

Mapping and Parallel Implementation of Bayesian Belief Networks

Nina Saxena
Dept. Compu. Sci. & Engg.
Univ. of South Florida
Tampa, FL 33620

Sudeep Sarkar
Dept. Compu. Sci. & Engg.
Univ. of South Florida
Tampa, FL 33620
{saxena,sarkar,ranganat}@cs.csee.usf.edu

N. Ranganathan
Cntr. for Microelectronics Rsch.
Univ. of South Florida
Tampa, FL 33620

I. Introduction

Traditionally, probabilistic reasoning has been plagued by its computational intractability in real situations. Complete joint probabilistic specifications are not possible in most situations and very simplistic probabilistic models which are computationally inexpensive do not capture the complexities of a real life problem. With the advent of Bayesian networks, we have seen a renewed interest in probabilistic reasoning. Bayesian networks provide us with a unique way of looking at a probability distribution in terms of a directed graph. Each node in the Bayesian network represents a random variable and the links denote direct dependencies between random variables. The links are quantified with the conditional probabilities. Not only does the graph structure allow a better visualization of the inherent dependencies among the random variables but the network translates into a distributed computational structure. Bayesian networks are presently being used in computer vision, artificial intelligence, medicine, CAM, troubleshooting and other applications wherein decisions are conditionally dependent on many controlling factors.

As we shall see, there is inherent parallelism and locality in updating a Bayesian network [2]. However, in spite of the distributed nature of the Bayesian network, to date its implementation has been mainly serial. In this paper we explore the mapping of a class of Bayesian networks called polytrees onto a hypercube parallel machine architecture. Issues such as synchronization, preventing deadlock without much busy-waiting, load balancing and preserving functional integrity are discussed. The proposed mapping is deadlock free since all the messages are received and processed in the order of the structural hierarchy of the nodes in a tree. The mapping scheme maintains parent-child adjacency and single hop message passing throughout the computation. The scheme was implemented and verified on a 64 node nCUBE. The task allocation is static and is done at the beginning of the computation. The proposed scheme allows

⁰This research was supported in part by NSF Grants # CDA-9522265 and # IRI 9501932.

for efficient mapping of arbitrarily large trees onto a fixed size hypercube. We found that the overall speed up corresponds to the height of the tree.

Application-specific task allocation is important. Each parallel algorithm can have different task schedules and the efficiency depends much on the task allocation. It is therefore absolutely essential to have a task mapping strategy that gives maximum benefit in terms of speed and cost. The literature on this topic is rich. Some algorithms map tasks based on the critical path and the completion time constraints as in [4]. Other mapping criteria that are used are the *Dilation Bound* and the *Expansion Ratio* as mentioned in [6]. *Dilation* is defined as the length of any edge in the graph after mapping. *Expansion Ratio* is the ratio of the total number of functional links between processors to the number of edges in the graph.

II. Bayesian Network Updating

Each node in a Bayesian network represents a random variable from the problem domain. The directed links between the node denote direct dependencies between nodes and are sometimes inferred using cause and effect principles. At each node X we store the conditional probabilities of possible values ($X = x$) given the states of the parents $\{U_1, U_2, \dots, U_n\}$. In practical terms we store the table $\{P(X = x | U_1 = u_1, U_2 = u_2, \dots, U_n)\}$ for all possible $\{x, u_1, u_2, \dots, u_n\}$. For binary random variables, each element in this latter set of possible values can be either 0 or 1. At steady state, each node of the Bayesian network also stores a probability vector $\{BEL(X = x)\}$ which reflects the probability that $X = x$ given the present state of knowledge. Whenever new evidence arrives for a sets of nodes, we would like to update the *BEL* vectors at all other nodes in light of this new evidence. This is called updating a Bayesian network.

Updating a general Bayesian network with no topological restrictions is an NP-hard problem [3]. However, Pearl [1] showed that if the Bayesian network is

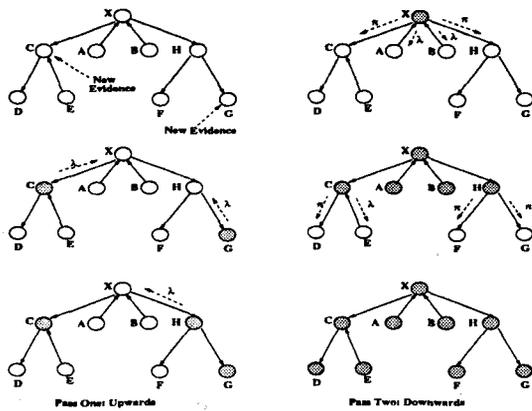


Figure 1: Flow of messages in a Bayesian network according to Peot and Shachter algorithm

a polytree¹ then there exists a very efficient probability updating algorithm which involves message (π and λ) passing among the random variables. The updating algorithm is $\mathcal{O}(mn)$ where n is the number of Bayesian nodes and m is the number of new evidence. Peot and Shachter [2] suggested a revised polytree algorithm in which the entire process of updating a Bayesian network can be completed in just three passes irrespective of the number of new evidence. The revised polytree algorithm is $\mathcal{O}(3n)$ where n is the number of Bayesian nodes.

The revised polytree scheme proceeds by identifying a pivot node which is usually the node with a large number of neighbors, e.g. the node marked X in Figure 1. Then conceptually we consider passing messages in this revised structure which is formed when we hang the polytree Bayesian network from the pivot node. This results in a tree structure (ignoring the Bayesian network link directions) as shown in Figure 1. We call this new tree structure the Peot-Shachter tree. Note that the parents of a node in the Peot-Shachter tree are different from the parents in the underlying Bayesian network. The resultant parents and children in the Peot and Shachter tree determine the directions of messages that are passed and the contents of the messages (λ and π) are determined by the Bayesian network's polytree structure.

The message passing scheme is simple. It involves three passes which proceeds in a vertical manner in the Peot-Shachter tree structure. Information from new evidences are absorbed as the messages proceed. Figure 1

¹A polytree is a tree where each node can have multiple children and multiple parents.

depicts the message passing scheme. The three passes are as follows:

- *Pass zero:* Starting from the pivot node a message passes downwards asking for messages from nodes with new evidence. This process initiates the actual updating process.
- *Pass one:* On receiving a pass zero request, a node with new evidence passes a message upwards to its parents in the Peot-Shachter tree. The content of the message is determined by the polytree Bayesian network status of the Peot-Shachter tree parent. Thus, if the parent of the present node in the Peot-Shachter tree is a child of the present node in the Bayesian network then we compute a π message or else we compute a λ message.
- *Pass Two:* Once all the message reach the root, the down pass begins in which all the nodes of the tree are updated starting from the root and traveling down to the leaves.

III. Mapping and Implementation

For implementing Bayesian Networks on parallel machines, the mapping scheme not only has to be efficient with respect to task scheduling and load balancing but also with respect to the inter-node communication problems required in the network. Also, when the number of nodes in the Bayesian network is much larger than the number of processors in the parallel machine, the task allocation must still maintain all requirements for efficient processing. Note that in the Peot-Shachter tree each node of the Bayesian network has only one parent. Thus, we can name each node in breadth first order and assign the levels.

To reduce the communication time and the number of *relay processors*, it is essential to maintain Peot-Shachter parent-child adjacency when mapping the tree, as much as possible, on to the parallel machine. A good way of achieving this would be to assign the root (*level 0*) of the tree always to *Node 0* of the hypercube and then assign its children (*level 1*) to its neighbors in a *round robin* fashion. This method is continued to map the rest of the levels in a similar fashion till the entire tree is mapped.

To maintain *load balancing*, the scheme should be adjusted to ensure uniform variation of tree nodes over hypercube nodes. It is done by changing the starting bit also in a *round robin* fashion at every level. This would mean that for the k -th level of a tree on an m degree

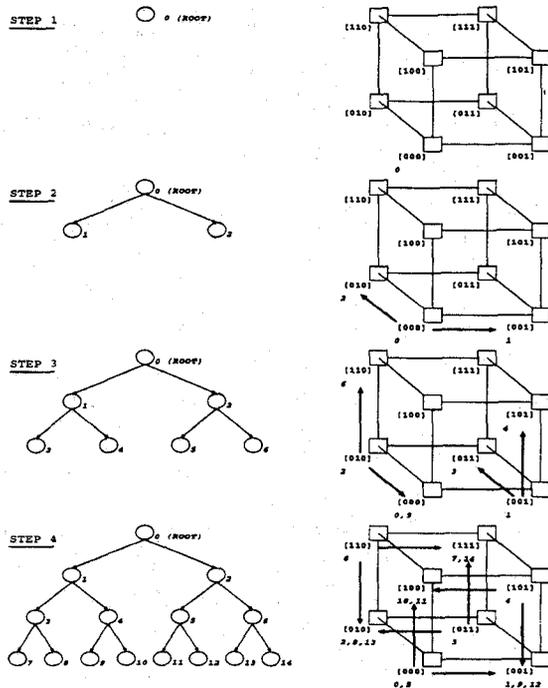


Figure 2: Load balanced adjacency mapping

hypercube, the starting bit would be the $(k \bmod m)$ -th bit. Figure 2 illustrates load-balanced, round robin mapping. The small circles represent tree nodes, the numbers near them, that are not within brackets, represent the corresponding node number in the tree itself. The numbers within the square brackets give the binary encoding of the hypercube node in which the specific tree node is stored.

In a typical message passing application, one of the greatest problems is that of deadlock. Moreover, since all processors work in parallel, it is also very important to preserve the sequence in which information travels. An out-of-turn message can lead to erroneous computation and propagation of errors. At the same time, since messages queue up in buffers, a message that is required first may arrive after another that is required later and cause a deadlock. This can be eliminated by handshaking and sequence ordering of the messages.

IV. Experimental Results and Performance Comparison

The scheme was implemented on a 64 node Silicon Graphics hypercube with IRIX 4.0.1 operating system².

²The authors would like to thank Dr. Sartaj Sahni, Dr. Sanjay Ranka and James F. Hranicky of the University of Florida.

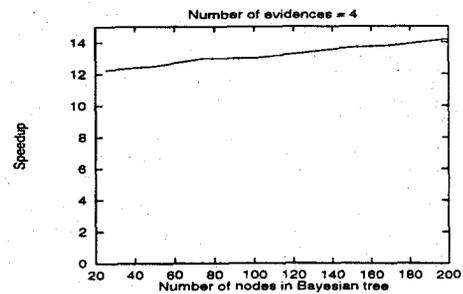


Figure 3: S vs n for 5d cube

The speedup due to the hypercube dimension and the Peot-Shachter tree height was observed using a number of Bayesian polytrees generated by a random Bayesian tree generator. The random generator was developed specifically for verification of the scheme. The Bayesian polytrees were generated with the characteristics needed for real time applications.

Figure 3 gives the actual experimental speedup obtained for trees of various sizes for cube of dimension 5. Figure 4 shows the speedup obtained by mapping a tree of 50 nodes onto cubes of various dimensions. Figure 5 shows the speedup obtained for different trees obtained by keeping the number of nodes constant but varying the height of the Peot-Shachter tree. The speedup remains more or less constant over the number of nodes in a Bayesian tree. The anomalies seen in the curves are due to the fact that the actual number of child nodes, for each node, may vary from tree to tree and node to node. Hence, the speedup sometimes fluctuates depending on the actual distribution of nodes in a tree. There is a decline in speedup as the height of the Peot-Shachter tree is increased, as shown in Figure 4. The rate of increase of speedup with dimension, shown in Figure 3, is not linear with respect to the number of processors, since Bayesian polytree updating communication intensive and simultaneous computation is not always possible due to the inherent hierarchy in the Peot-Shachter tree. We have found these empirical behaviors to follow trends predicted by the analytical estimates [8]³

The actual timings obtained for each of these operations are as follows. The time taken for computations for the λ messages to be sent ranges between 7778 X 50 nsec and 7788 X 50 nsec. Similarly, for π messages sent, the time is usually between 22774 X 50 nsec and 22813 X

Gainesville, for their suggestions and help during the course of implementation on the nCUBE.

³We omit the theoretical analysis of the mapping algorithm due to space constraints.

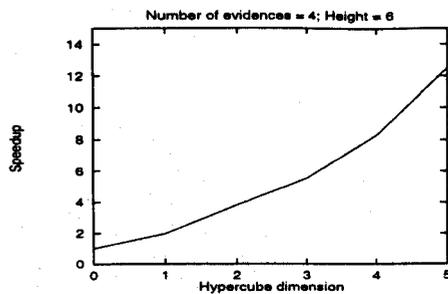


Figure 4: S vs d for tree of 50 nodes

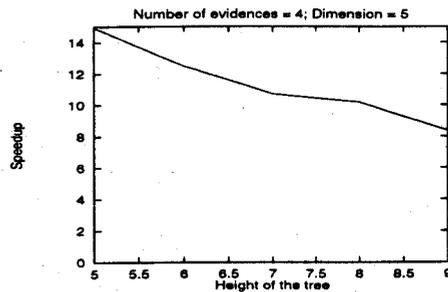


Figure 5: S vs h for tree of 50 nodes

50 nsec. Once the messages are received, the program takes from 12556 X 50 nsec to 12579 X 50 nsec for belief updation due to the λ messages and from 35062 X 50 nsec to 35084 X 50 nsec for belief updation due to the π messages. Each message passing takes constant amount of time too, since only a single hop is required in every case. In addition to this, each message passing requires about 4 clock cycles. Hence, in any cube of dimension 1 or more, this time gets included too. However, the nCUBE, being a loosely coupled message passing machine, may take as much as 350000 X 50 nsec to pass a single message from one processor to another due to the delays encountered because of machine interrupts, user interrupts and other jobs.

It was observed that maximum speedup occurs when (i) the number of children of the nodes is close to a multiple of the dimension of the hypercube, or (ii) the number of tree nodes is larger than the number of processors, or (iii) the tree has a slightly random nature. The random trees map with greater load balancing than complete and fully balanced trees.

Table 1 summarizes the ways in which this work employs a strategy that is superior to the current task allocation strategies, in mapping the Bayesian tree onto the parallel machine. Scheme 1 refers to the method discussed in [6], scheme 2 to that in [5] and scheme 3 to the one in [7].

Method	Hops	Processors	Sync.
1	Multiple	≤ 1	Not reqd.
2	Multiple	≤ 1	Not reqd.
3	Multiple	≥ 1	Required
This work	Single	1	Not reqd.

Table 1: Comparison with other mapping approaches

V. Conclusions

We presented an efficient technique for mapping arbitrarily large Bayesian belief networks on hypercubes with deadlock-free implementation. We showed that the speedup does not vary with the number of nodes in the Bayesian network and is limited by the height of the Peot-Shachter tree which is obtained by hanging the Bayesian polytree by a pivot node. We also found that the overhead in implementing Bayesian networks on parallel machines like hypercubes can be large because of the communication intensive nature of the network.

VI. References

- [1] Judea Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*, Morgan Kaufmann Publishers, first edition, 1988.
- [2] M. A. Peot and R. D. Shachter, "Fusion and Propagation with Multiple Observations in Belief Networks (Research Note)", *Artificial Intelligence* 48, pp 299-318, 1991.
- [3] G. Cooper, "Computational complexity of probabilistic inference using Bayesian belief networks (Research Note)", *Artificial Intelligence* 42, pp 393-405, 1990.
- [4] K. R. Pattipati, T. Kurien, Rong-Tay Lee and P. B. Luh, "On Mapping a Tracking Algorithm Onto Parallel Processors", *IEEE Trans. on Aerospace and Electronic Systems*, Vol. 26, No. 5, pp 774-791, September 1990.
- [5] Satoshi Horiike, "A Task Mapping Method for a Hypercube", *Systems and Computers in Japan*, Vol. 22, No. 9, pp 14-22, 1991.
- [6] A. Kavianpour and N. Bagherzadeh, "A Systematic Approach for Mapping Application Tasks in Hypercubes", *IEEE Transactions on Computers*, Vol. 42, pp 742-746, June 1993.
- [7] T. Kawaguchi, Y. Tamura and K. Utsumiya, "A Task Mapping Algorithm for Linear Array Processors", *IEICE Transactions on Information and Systems*, Vol. E77-D, No. 5, pp 546-554, May 1994.
- [8] N. Saxena, "Mapping and Parallel Implementation of Bayesian Belief Networks," M.S. Thesis, Department of Computer Science and Engineering, University of South Florida, Tampa, Dec. 95.