Parallel Bidirectional A* Search on a Symmetry Multiprocessor†

Andrew Sohn
Dept. of Computer and Information Science
New Jersey Institute of Technology
Newark, New Jersey 07102-1982, sohn@cis.njit.edu

Abstract - *This report presents our experiences parallelizing and implementing search problems. We take the sequential A* search and parallelize it by combining with bidirectional search, called Parallel Bidirectional A* Search (PBiA*S). To identify the effectiveness of the PBiA*S, we implement two search problems, the Eight Puzzle and the Tower of Hanoi, on a Symmetry multiprocessor. Execution results demonstrate that the PBiA*S can be an effective parallel search method as it gives two to 12-fold speedup over the unidirectional A* search for the two search problems.*

1 Introduction

Search is one of the frequently used problem solving methods [1,5]. Given an initial state, it is to find a path which leads to the goal state. When finding a path, it encounters situations in which it decides which choice to take. Simple search strategies such as depth-first search, breadth-first search, bidirectional search, etc. can be used to select the choice points. However, these unguided brute-force search suffer from an efficiency issue. For a search tree of depth d with a branching factor b, depth-first search and breadth-first search may require the amount of processing time to be of order b^d or $O(b^d)$. For small b and d, these unguided search would work. However, as b and d become large, it is simply impractical to use these unguided search because of exponential processing time and memory usage.

Techniques to reduce the processing time and space complexity of search problems are typically classified into two categories: (1) algorithmic approach [3,6] and (2) parallel processing approach [4,7,10]. Algorithmic approach includes heuristic search [9]. The main idea is to use domain knowledge to guide the search process. Brute-force search follows all the paths until a solution is found. Heuristic search follows *selective* paths by using information which distinguishes promising states among others. A number of heuristic search has been developed, including hill-climbing, best-first search, A*, iterative-deepening A*, etc. [3,6,9].

Parallel processing approach emphasizes implementation. A simple parallel search would be to use depth-first search and assigns a processor to each path whenever more paths are to be searched as the search tree grows. This type of parallelization is problematic since for large search tree an infinite number of processors is needed. A better parallel search would be to use A* (or IDA*) search and find successors of n nodes in each iteration, where n is the number of processors. Or, one can also choose parallel bidirectional A* search where search is performed from both directions simultaneously while each direction performs parallel A* search. It is exactly the main objective of this report to investigate the parallel bidirectional A* search.

We start our discussion in section 2 by identifying potential parallelism in search. Section 3 presents parallel bidirectional A* search. Section 4 lists actual execution results of the two problems, 8-Puzzle (8-P) and Tower of Hanoi (ToH), on a 26-processor Sequent Symmetry multiprocessor. Section 5 summarizes our experimental results. Conclusions as well as future directions on parallel search are made in the last section.

2 Parallelism in search

A typical search process involves two lists: OPEN and CLOSED. OPEN contains nodes to be examined whereas CLOSED has those nodes that are already examined. Figure 1 shows a generic, sequential search process.

```
OPEN={initial_state}, CLOSED=∅
Repeat
1.  TEMP ← select(OPEN)
2.  SUCC ← expand(TEMP)
3.  SUCC ← filter(SUCC,OPEN,CLOSED)
4.  OPEN ← merge(SUCC,OPEN − TEMP)
5.  CLOSED ← merge(TEMP,CLOSED)
Until (goal_state ∈ SUCC) or (OPEN = ∅)
```

Figure 1: A generic and sequential search

Line 1 of Figure 1 is a selection step which selects a *single* node from OPEN. Line 2 generates successors of the selected node. Line 3 removes from SUCC those nodes that are either on OPEN or CLOSED. Line 4 merges two lists to form a new OPEN. Line 5 also forms a new CLOSED.

The above search algorithm is sequential in three facts: First, the entire loop is sequential due to loop-carried dependencies between iterations. The central data-structure is

† This work is supported in part by the NJIT SBR No. 421820.

OPEN. Iteration i uses OPEN which was *modified* at iteration $i-1$. No two iterations can therefore be executed in parallel. The second sequentiality lies within the loop. Each iteration consists of five steps. Those five steps are again sequential due to their data dependencies. For example, lines 2 and 3 cannot be executed in parallel due to the true data dependence. The third sequentiality stems from the fact that the selection step selects only *one* out of many nodes on OPEN. The expansion step therefore works only on one node at a time in each iteration.

Figure 2 shows an instruction level parallelism profile for the tree depth 7 of 8-P. Parallelism refers to a number of instructions that can be executed in parallel. The profile assumes an infinite number of processors and is constructed from the data-flow graph generated by Sisal [8]. The top one shows a complete parallelism profile for a search tree of depth 7 (or 7 iterations) while the bottom one is a closed-up version of the third iteration. The x-axis indicates execution time in terms of instruction cycles while the y-axis plotted to *logarithmic* scale is a number of instructions that can be executed in parallel, namely, parallelism.

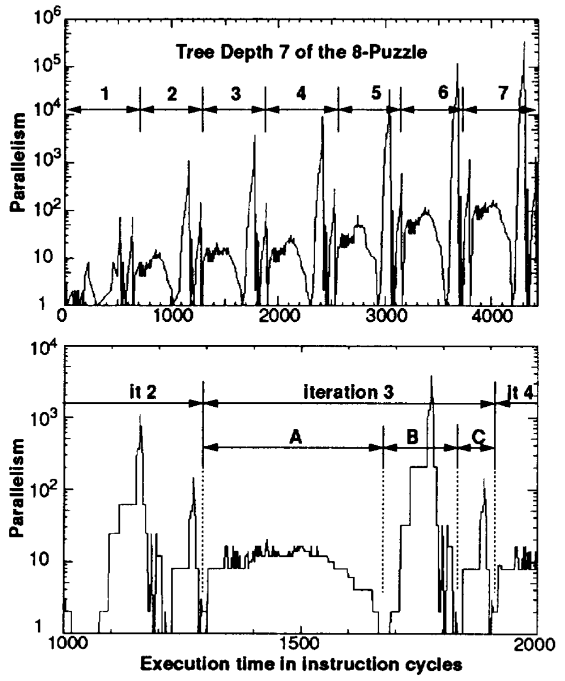

Figure 2: Parallelism profiles. The y-axis is in log scale.

The top plot shows that the parallelism in the 8-P exponentially increases as the number of iterations increases. For example, when the tree depth is 2, the potential parallelism reaches 1,080. However, when the tree depth becomes 4, the potential parallelism now reaches roughly 10,000. As the tree depth (iteration) reaches 7, the potential parallelism becomes 344,740!

The bottom plot of Figure 2 shows the third iteration and part of the fourth iteration. Iteration 3 can be divided into three regions marked A, B, and C. Each of these three regions corresponds respectively to expanding successors, checking the existence of newly generated successors, and testing the two termination conditions.

3 Parallel Bidirectional A* Search

Figure 2 has shown that the 8-Puzzle does have a large amount of parallelism which can be utilized by multiprocessors. We parallelize the sequential A* search by combining with bidirectional search, resulting in Parallel Bidirectional A* search (PBiA*S). The PBiA*S searches from both directions while each direction performs the parallel A* search. To illustrate the PBiA*S, we need five lists: TOPEN and TSUCC for top-down (forward) direction, and BOPEN and BSUCC for bottom-up (backward) direction. CLOSED does not change. Figure 3 illustrates the PBiA*S. Let $N_{f,g,h}$ be a node, where g is the cost to the current state, h is an estimated cost of reaching the goal state from the current state, and f is the sum of g and h. Assuming that g is 1 per arc and h is the number of misplaced tiles, $2_{4,1,3}$ will denote node 2 with f=4, g=1, and h=3.

Suppose we have *four* processors. Given TOPEN = (node 0), BOPEN = (node 16), and CLOSED = (), the PBiA*S will execute two iterations:

Fwd It. #	Slct/Expnd		TSUCC		TOPEN	CLOSED
	P0	P1	P0	P1	P0	
0					0	
1	0		1, 2, 3		$1_{4,1,3}$, $2_{4,1,3}$, $3_{6,1,5}$	0, 16
2	1	2	4, 5, 6	7	$7_{4,2,2}$, $3_{6,1,5}$, $5_{6,2,4}$, $4_{7,2,5}$, $6_{7,2,5}$	0, 16, 1, 2, 7, 19

Bwd It. #	Slct/Expnd		BSUCC		BOPEN
	P2	P3	P2	P3	P2
0					16
1	16		19, 20, 7, 21		$7_{4,1,3}$, $19_{5,1,4}$, $20_{6,1,5}$, $21_{6,1,5}$
2	7	19	2, 26	22,23	$2_{4,2,2}$, $22_{6,2,4}$, $26_{6,2,4}$, $20_{6,1,5}$, $21_{6,1,5}$, $23_{7,2,5}$

The first column shows forward/backward iteration numbers while the second column indicates select/expand operations. Nodes in TOPEN and BOPEN are listed in the ascending order of f value. For example, at iteration 2, the four processors P0,...,P3 perform the following four activities simultaneously:

- P0 generates (4, 5, 6) by expanding 1.
- P1 generates (7) by expanding 2. TOPEN = (7,3,5,4,6).
- P2 generates (2, 26).
- P3 generates (22, 23) BOPEN = (2, 22, 26, 20, 21, 23).

Iteration 2 then gives CLOSED = (0,16,1,2,7,19). At this moment, the search process stops because (TSUCC,BSUCC)

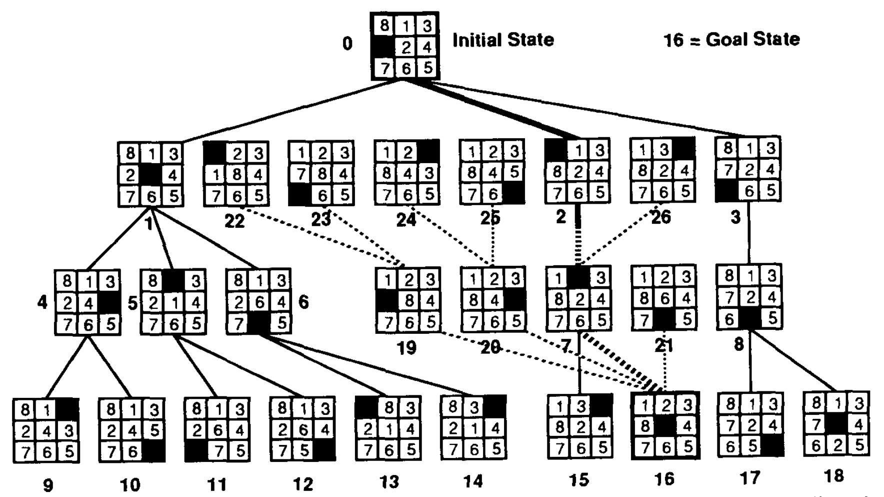

Figure 3: Parallel Bidirectional A* Search. Thick lines=solution path, solid lines=forward search, dotted lines=backward search.

$\cap$ (TOPEN,BOPEN,CLOSED) = 7 $\neq \emptyset$. The PBiA*S takes two iterations to meet in between the search space.

4 Experimental Results

We have implemented the 8-P and the ToH on a Sequent Symmetry shared-memory multiprocessor. All the programs are written in a functional language Sisal [8]. The Optimizing Sisal Compiler has significantly helped the automatic parallelization in implementation level [2].

For the Tower of Hanoi, we have executed five different problem sizes: 3-7 disks. For the Eight-Puzzle, we have also implemented 5 different problems as shown in Figure 4. Each problem is defined in terms of an optimal number of iterations (or tree depth). A particular initial state of Figure 4 and the goal state of Figure 3 uniquely define the problem size of the 8-P since the optimal number of iterations is fixed for the given initial state. For example, the left most initial state of Figure 4 (*d*=10) indicates that the tree depth (or optimal number of iterations to reach the goal state) is 10.

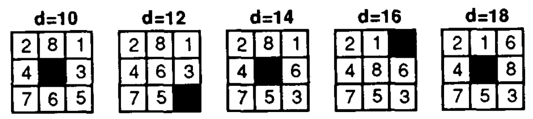

Figure 4: Five initial states for the 8-P. Goal state is in Figure 3.

Three types of execution results are collected: *execution time*, *number of iterations*, and *number of nodes searched.* Execution time is listed in Tables 1 and 2. Other results are listed in [10,11].

5 Discussion

Figure 5 shows speedup curves for the two search strategies. Speedup is defined as the execution time on 1 processor over *n* processors. We find that speedup is disappointing as the maximum speedup for the 8-P is merely 6.5. The main reason is because problem size is rather small which is evidenced by the fact that speedup increases as the problem size increases for both the PA*S and the PBiA*S. There are several other important reasons but we are unable to list all of them here. More details are presented in [10,11].

Figure 6 compares the two search strategies. The *y*-axis now shows the execution time ratio of PA*S to PBiA*S.

No. of Prcssrs	Parallel Bidirectional A* Search				
	10	12	14	16	18
1	0.83	2.69	8.18	11.73	94.53
2	0.57	1.51	4.81	7.85	52.46
4	0.68	1.62	2.79	5.57	34.93
6	0.35	1.40	2.84	4.41	25.21
8	0.46	1.07	2.67	4.02	21.96
10	0.53	1.29	2.27	4.64	18.28
14	0.66	0.91	1.97	3.65	17.45
18	0.74	1.21	2.57	3.19	16.59
22	0.86	1.46	3.35	4.27	17.27
26	1.03	1.83	4.44	5.58	17.04

Table 1: Execution time of the 8-Puzzle (in seconds)

No. of Prcssrs	Parallel Bidirectional A* Search				
	3 disks	4 disks	5 disks	6 disks	7 disks
1	0.06	0.33	2.26	19.68	164.22
2	0.05	0.27	1.69	13.14	125.39
4	0.06	0.19	1.37	10.54	94.76
6	0.05	0.20	1.04	10.53	92.54
8	0.06	0.19	0.96	9.00	88.50
10	0.05	0.19	0.99	7.42	97.61
14	0.06	0.20	0.95	6.93	58.51
18	0.07	0.20	0.98	6.83	64.78
22	0.07	0.20	0.97	6.82	59.11
26	0.06	0.22	0.99	7.15	58.35

Table 2: Execution time of the Tower of Hanoi (in seconds)

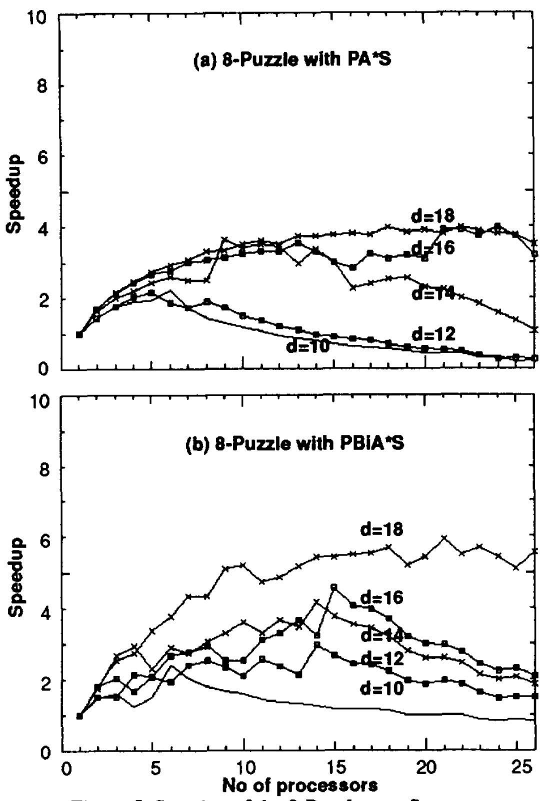

Figure 5: Speedup of the 8-Puzzle on a Symmetry.

We find from the plot that the PBiA*S is much faster than the PA*S. For the *d*=18 of the 8-P, the PBiA*S is over 10 times faster than the PA*S. As the problem size increases, the performance of PBiA*S also improves, in general.

To further explicate our approach to parallel bidirectional search, we are currently undertaking three tasks on: (1) different search problems, (2) large problem sizes, and (3) developing other parallel search techniques including *multidirectional* search.

6 Conclusions

This paper has presented our experiences parallelizing and implementing search problems in a multiprocessor environment. We have taken the sequential A* search and parallelized it by using bidirectional search, resulting in Parallel Bidirectional A* Search (PBiA*S). To identify the performance of PBiA*S, we implemented the 8-Puzzle and the Tower of Hanoi on a 26-processor Sequent Symmetry shared-memory multiprocessor. Both the sequential and parallel versions of search strategies have been implemented in a functional language Sisal. Actual execution results have demonstrated that our parallel search strategy is efficient in two facts: First, the PBiA*S has given over six-fold speedup on 26 processors. Second, the PBiA*S has also given over 10-fold speedup when compared with a unidirectional A* search. To further speedup the search process, we are currently generalizing PBiA*S to parallel multidirectional search.

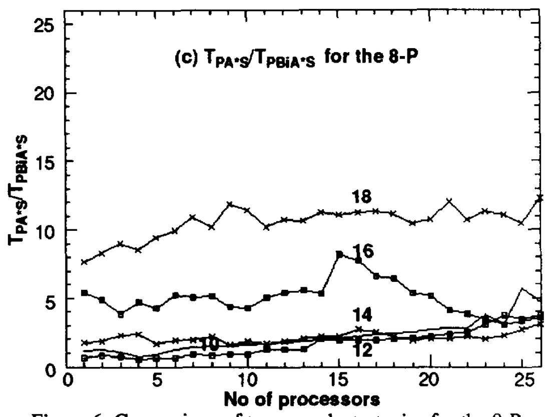

Figure 6: Comparison of two search strategies for the 8-P.

References

1. Bolch, L., and Cytowski, J., Search Methods for Artificial Intelligence, Academic Press, 1992.
2. Cann, D.C., and Oldehoeft, R.R., "A Guide to Optimizing SISAL Compiler," *Technical Report* UCRL-MA-108369, Lawrence Livermore Laboratory, Livermore, CA, 1991.
3. Hart, P.E., Nilson, N.J., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on SMC* 4(2), pp.100-107, 1968.
4. Kale, L.V., and Saletore, V.A., "Parallel State-Space Search for a First Solution with Consistent Linear Speedups," in *International Journal of Parallel Programming 19*, 1990, pp.251-293.
5. Kanal, L., and Kumar, V. (Eds.), Search in Artificial Intelligence, Springer-Verlag, 1989.
6. Korf, R., "Depth-first Iterative Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence* 27, 1985, pp.97-109.
7. Kumar, V., Ramesh, K., and Rao, V.N., "Parallel Best-First Search of State-Space Graphs: A Summary of Results," in *Proc. AAAI-88*, pp.122-127.
8. McGraw, J.R., Skedzielewski, S.K., Allan, S.J., Oldehoeft, R.R., Glauert, J., Kirkham, C., Noyce, W., and Thomas, R., "SISAL: Streams and Iteration in a Single Assignment Language: Reference Manual version 1.2," Lawrence Livermore Laboratory, Livermore, CA, 1985.
9. Pearl, J., Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, 1984.
10. Sohn, A., Gaudiot, J-L., and Sato, M., "Performance Studies of the EM-4 Data-flow Multiprocessor on Combinatorial Search Problems," *Technical Report* 92-39, Electrotechnical Laboratory, Tsukuba, Japan, Nov. 1992.
11. Sohn, A., "A Parallel Implementation of Bidirectional A* Search on a Symmetry Multiprocessor," *Technical Report* CIS-93-04, NJIT CIS Dept., May 1993.