IBIS ASSIGNMENT

24-PUZZLE

September 23, 2018

Name : Robin Kumar Sharma
VUnet ID: rsa890
Vrije University, Amsterdam

Contents

1. Abstract 2
2. Problem statement 2
3. Design 2
4. Implementation 3
5. Result Analysis 3
0.1 Sequential 24-puzzle 4
0.2 Parallel 24-puzzle 4
6. Conclusion 8
References 8

1 Abstract

Objective of this assignment is to implement a parallel 24-puzzle in Java using Ibis system on DAS-4 platform. This report contains design, implementation, result analysis.

2 Problem Statement

24-puzzle is a 5X5 board game in which numbers are filled from 1 to 24 and one empty slot is available in the board to solve the board. The program can move the empty slot up, down, left and right (if possible). At a time from one position maximum 3 moves are possible since going back to previous move is redundant work. The sequential board solver shuffles the board till the length give as command line argument. After shuffling is done, game solver try to find the solve the board from a particular position if possible otherwise it moves to the next board position and try to solve recursively. From a particular board position a solution is not found if board distance becomes greater then bound.
In parallel version of the 24-puzzle solver, after shuffling the board, boards the given to the processors and they try to solve it. We have implemented it in work stealing model, it means if a client finish solving the board then it asks more work from server immediately. It's called pull model as well. Ibis is based on client-server paradigm. One server is elected and then clients can connect to that server. Once the pool is closed which is basically all the clients have connected to the server, server gives work to the clients on their request until all solutions are not found.

3 Design

The provided sequential 24-puzzle is an iterative recursive algorithm. For each board position it calculate the solution recursively and decide if it can find solution.
In Parallel 24-puzzle implementation, we try to minimize the execution time using the multi-processors.
Initially we have shuffles board based on length, P processors and sequential 24-puzzle algorithm. We have to divide equal amount of work among P processors. To keep balance of work, we pre-compute board position up till some limit, which is controlled by pool size and length factor.
To implement the client and server, first we initialize the SendRequestBoardPortType, SendRequestID-PortType, SendRecvReplyPortType which are used for receiving/sending board , Port type

used for receiving/sending ID of clients and port type used for receiving/sending a reply from client respectively. For receiving the client request upcalls have been used explicitly.

4 Implementation

Every process does execute the same program but on different data set. This is SPMD paradigm.

Server Working

- initialize the board based on the length
- Receive client ID
- generate enough boards based on the pool size
- based on client request, give boards to the clients
- receive solutions found from clients and show final result

Client working

- Send client id to server
- receive board from client
- solve the board
- send solution to server
- ask more board if connection not closed by server

In the end server collect all results and print.

5 Result Analysis

Sequential and Parallel algorithms are executed on DAS4 cluster with hardware specification as follows, 16 nodes with Intel Xeon CPU 2.40GHz, 23GB memory per node, gigabit Ethernet network connection. Performances are evaluated in terms of speedup and efficiency.

$$speedup(S) = \frac{Tseq}{Tpar}$$

$$efficiency = \frac{S}{P}$$

The results are evaluated in following manner - For every result 10 runs have been collected and average of those have been reported as time taken in milliseconds. Prun have been used for collection of parallel and sequential results. All the execution reported is in milliseconds. Program passed the sanity check successfully.

0.1 Sequential 24-puzzle

Fig 1. below show the execution time, number of steps and out of those solutions found by the sequential board solver with cache enabled. These results were collected using prun on 1 processor.

Length	Time	Solution	Steps
10	1	1	10
50	1.6	1	50
100	2562.6	1	100
105	855.066	1	105
290	4271	1	60
300	1646	1	60
1000	2722.2	1	54
2000	1	1	16
3000	27057	3	64
40000	39367	4	64

fig 1. Sequential 24-puzzle execution time

0.2 Parallel 24-puzzle

In this section, we will present the results and analysis of the parallel 24-puzzle.

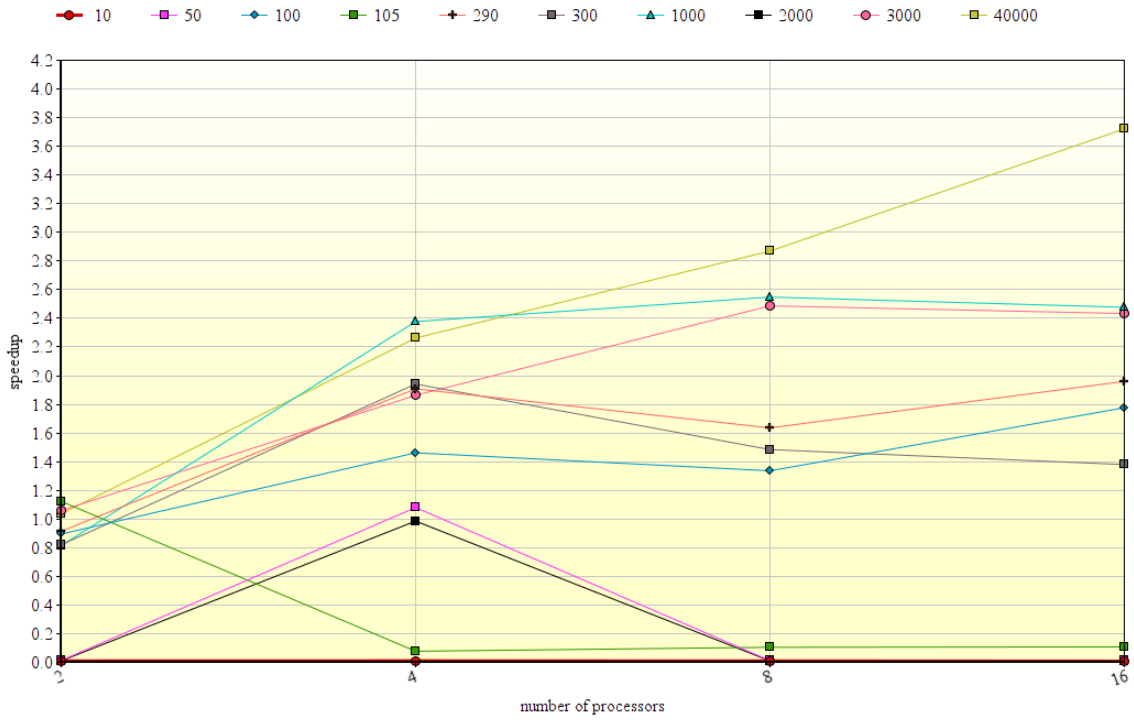
Fig 2. below show the execution time for parallel board solver with cache enabled. These results were collected using prun on given processors as column headers.

Length - num_processors->	**2**	**4**	**8**	**16**
10	102.4	96.2	106.9	127.7
50	118.7	109.9	135.1	117.3
100	2862.1	1957.9	1918	1442.6
105	761.45	9824.8	8137.4	7946.9
290	4668.8	2447.1	2611.1	2179.1
300	2007.1	1033.6	1109	1192.9
1000	3343.9	1406.5	1068.25	1099
2000	94.8	96.2	100.7	121
3000	25546.2	13713.2	10879.6	11118.2
40000	37987.8	16792.4	13722	10578.2

fig 2. Parallel 24-puzzle execution time

Length - num_processors->	**2**	**4**	**8**	**16**
10	0.009765625	0.0103950104	0.00935453695	0.007830853563
50	0.01347935973	1.080072793	0.0118430792	0.0136402387
100	0.8953565564	1.461821339	1.336079249	1.776375988
105	1.122945258	0.07750284993	0.1050786082	0.1075975118
290	0.9147960932	1.907890973	1.635709088	1.959983479
300	0.8200886852	1.941853715	1.484220018	1.379830665
1000	0.8140793684	2.377461785	2.548279897	2.476979072
2000	0.01054852321	0.9854469854	0.009930486594	0.00826446281
3000	1.059139911	1.862891229	2.48694805	2.433577378
40000	1.036306393	2.262201949	2.868896662	3.72152162

fig 3. show the speed up graph of various length with number of processors.



Speed up results are not even close to perfect speedup, it's mainly due to that fact that 24-puzzle is recursive and exponential search. The best speedup we get is for length 40000, which means as the number of length has increased the performance with more processors increases.

Using cache improve execution time but very marginally. Cache and nocache doesn't cause much difference just due to the fact that current java language version has very good garbage collection and the following results shows same -

Length - no/cache->	**4-cache**	**4-nocache**	**8-cache**	**8-nocache**
1000	1575.6	1631.3	1126.1	1175
2000	94.6	92.1	102.7	98
3000	10960.7	10802.4	11495.4	11395.2
40000	18961.9	18456.3	10679	10480.1

fig 4. execution time with move factor - 8

To improve the performance following steps were taken -

a. different length factor - based on pool size and length factor, program precomputed boards

for load balancing the results achieved based on different board length are follows.

Length - num_processors->	**2**	**4**	**8**	**16**
10	114.1	93.8	99.6	122.4
50	115.1	118.4	118.1	143.7
100	2807.2	2154.6	1963.4	1965.5
105	15302.9	8075.9	8416	8189.6
290	4640.9	2636.4	2492	2618.7
300	1955.6	1107.8	1056.5	1147.6
1000	3279.8	1463.3	1112.4	1131.9
2000	107.7	92.2	94.9	175.7
3000	25109.6	14120.2	13638.5	10814.9
40000	38024.8	16741.2	13525.2	10557.4

fig 5. execution time length factor - 2 with cache enable

Length - num_processors->	**2**	**4**	**8**	**16**
length	2	4	8	16
10	100.5	91.7	101.3	112.7
50	128.4	109.5	100.6	119.1
100	2935.2	1917.3	1441.8	1523.7
105	15392.7	8683.9	8282.2	6116.3
290	4821.2	2936	2220.7	2189.7
300	2036.9	1205.4	1213.6	1430.5
1000	3473.9	1575.6	1126.1	1346.1
2000	100.3	94.6	102.7	147.9
3000	25749.9	10960.7	11495.4	9472.5
40000	38799.7	18961.9	10679	9776.4

fig 6. execution time length factor - 8 with cache enable

As we can see in fig 5 and fig 6, higher length factor definitely improve the execution time, thats why length factor of 8 perform better because it does load balancing properly.

Some corner cases were found with different length where sequential and parallel program

takes too long time due to exponential search due to which solution is found in the last board step, such as for length = 900, time taken by sequential program was 7291507.333 milliseconds and length = 110, sequential time = 4999309, 2 processor time = 5067029 milliseconds.

6 Conclusion

Current implementation could be extended via use of multi-threaded for each processor board calculation. It would improve performance in case of exponential search. Caching use could be improved to have better results.
Overall it was a good learning experience to work and learn ibis with limit resource availability.

References

[1] Ibis library manual

[2] Sequential 24-puzzle program