Scalable Parallel Implementation of Bayesian Network to Junction Tree Conversion for Exact Inference[1]

Vasanth Krishna Namasivayam, Animesh Pathak and Viktor K. Prasanna
Department of Electrical Engineering
University of Southern California, Los Angeles
{namasiva, animesh, prasanna}@usc.edu

Abstract

We present a scalable parallel implementation for converting a Bayesian network to a junction tree, which can then be used for a complete parallel implementation for exact inference. We explore parallelism during the process of moralization, triangulation, clique identification, junction tree construction and potential table calculation. For an arbitrary Bayesian network with n vertices using p processors, the worst-case running time is shown to be $O(n^2w/p+wr^wn/p+n\log p)$, where w is the clique width and r is the number of states of the random variables. Our algorithm is scalable over $1 \leq p \leq nw/\log n$.

We have implemented our parallel algorithm using OpenMP and experimented with up to 128 processors. We consider three types of Bayesian networks: linear, balanced and random. While the state of the art PNL library implementation does not scale, we achieve speedups of 31, 29 and 24 for the above graphs respectively on the DataStar cluster at San Diego Supercomputing Center.

1 Introduction

Belief networks or Bayesian networks have been used in Artificial Intelligence since the 1960s, especially in medical diagnosis [12]. They have found application in a number of domains, including consumer help desks, nuclear reactor diagnosis, tissue pathology, pattern recognition, credit assessment, data mining [4], image analysis, robotics, genetics [13] and computer network diagnosis [12].

There are two main approaches for computing probabilities in a Bayesian network - $exact\ inference$ and $approximate\ inference$. The Lauritzen Speigelhalter algorithm [9] is the most popular exact inference algorithm. A program implementing the LS algorithm could be divided into four distinct modules as mentioned in [8]. They are $Junction\ Tree\ generation$, $Junction\ Tree\ pre-computation$, $Junction\ Tree\ propagation$ and $Belief\ Computation$. This paper deals with the first module, the Junction Tree generation. This paper explores a scalable, parallel algorithm for Junction Tree generation. We have addressed the rest of the exact inference algorithm in an earlier paper [10].

The time taken for generating a junction tree from a Bayesian network can be a significant fraction of the overall time for exact inference. On a 1000 node linear Bayesian network with number of states varying from 2 to 16, we observed that the junction tree conversion time was as high as 50% of the overall serial time for exact inference.

There has been considerable work in parallelizing exact inference. These include algorithms in [2], [3], [7], [8], [14] and [11]. The popular exact inference junction tree algorithm for multiply connected networks by Lauritzen and Spiegelhalter [9] was also conceived as a parallel algorithm with one processor per clique.

The junction tree generation consists of five main stages: *moralization*, *triangulation*, *clique identification*, *junction tree construction* and *potential table computation*. We have parallelized all the above stages, and combined the triangulation and clique identification stages. The combined time complexity of our parallel approach is $O(n^2w/p + wr^wn/p + n\log p)$. The descriptions of these stages, parallel algorithms for them, and detailed analysis is presented in Section 3. Our algorithm is scalable over $1 \leq p \leq nw/\log n$.

We have implemented the above parallel algorithm using OpenMP on the DataStar Cluster at the San Diego Supercomputer Center. The number of processors used in our experiments ranged from 1 to 128. We consider three types

[1]The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA) or the U.S. government.

Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) through the Department of the Interior National Business Center under grant number NBCH104009.

Proceedings of the 18th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'06)
0-7695-2704-3 /06 $20.00 © 2006IEEE

IEEE COMPUTER SOCIETY

of Bayesian networks: linear, balanced and random. While the state of the art PNL library [5] implementation does not scale, we achieved speedups of 31, 29 and 24 respectively for the above graphs.

The rest of the paper is organized as follows: in Section 2 we cover the basics of Bayesian inference and Junction trees. Section 3 describes the parallel algorithm and our implementation. Section 4 describes the experiments which were conducted to explore scalability and efficiency of the parallel algorithm. Section 5 concludes the paper and discusses possible future work.

2 Background

2.1 Bayesian Networks

A *Bayesian network* [12] is represented by a graph which is composed of a set of nodes, each node representing a variable. Each edge connects two nodes, and has a direction assigned to it. An edge between two nodes indicates a relation between the nodes and the direction indicates the causality. Let the variables be $\{X(1), X(2), \ldots, X(n)\}$. The probability that variable $X(i)$ has a value k_i, is given as $P(X(i) = k_i)$ where $k_i \in \{k_1, k_2, k_3, \ldots, k_r\}$.

Node A is a *parent* of node B if there is an edge from A to B. If a node has a known value, it is said to be an *evidence node*. Let $parents(A)$ be the parents of the node A. Then the joint distribution for $X(1)$ through $X(n)$ is represented as the product of the probability distributions $P(X(i)|parents(X(i)))$ for $i = 1$ to n. In mathematical notation the joint probability, also known as the *conditional probability table* (CPT), is expressed as

$$P(A) = \Pi(P(X(i)|parents(X(i))))$$

2.2 Bayesian Inference

In a Bayesian network, when we get new information about variables in the network, we update the conditional probability tables to reflect this new information. This updating is known as $evidence\ propagation$ [9]. Once all the beliefs are updated, the conditional probability tables contain the most recent beliefs in any variable and can be queried like a simple database to evaluate probabilities. The bigger problem we are exploring is the problem of *exact inference* on a Bayesian network. Exact inference involves determining the probabilities of the query variables, given the exact state of the evidence variables.

2.3 Junction Trees

For the sake of completeness we borrow the following definitions and notations from [1]. An undirected graph is said to be chordal or *triangulated* iff every cycle of length four or greater contains a chord between any two non adjacent nodes in the cycle. A *clique* in an undirected graph $G(V, E)$ is a set of vertices V such that the subgraph induced by V is a complete graph. A *junction tree* of a chordal graph G is a tree J such that each maximal clique C of G is a node in J, and all the cliques represented by the nodes of the junction tree J satisfy the *running intersection property* [7]. i.e for each node v of G, the nodes representing the cliques in J containing v form a connected subtree of J. Each edge (u, v) in J is labeled with the intersection of the cliques represented by u and v; these labels are called separator sets or *sepsets*. The *clique width* of a junction tree J is defined as the maximum number of random variables in the clique represented by any node in J. A *maximal clique* is a clique which is not a proper subgraph of any other clique.

Most exact inference algorithms like the popular LS algorithm [9] do not work on the Bayesian network directly, instead they work on the junction tree generated from it. The main reason for this conversion is that evidence propagation based on Bayes rule can not be applied directly to non-singly connected networks i.e Bayesian networks with loops, as it would yield erroneous results. Loops are undirected cycles in the underlying network.

Given the joint probability distribution of an arbitrary Bayesian network, we can perform exact inference on the network. The authors in [9] show that the joint probability distribution computation can be rearranged to correspond to the topology of the junction tree derived from the Bayesian network. Hence a junction tree can be used in place of a Bayesian network for inference. It should be noted that since each nodes in the junction tree represents a set of nodes in the original Bayesian network, the information contained in the edges of the original DAG is lost. However, that information is re-inserted in the nodes of the junction tree by the steps of potential table creation and propagation.

After the above transformation, the resulting junction tree can be used for computing inference. The algorithm employed for a junction tree is similar to the one employed for the Bayesian network, save for the actual contents (and sizes) of the messages passed between the nodes during the computation. More details of the above process can be found in [10].

3 Parallel Algorithm

In [9], the authors approached the problem of non-singly connected networks from a mathematical perspective. This section discusses our parallel version of their sequential algorithm for Junction Tree Generation, followed by a detailed analysis. Throughout this paper, we represent graphs using the adjacency list representation. Our parallel algorithms proceeds in the following four stages.

IEEE COMPUTER SOCIETY

3.1 Moralization

A *moral graph* of a DAG G is an undirected graph where all the parents of each node in G are connected to each other. Our parallel algorithm involved in obtaining a moralized graph from a DAG is shown in Figure 1.

Algorithm Moralize(G)
Input: $G(V, E)$: G is a DAG
Output: G_{mor}: the moralized version of G
1: $G' = (V', E')$ // $V' = V$, $E' = \phi$
2: begin parallel across all processors
//at processor k ($k = 0$ to $p-1$) process n/p nodes
3:for $i = k \cdot (n/p)$ to $(k+1) \cdot (n/p) - 1$
4: $\forall (v_i, w) \in E$, insert (v_i', w') in E'
5: $\forall (u, v_i) \in E$, insert (v_i', u') in E'
6: end for i
7: end parallel
// G' is an undirected version of G now
8: begin parallel across all processors
//at processor k ($k = 0$ to $p-1$) process n/p nodes
9: for $i = k \cdot (n/p)$ to $(k+1) \cdot (n/p) - 1$
10: $Parent_i = u : (u, v_i) \in E$ //parents of v_i
11: $\forall u, w \in Parent_i$, insert (u', w') in E'
12: end for i
13: end parallel
14: return G'

Figure 1. Parallel algorithm for moralization

3.2 Triangulation and Identifying Maximal Cliques

An undirected graph is *triangulated* iff every cycle of length four or greater contains a chord between any two non adjacent nodes in the cycle. Our parallel algorithm for the fill-in computation to obtain a triangulated graph is shown in Figure 2. Since the task of finding an optimal triangulation with minimum clique width is NP hard [9], we use a greedy heuristic for the same. A unique feature of our algorithm is that we are able to identify the cliques from the triangulated graph as it is being constructed as seen in Steps 16 to 21 in the algorithm.

3.3 Constructing the Junction Tree

After identifying the cliques, we proceed to connect them to obtain the junction tree. The parallel algorithm for this stage is shown in Figure 3. The output of this algorithm is an adjacency list which represents the connections between supernodes represented by the cliques. The running

Algorithm TriCliq(G)
Input: $G(V, E)$: G is a moralized undirected graph
Output: C: a set of cliques formed from G
1: $G'(V', E') = G$ // Make a temp copy
2: $C = \phi$
3: while ($V' \neq \phi$) do
4: begin parallel across all processors
// each processor computes $f(v)$ for n/p nodes
// and they together compute the min
//$f(v')$ = number_of_neighbors(v')
5: find $v' : f(v') \leq f(u'), \forall u' \in V'$
6: end parallel
7: $N(v') = \{u' : (u', v') \in E'\}$// Neighbors of v'
8: $C_{v'} = N(v') \cup \{v'\}$
// v' and it's neighbors form a *cluster*
9: begin parallel across all processors
10: $\forall u', w' \in C_{v'}$ do
// each processor is assigned the same number of
// elements of $C_{v'}$
11: insert (u', w') in E'
12: insert (u, w) in E
13: end do
14: end parallel
15: delete v' from V'
16: begin parallel across all processors
//at processor k ($k = 0$ to $p-1$) process $|C|/p$ nodes
17: $flag_k$ = false // local flag
18: for $j = k \cdot (|C|/p)$ to $(k+1) \cdot (|C|/p) - 1$
//C_i is the ith element of C
19: if $C_{v'} \subset C_i$, $flag_k$ = true
20: end for
21: end parallel
//gather all $flag_i$ in parallel in $\log p$ time
22: global_flag = gather(AND, $flag$)
23: if global_flag == true
insert C_v in C
24: end while
25: return C

Figure 2. Parallel algorithm for triangulation and identifying maximal cliques

intersection property discussed in Section 2.3 ensures that clique C_i exists in step 4. The random variables associated with each clique along with the corresponding potential tables are listed in an array.

3.4 Potential Table Calculation

As mentioned in Section 2.3, potentials tables need to be generated for each node and each edge in the junction tree

IEEE COMPUTER SOCIETY

Algorithm: Construct(C)
Input: Set of Clusters $C = \{C_1, C_2, \ldots, C_t\}$ ($|C| = t$)
Output: Junction Tree $J(C, CE)$
1: begin parallel across all processors
//at processor k ($k = 0$ to $p-1$)
//process t/p nodes
2: for $j = k \cdot (t/p)$ to $(k+1) \cdot (t/p) - 1$
3: $CU_j = C_j \cap \{C_1 \cup C_2 \cup \ldots \cup C_{j-1}\}$
4: find C_i $(i < j)$ such that $CU_j \subseteq C_i$
//C_i is the parent of C_j
5: add (C_i, C_j) to CE
6: compute $C_i \cap C_j$, the separator set of (C_i, C_j)
7: end for j
8: end parallel

Figure 3. Parallel algorithm for constructing junction tree

J. Since each node in J represents a Clique of w nodes in the original Bayesian network, the potential table creation involves finding the joint probabilities of all those variables. Since each variable can take one of r values, there are r^w of the combined random experiment represented by each node in J. Similar argument holds for the edges in J, which each represent a set of nodes in the original Bayesian network. Our algorithm for the calculation of potential tables in shown in Figure 4.

3.5 Analysis

For the purpose of analysis consider a Bayesian network with n nodes, each with the maximum degree k. Therefore, each clique that induces a node in the junction tree J is at most of size $w(= k+1)$. We assume a p processor CREW PRAM [6] as our computation model.

Moralization: Steps 2 – 7 take $O(nk/p)$ time to construct an undirected version of G. Steps 8 – 13 take a further $O((k+k^2)n/p)$ time to moralize it. Note that the k^2 factor comes from step 11, since a node can have at most k parents. Overall, this stage takes $O(nk/p + nk^2/p)$ time.

Triangulation and Identifying Cliques: Steps 4 – 6 take $O(nk/p + \log p)$ time to find v'. Since we sort the cluster elements at the time of cluster creation, comparisons of clusters in step 19 takes $O(w)$ time only. Step 22 takes $\log p$ time to collect the flags. The total time complexity of this stage is $O(n^2k/p + n\log p)$.

Constructing the Junction Tree: For this step, we compute the edges of the junction tree in parallel. For computing each edge, we need to perform a set of set intersections and unions. Using sorted linked lists to represent sets, and using an n-bit vector to compute set intersection, step 3 can be performed in $O(nw)$ time. The computation in step 6 takes $O(w)$ time. The total time taken in this stage is $O(n^2w/p)$.

Algorithm: Potential($J(C, CE)$)
Input: Junction Tree $J(C, CE)$
Output: Set of Potential Tables T_i, one for each cluster and one for each edge
1: begin parallel across all processors
//at processor k ($k = 0$ to $p-1$)
//process n/p cliques
2: for $i = k \cdot (n/p)$ to $(k+1) \cdot (n/p) - 1$
3: Compute the potential table of $C_i \in C$
//above step fills in entries in a r^w table
4: end for
5: end parallel
6: begin parallel across all processors
//at processor k ($k = 0$ to $p-1$)
//process n-1/p edges
7: for $i = k \cdot ((n-1)/p)$ to $(k+1) \cdot ((n-1)/p) - 1$
8: Compute the potential table of $E_i \in CE$
//above step fills in entries in a r^{w-1} table
9: end for
10: end parallel
11: return (C, CE)

Figure 4. Parallel algorithm for potential table creation

Computing Potential Tables: Since the size of each clique is w, the maximum size of the *edges* can be $w-1$. Further, since there are n nodes and $n-1$ edges in J, it is evident that the first loop dominates the computation. Each execution of step 3 involves multiplying w probabilities for each of the r^w entries in the potential table for each C_i. Hence the total time taken by each processor in Steps 1 – 5 is $O(wr^w \cdot n/p)$.

To summarize, the combined complexity of all four stages of our parallel version of the algorithm is $O(nk^2/p + n^2w/p + wr^wn/p + n\log p)$. Noting that $w, k < n$, this can be further simplified to $O(n^2w/p + wr^wn/p + n\log p)$.As can be seen, the synchronization/communication cost is a very small part of the total time complexity.

Scalability Analysis: To analyze the scalability of the parallel algorithm, we consider the following two cases –
Case 1: In graphs where $n > r^w$, the complexity of the parallel algorithm is $O(n^2w/p + n\log p)$. For $1 < p < nw/\log n$, n^2w/p is $O(n\log n)$, and $n\log p$ is also $O(n\log n)$ (actually $O(n\log n - n\log\log n)$). Hence the algorithm is scalable for up to $nw/\log n$ processors in this case, with complexity of $O(n^2w/p)$.
Case 2: In graphs where $r^w > n$, the complexity of

IEEE COMPUTER SOCIETY

the parallel algorithm is $O(wr^w n/p + n \log p)$. For $1 < p < wr^w / \log n$, $wr^w n/p$ is $O(n \log(wr^w))$, and $n \log p$ is also $O(n \log(wr^w))$ (actually $O(n \log(wr^w) - n \log \log n)$). Hence the algorithm is scalable for up to $wr^w / \log n$ processors in this case, with complexity of $O(wr^w n/p)$.

The above analysis proves that the algorithm clearly scales for values of p between 1 and $nw / \log n$.

Sequential Complexity: For comparison, we compute the time taken if all the stages of this algorithm were performed sequentially. The complexity analysis of the original sequential algorithm is similar to that of the parallel version, but for the parallel loops and the scatter/gather operations. In that case, Moralization takes $O(nk + nk^2)$ time and computing the potential tables takes $O(wr^w n)$ time. Similarly, the junction tree creation step would take $O(n^2 w)$ time if done sequentially. Since the triangulation with clique selection stage use communication across processors, the analysis differs slightly. For example, Steps 16 – 23 in Figure 2 now take $O(nw)$ time, thus taking the sequential time complexity of this to $O(n^2 k + nw)$. The combined sequential complexity of the four stages is thus $O(nk^2 + n^2 w + wr^w n)$. Noting that $k, w < n$, this can be further reduced to $O(n^2 w + wr^w n)$.

3.6 Exact Inference

Our previous work [10] illustrates a parallel algorithm to compute exact inference given a junction tree with potentials calculated for each node. The global propagation of the potentials of each node is part of the algorithm discussed in [10]. This paper deals with the parallelization of the Junction Tree Generation Step. The output of this step can then be utilized by the algorithm developed in [10] to compute exact inference for the Bayesian network we started with.

4 Experimental Results

4.1 Computing Facilities

We used three machines for our experiments. The Shared Memory Processor at USC is a SunFire 15K system. It has 64 UltraSPARC III 1.2 GHz processors and a 150 MHz Sun Fireplane redundant 18X18 data, address, and response crossbar interconnect. The operating system is SUN OS 5.9 with MPICH for communication. For larger experimental runs we accessed the computing resources at the San Diego Supercomputer Center. One of the machines is a DataStar cluster with 1024 IBM P655 nodes running at 1.5 GHz with 2 GB of memory per processor. The theoretical peak performance of this machine is 15 TeraFLOPS. Furthermore each node is connected to a GPFS (parallel file system) through a fiber channel. The second large machine we ran the experiments on is the Teragrid machine - SGI Altix. It has 1024 1.6 GHz Itanium 2 processors with 1024 Gbytes of shared memory. It runs SGI ProPack 3.4 with OpenMP.

4.2 Experiments

We used three types of input Bayesian networks: linear, balanced, and arbitrary. All graphs were of 1024 nodes, and we ran the experiments with variables having 2, 4, and 16 states. The node in(out) degrees for the three networks were 1(1), 1(2), and 5(5) respectively.

At the time of writing, our implementation used simplified techniques for rapid program development. The OpenMP directive used were `omp parallel` and `omp parallel for`. We used an adjacency matrix to describe the belief network. The resulting junction tree data structure stores the following parameters –

- The indices of the nodes in each clique
- The clique connectivity matrix
- The number of nodes in each clique
- The ranges of the nodes in each clique
- The entries in the potential tables
- The nodes in the separator sets
- The separator set potential tables

Distribution of execution time
Results from our experiments with PNL on a single machine are shown in Figure 5. We note that the potential table creation step constitutes a major part of the sequential execution time, as expected with the dominating r^w factor. However, the other steps taken together can constitute for up to 50% time in some cases. This highlights the need to extract parallellism from the first three stages also.

Baseline comparisons
The *Probabilistic Network Library* (PNL) [5] created by Intel is a full function, free, open source, graphical models library released under a BSD style license. It is part of the collection of Intel's Open Source Libraries intended to promote and foster the use of computational intelligence and optimization techniques in society. The commonly used version is written in C++ and is sequential. PNL also has a parallel version which uses OpenMP. We conducted experiments to explore the scalability of the exact inference algorithm using the PNL library. The results are shown in Figure 6. Clearly, the parallel version of the PNL library does not scale well. We believe that this is due to the fact

IEEE COMPUTER SOCIETY

that simple parallellization of serial code using openMP does not fully exploit the parallellism in the problem.

Scalability

Parallel Junction Tree Generation: We implemented the junction tree generation algorithm in OpenMP. The execution times for the linear, balanced and arbitrary junction trees of 1024 nodes are shown in Figures 7 and 8. We obtain speedups of 31, 29 and 24 for linear, balanced and random Bayesian Networks on 128 processors. Our experiments show that our implements scale very well with increase in the number of processors.

Exact Inference: We have a complete parallel solution for exact inference from a Bayesian network. The solution is scalable and performs inference in logarithmic time. The techniques used are described in [10], Figure 9 shows the overall execution time for exact inference on linear, balanced and random Bayesian networks.

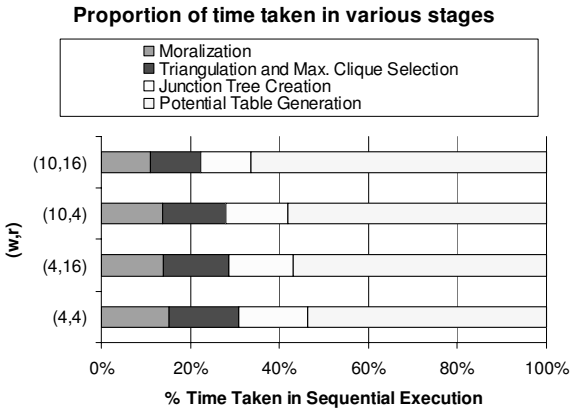

Figure 5. Distribution of execution time in sequential execution

5 Conclusion

We have presented an implementation of a parallel algorithm for junction tree creation. It accepts an arbitrary Bayesian network as input and outputs a junction tree after going through the steps of moralization, triangulation and maximum cardinality search. Our parallel algorithm has a parallel time complexity of $O(n^2w/p + wr^w n/p + n \log p)$. Our algorithm is scalable over $1 \leq p \leq nw/\log n$. We show by our experiments that the Intel PNL library does not scale while our implementation scales very well. We note that although the easily parallelized potential table generation step dominates the execution time due to being exponential in the clique width w, the first three stages are also important in certain networks. Our future work will involve the extraction of more parallelism from the first three stages, as well as parallel techniques to reduce the width of the cliques generated.

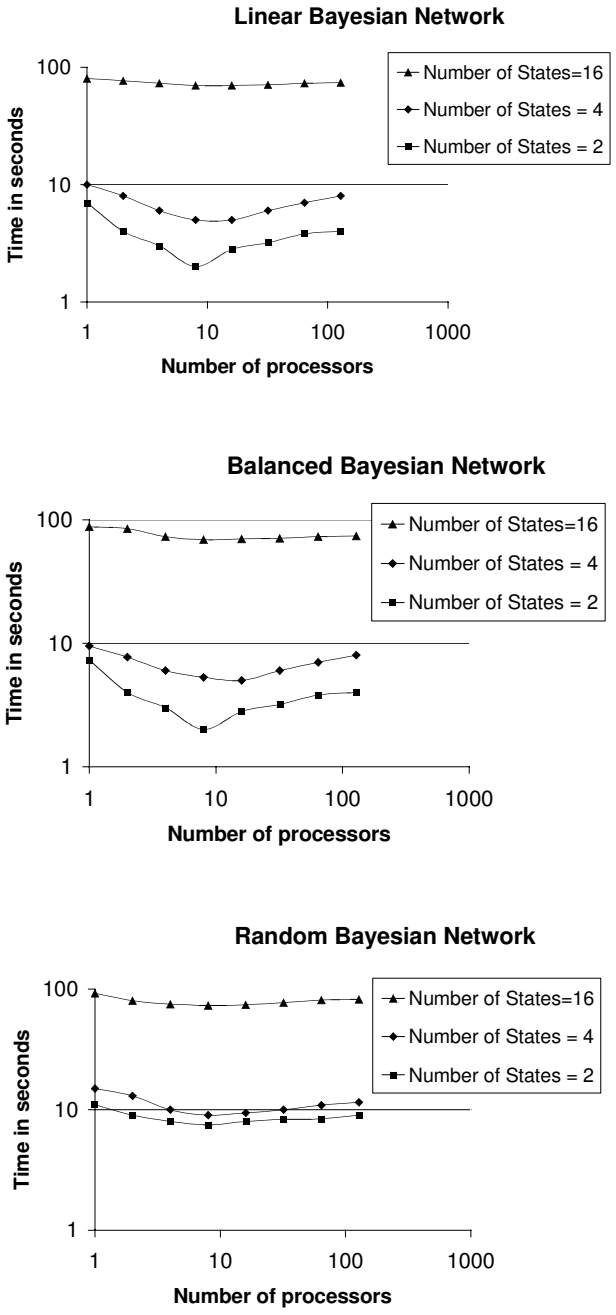

Figure 6. Execution times of PNL on DataStar

IEEE COMPUTER SOCIETY

References

[1] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Conference on Uncertainty in Artificial Intelligence (UAI-96)*, USA, 1996.

[2] B. D'Ambrosio. Parallelizing probabilistic inference: Some early explorations. In *Conference on Uncertainty in Artificial Intelligence (UAI-92)*, pages 59–66, USA, 1992.

[3] F. J. Diez and J. Mira. Distributed inference in bayesian networks. In *Cybernetics and Systems 25(1)*, pages 39–61, January 1994.

[4] D. Heckerman. Bayesian networks for data mining. In *Data Mining and Knowledge Discovery*, 1997.

[5] Intel. *Probabilistic Network Library, http://www.intel.com/technology/computing/pnl/index.htm.*

[6] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[7] A. V. Kozlov. Parallel implementations of probabilistic inference. In *Computer 29(12)*, pages 33–40, Dec 96.

[8] A. V. Kozlov and J. P. Singh. A parallel lauritzen-spiegelhalter algorithm for probabilistic inference. In *SC 94*, pages 33–40, 1994.

[9] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. In *J. of the Royal Statistical Society*, pages 157–224, 1988.

[10] V. K. Namasivayam and V. K. Prasanna. Scalable parallel implementation of exact inference in bayesian networks. In *Twelfth International Conference on Parallel and Distributed Systems (ICPADS)*, July 2006.

[11] D. M. Pennock. Logarithmic time parallel bayesian inference. In *Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 431–438, USA, July 1998.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, 1995.

[13] E. Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller. Rich probabilistic models for gene expression. In *ISMB*, pages 243–252, 2001.

[14] R. D. Shachter, S. K. Andersen, and P. Szolovits. Global conditioning for probabilistic inference in belief networks. In *Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 514–522, 1994.

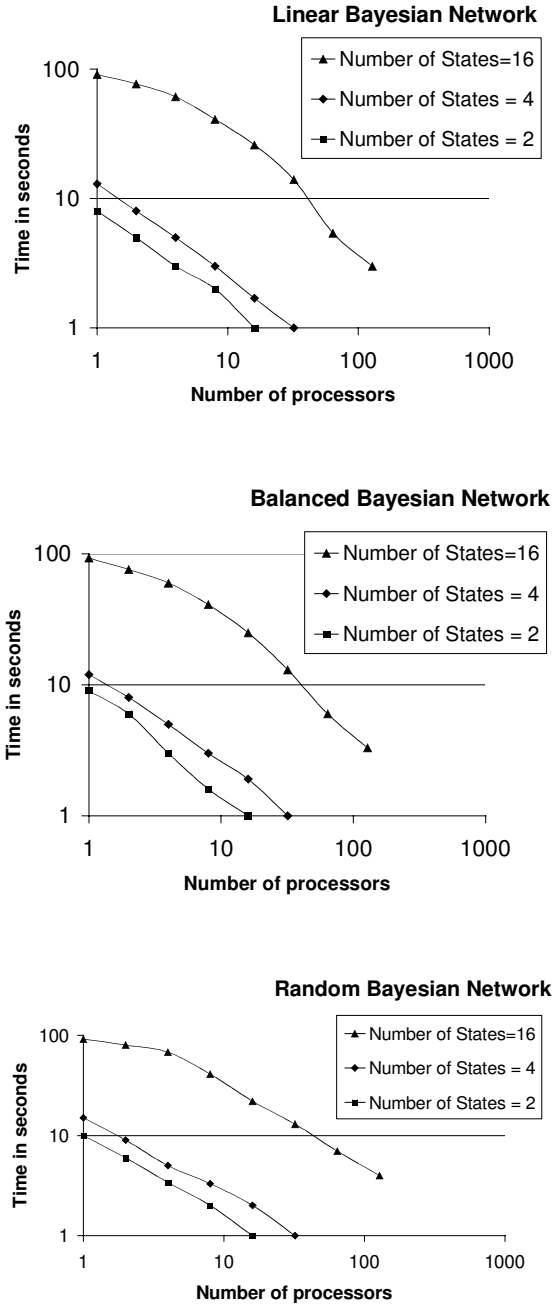

Figure 7. Execution times using OpenMP on DataStar

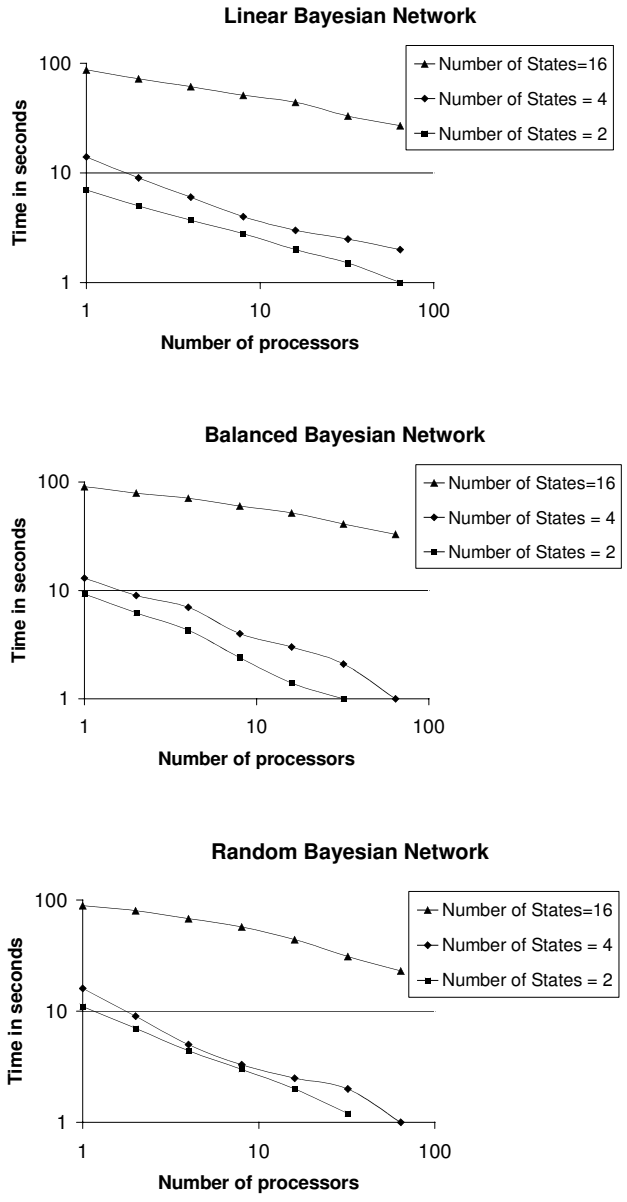

Figure 8. Execution times using OpenMP on Altix SGI

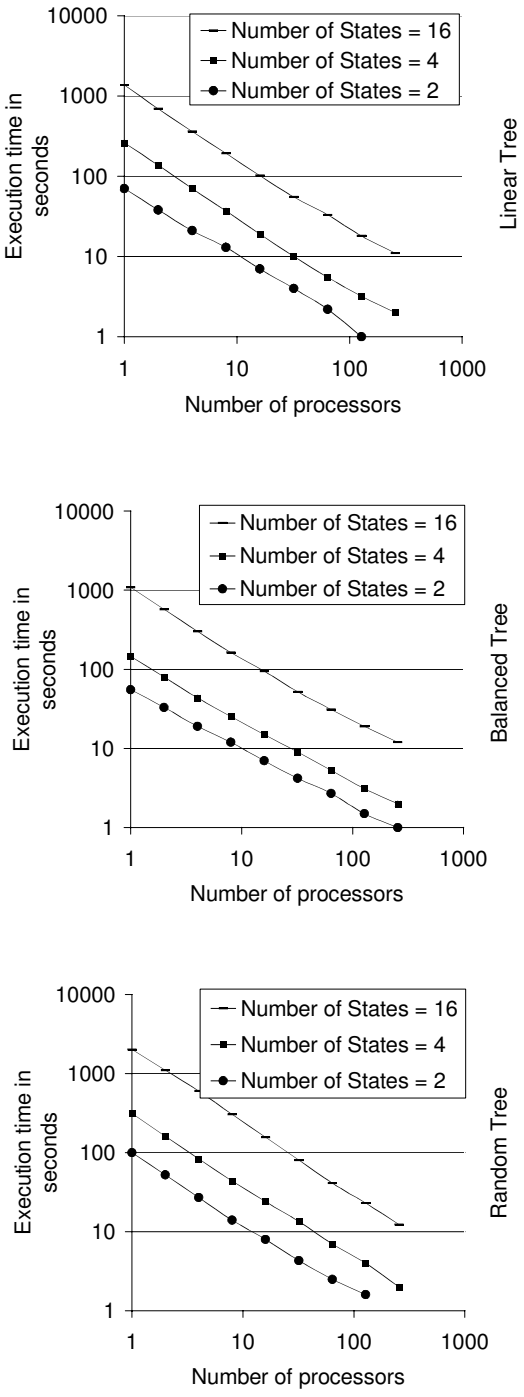

Figure 9. Overall execution times for exact inference using OpenMP on DataStar

IEEE COMPUTER SOCIETY