2→1. Reversely for an odd n the directions are reversed i.e. the direction for even-numbered disks is 1→2→3→1, and for odd-numbered disks is 1→3→2→1.
Thus in total six cases are required to account for all the possible movements of disks, with three cases pertaining to the peg on which disk currently resides and for each peg, two cases for the clockwise or anti-clockwise movement. We can find the peg at which the disk currently sits and the peg on which it would be placed next using the following ideas:

- The step numbers at which movement of a particular disk is performed follows an airthematic sequence.
- By using the value of n i.e. total number of disks and the value of d i.e. the disk number to be moved (which was calculated previously), we can calculate the clockwise or anticlockwise movement to be performed.

Thus, through above two ideas combined together, we can find the current peg number on which the disk sits as well as the next peg where the disk is to be moved. Examples:

1. For n=4 (total number of disks is 4)
   For i=3, we have d=1 (disk number 1) i.e. disk 1 is involved in the movement
   The step number sequence for disk 1 would be 1, 3, 5, 7 and so on with the first element=1, common difference=2.Using
   e= f + (no-1)cd
   where e = new element (here 3)
   f = first element
   cd = common difference
   no = element position in sequence
   3 = 1 + (no – 1) 2
   no - 1 = 1
   no = 2

Also n=4 i.e. even and d = 1 i.e. odd, hence the movement is being performed in clockwise direction.
As no = 2, thus 3 is the second element of the sequence, hence disk number 1 should be at peg B (second position from the start in clockwise direction) and should be moved to peg C. (clockwise movement).

2. For n=4 (total number of disks is 4)
   For i=10, we have d=2 (disk number 1) i.e. disk 2 is involved in the movement
   The step number sequence for disk 1 would be 2, 6, 10, 14, with the first element=2, common difference=4.Using
   e= f + (no-1)cd
   10 = 2 + (no – 1) 4
   no - 1 = 2
   no = 3
   Here n=4 i.e. even and d = 2 i.e. even, hence the movement is being performed in anti-clockwise direction.
   As no = 3, thus 10 is the third element of the sequence, hence disk number 2 should be at peg B (third position from the start in anti-clockwise direction) and should be moved to peg A. (anti-clockwise movement).
3. For n=3 (total number of disks is 4)
   For i=5, we have d=1 (disk number 1) i.e. disk 1 is involved in the movement
   The step number sequence for disk 1 would be 1, 3, 5, 7 with the first element=1, common difference=2.Using
   e= f + (no-1)cd
   5 = 1 + (no – 1) 2
   no - 1 = 2
   no = 3
   Here n=3 i.e. odd and d = 1 i.e. odd, hence the movement is being performed in anti-clockwise direction.
   As no = 3, thus 5 is the third element of the sequence, hence disk number 1 should be at peg B (third position from the start in anti-clockwise direction) and should be moved to peg A. (anti-clockwise movement).

4. For n=3 (total number of disks is 4)
   For i=6, we have d=2 (disk number 1) i.e. disk 2 is involved in the movement

The step number sequence for disk 2 would be 2, 6 with the first element=2, common difference=4.Using
e= f + (no-)cd
6 = 2 + (no – 1) 4
no - 1 = 1
no = 2
Here n=3 i.e. odd and d = 2 i.e. even, hence the movement is being performed in clockwise direction.
As no = 2, thus 6 is the second element of the sequence, hence disk number 2 should be at peg B (second position from the start in clockwise direction) and should be moved to peg C. (clockwise movement).
The pseudo-code is depicted as under:

```
  void towers(int n)begin
 for i=1→ $2^n$-1 begin
      for j=1→n begin
           if (i%$2^j$==$2^{j-1}$) then
                d=j
                break
           endif
           endfor
      switch((i-$2^{d-1}$/$2^d$)%3)begin
           case 0:
                if((d%2==1) and (n%2==0) or   (d%2==0) and (n%2!=0))then
                     print " A->B"
                endif
                else
                     print " A->C "
                endelse
                break
           case 1:
                     if((d%2==1) and (n%2==0) or (d%2==0) and (n%2!=0))then
                     print " B->C"
                endif
                     else
                          print " C->B "
                endelse
                     break
                case 2:
  if((d%2==1) and (n%2==0) or (d%2==0) and (n%2!=0))then
                          print " C->A"
                     endif
                     else
                          print " B->A "
                     endelse                              break
           endswitch
      endfor
   end
```

Time complexity

The outer loop is going to execute for the 2^n steps, the overall time complexity T(n), is dependent on the number of times the inner loop (for disk selection) is executed along with the constant execution time k for the rest of the computation (computing movements to be performed).
The number of times inner loop is executed can be calculated using following observations:

- For half of the steps i.e. $2^n/2$ steps, the first disk would be selected and hence the loop will only execute once.
- The disk 2 would be selected for $2^n/4$ steps, hence the inner loop will execute twice for $2^n/4$ steps.
- Similarly disk 3 would be selected for $2^n/8$ steps, hence the inner loop will execute three times for $2^n/8$ steps.

Thus, in general, corresponding to disk d, the inner loop will be executed $d\times 2n/2^d$ times.
Thus the overall time complexity is the total number of steps involved which would:
$T(n) = 1(2^n/2) + 2(2^n/4) + \ldots\ldots.. + n(2^n/2^n) + k(2^n)$
$T(n) = 2^n (1/2 + 2/4 + 3/8 + 4/16 + \ldots. n/2^n) + k(2^n)$
The series $(1/2 + 2/4 + 3/8 + 4/16 + \ldots. n/2^n)$ converges to 2 at ∞. Hence,
$T(n) = 2^n (2 + k)$
$T(n) = K1\times 2^n$
Time complexity is O(2n).
Space complexity
In the case of the above iterative algorithm, the space is required to store the following:

- The value of counter variables i and j.
- The value of temporary calculations inside the loop.
- Other data like Program Counter etc.

The value of counter variables i and j as well as program counter need only fixed memory space i.e. O(1). Each iteration of the loop needs O(1) space for temporary variables, however since loop iterations are independent of each other hence the space for temporary variables can be reused for the other iterations as well.
Hence, the overall space complexity is O(1).

Future work
The Towers of Hanoi problem requires 2^n-1 disk movements to transfer all n disks from peg A to peg C using peg B. The solution is exponential to get some representation of series of move. However, it can be observed that the problem has overlapping solution i.e. number of different possible movements among pegs A, B and C is fixed. It is equal to different permutations possible, which is equal to 6. So, with dynamic problem one can trade exponential space for less than exponential time, at least for a region in which dealing with exponential space does not require exponential time [4].
[4] To solve n disk problem, one can have a memoization matrix of n+1 rows and 6 columns to contain subproblem solutions. Character strings will represent the problem solution. Each move will be represented by a single character to store earlier results. Every disk movement operation is treated as an ordered pair, giving the source and destination pegs, ranging from 01 up through 21. If these are considered to be base-3 numbers, the range of magnitudes runs from 1 through 7, and can be stored as a single octal digit. The solution then becomes a string of octal digits representing all moves to accomplish the transfer of all disks from the initial source tower to the final destination tower. Thus, for the transfer of n disks from peg A to peg C, using peg B for intermediate storage, the solution will be the transfer of (n–1) disks from A to B, using C for intermediate storage, followed by the transfer of one disk from A to C, and ending with the transfer of (n–1) disks from B to C, using A for intermediate storage. This represents the solution for (n–1) from A to B using C, concatenated with the single digit representing the move from A to C, and then concatenated with the solution for (n–1) from B to C using A.

Conclusion
The Towers of Hanoi problem is one of the most famous problems in the field of computer science. It had been solved and explained using the recursive algorithm, but apart from this recursive solution, the iterative solutions to this problem also exist. The main idea behind all these iterative algorithms is that the disk transfer can occur in two directions using cyclic moves.
An iterative solution for Tower of Hanoi was presented with the same idea. The algorithm finds the disk number involved in the movement along with the movement to be performed.
The space complexity of the recursive solution is linear as compared to constant space complexity of the suggested iterative solution. Thus, the suggested algorithm is space efficient as compared to recursive algorithm.
The time complexity of both the algorithms is $O(2^n)$. However, due to the fact that the iterative algorithm involves independent iterations, hence parallel computing can be used to find the solution more quickly.
For example, to solve the problem on n disks, the recursive solution takes time T. Thus, using the above suggested algorithm with n system cores or n systems, the running time for each system would be T/n, and hence the solution could be achieved in T/n time which is far more efficient than time T.

References

1. J.A.Storer, Springer, 2002 Towers of Hanoi Puzzle(from an introduction to Algorithms and Data structures).
2. Hayedeh Ahrabian, Comfor Badamchi and Abbass Nowzaru-Dalini, On the Solution of the Towers of Hanoi Problem. World Academy of Science, Engineering and Technology, Vol:5 2011-01-24
3. Gottlieb, Allan; Almasi, George S. (1989). Highly parallel computing. Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1.
4. Rolfe, Timothy J., "Dynamic Programming the towers", inroads, Vol 3, No. 3, (September 2012), pp. 40-45
5. Sniedovich, Moshe(2002). "OR/MS Games: 2. The Towers of Hanoi Problem"
6. Chan, T.(1988). "A statistical analysis of the towers of Hanoi problem". Internat J. COmput. Math. 28:57-65.
7. Gedeon, T.D.(1996). "The cyclic Towers of Hanoi:An iterative Solution produced by Transformation".The computer journal 39(4).
8. M.C.ER(1984):"The Cyclic Towers of Hanoi:A representation Approach"
9. Hinz, A.(1989). "The Tower of Hanoi". L'Enseignement Mathematique 35:289-321.
10. Troshkin, M. "Doomsday Comes: A Nonrecursive Analysis of the Recursive Tower-of-Hanoi Problem"
11. Daniel W.Palmer:"Exploring recursion with variations on the Towers Of Hanoi"
12. Paul Cull And E.F.ECKLUND.JR.:"Towers Of Hanoi and Analysis of Algorithms"
13. Dr.J.-S.Wu:"The towers of Hanoi problem with parallel moves"
14. M.C.ER(1983):"An iterative Solution to the Generalized Towers Of Hanoi Problem"
15. M.C.Er(1986):"A loopless approach for constructing a fastest algorithm for the towers of hanoi problem