Distributed Evidence Propagation in Junction Trees

Yinglong Xia
Computer Science Department
University of Southern California
Los Angeles, CA 90089, U.S.A.
Email: yinglonx@usc.edu

Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089, U.S.A.
Email: prasanna@usc.edu

***Abstract*—Evidence propagation is a major step in exact inference, a key problem in exploring probabilistic graphical models. In this paper, we propose a novel approach for evidence propagation on clusters. We decompose a junction tree into a set of subtrees, and then perform evidence propagation in the subtrees in parallel. The partially updated subtrees are merged after evidence collection. In addition, we propose a technique to explore tradeoff between overhead due to startup latency of message passing and bandwidth utilization efficiency. We implemented the proposed method on state-of-the-art clusters using MPI. Experimental results show that the proposed method exhibits superior performance compared with the baseline methods.**

***Keywords*-exact inference; junction tree; parallel computing; cluster;**

I. Introduction

A full joint probability distribution can be used to model any real-world system. However, such a distribution increases dramatically with the number of variables in the distribution. *Bayesian networks* greatly reduce the size of the joint probability distributions by utilizing conditional independence relationships among the variables. Bayesian networks have found applications in a number of domains, including medical diagnosis, consumer help desks, data mining, genetics, etc. [1], [2].

Evidence propagation in a Bayesian network is the computation of the updated conditional distribution of the variables in the Bayesian network, given a set of *evidence* variables as the knowledge to the network. The most popular exact inference algorithm for multiply connected networks converts a Bayesian network into a *junction tree*, and then performs evidence propagation in the junction tree [3], [4]. The complexity of exact inference increases dramatically with the various parameters of junction trees, such as the number of clique, the clique width and the number of states of the random variables. In many cases exact inference must be performed in real time.

Designing efficient parallel algorithms for evidence propagation must take into account the characteristics of the platform. Most supercomputers are clusters of processors, where the communication time consists of startup time known as *latency* and data transfer time. On a cluster with dual quadcore AMD 2335 processors connected by Infiniband (IB) cables, we observed that the startup latency for a single message passing is 3.31 microseconds, equivalent to completing 211×10^3 floating point operations on a compute node. On the other hand, the aggregate memory of a cluster increases linearly with the number of processors. Thus, for algorithm design on clusters, we must minimize the overhead due to startup latency in communication even by duplicating some data.

Our contributions include: (1) Heuristic to decompose a junction tree into subtrees with approximately equal weights so as to perform evidence propagation in the subtrees in parallel, (2) a technique to merge partially updated subtrees, (3) a method to explore the tradeoff between overhead due to startup latency and bandwidth utilization efficiency in communication, (4) implementation and experimental evaluation on state-of-the-art clusters.

The rest of the paper is organized as follows: In Section II, we review the background. Section III discusses related work. Our proposed method is discussed in Section IV. We show experimental results in Section V and conclude the paper in Section VI.

II. Background

A. Bayesian Network and Junction Tree

A *Bayesian network* is a probabilistic graphical model that exploits conditional independence to represent compactly a joint distribution. Figure 1 (a) shows a sample Bayesian network, where each node represents a random variable. Each edge indicates the probabilistic dependence relationships between two random variables. Notice that these edges can *not* form directed cycles. Thus, the structure of a Bayesian network is a *directed acyclic graph* (DAG). The *evidence* in a Bayesian network is the variables that have been instantiated [3], [4].

Traditional exact inference using Bayes' theorem fails for networks with undirected cycles [3]. Most inference methods for networks with undirected cycles convert a network to a cycle-free hypergraph called a *junction tree*. We illustrate a junction tree converted from the Bayesian network in Fig. 1, where all undirected cycles in are eliminated. Each vertex in Fig. 1(b) contains multiple random variables from the

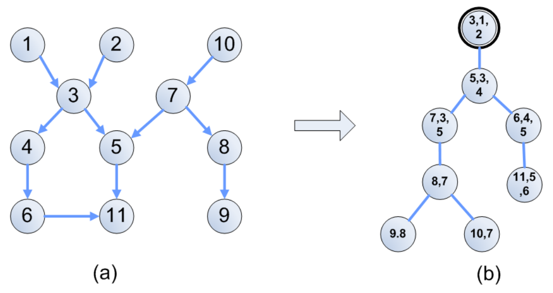


Figure 1. (a) A sample Bayesian network and (b) the corresponding junction tree.

Bayesian network. For the sake of exploring evidence propagation in a junction tree, we use the following notations. A junction tree is defined as $J = (\mathbb{T}, \hat{\mathbb{P}})$, where $\mathbb{T}$ represents a tree and $\hat{\mathbb{P}}$ denotes the parameter of the tree. Each vertex $\mathcal{C}_i$, known as a clique of J, is a set of random variables. Assuming $\mathcal{C}_i$ and $\mathcal{C}_j$ are adjacent, the *separator* between them is defined as $\mathcal{C}_i \cap \mathcal{C}_j$. $\hat{\mathbb{P}}$ is a set of *potential tables*. The potential table of $\mathcal{C}_i$, denoted $\psi_{\mathcal{C}_i}$, can be viewed as the joint distribution of the random variables in $\mathcal{C}_i$. For a clique with w variables, each having r states, the number of entries in $\mathcal{C}_i$ is r^w.

B. Evidence Propagation

In evidence propagation, we update a junction tree in two stages, i.e., *evidence collection* and *evidence distribution*. In evidence collection, evidence is propagated from the leaves to the root in topological order, where each clique $\mathcal{C}$ updates $\psi_\mathcal{C}$ using the separators between $\mathcal{C}$ and its children. Then, $\mathcal{C}$ updates the separator between $\mathcal{C}$ and its parent using the updated potential table $\psi^*_\mathcal{C}$. Evidence distribution is as the same as collection, except the evidence propagation direction is from the root to the leaves.

In a junction tree, evidence is propagated from a clique to its neighbors as follows: Assuming evidence is $E = \{A_i = a\}$ and $A_i \in \mathcal{C}_\mathcal{Y}$, E is *absorbed* at $\mathcal{C}_\mathcal{Y}$ by instantiating the variable A_i and renormalizing the remaining variables of the clique. The evidence is then propagated from $\mathcal{C}_\mathcal{Y}$ to all adjacent cliques $\mathcal{C}_\mathcal{X}$. Let $\psi^*_\mathcal{Y}$ denote the potential table of $\mathcal{C}_\mathcal{Y}$ after E is absorbed, and $\psi_\mathcal{X}$ the potential table of $\mathcal{C}_\mathcal{X}$. Mathematically, evidence propagation is represented as [3]:

$$\psi^*_\mathcal{S} = \sum_{\mathcal{Y} \backslash \mathcal{S}} \psi^*_\mathcal{Y}, \quad \psi^*_\mathcal{X} = \psi_\mathcal{X} \frac{\psi^*_\mathcal{S}}{\psi_\mathcal{S}} \tag{1}$$

where $\mathcal{S}$ is the separator between cliques $\mathcal{X}$ and $\mathcal{Y}$; $\psi_\mathcal{S}$ ($\psi^*_\mathcal{S}$) denotes the original (updated) potential table of $\mathcal{S}$; $\psi^*_\mathcal{X}$ is the updated potential table of $\mathcal{C}_\mathcal{X}$. The computations among cliques tables in Eq.(1) are called *node level primitives*, including potential table multiplication, division, extension and marginalization [5].

III. Related Work

There are several works on parallel exact inference, such as Pennock [4], Kozlov and Singh [6] and Szolovits [7]. However, some of these methods, such as [6], are developed for a class of Bayesian networks, such as Polytrees. Our proposed method can be used for arbitrary Bayesian networks. The execution time of some other methods, such as [4], is proportional to the number of cliques with evidence variables. Our proposed method does not depend on the number of evidence variables. In [5], the node level primitives are parallelized using message passing to explore data parallelism. In this paper, we investigate task level parallelism so as to handle junction trees with limited level parallelism. A junction tree decomposition method is provided in [8] to decompose a junction tree into chains. Unlike [8], we decompose a junction tree into subtrees in this paper so as to reduce clique duplication. In addition, we investigate tradeoff between the number of communication steps and bandwidth utilization efficiency.

IV. Distributed Evidence Propagation Based on Decomposition

A. Overview

We decompose a junction tree into a series of *subtrees*, each consisting of one or more root-leaf paths in the input junction tree (see Figure 2). Note that the decomposition is different from conventional tree *partition* that divides a given tree without duplicating any node [9]. A junction tree can be decomposed at various granularity levels to generate different number of subtrees. Given the number of processors P in a cluster, we decompose a given junction tree into P subtrees of approximately equal task weights, so that each processor can host a distinct subtree. In this paper, we assume the number of leaf cliques of the input junction tree is greater than the number of available processors.

Since cliques may be duplicated in several subtrees, we must *merge* partially updated cliques in the subtrees to obtain fully updated cliques. For example, in Figure 2(c), the root exists in all the three subtrees. After evidence collection, the root in each subtree is partially updated, since the root in the first subtree cannot collect evidence from Cliques 3, 5, 6, 7 and 10. Thus, we must merge the duplicated cliques. Merging cliques requires communication among the processors. Our proposed method guarantees all the duplicated cliques can be merged in parallel. After clique merging, we perform evidence distribution in each subtree in parallel and obtain the final output of evidence propagation. Note that we do not need to merge cliques after evidence distribution.

B. Junction Tree Decomposition

We show our proposed heuristic for junction tree decomposition in Algorithm 1. This algorithm decomposes the input junction tree into P, $P \geq 1$, subtrees of approximately

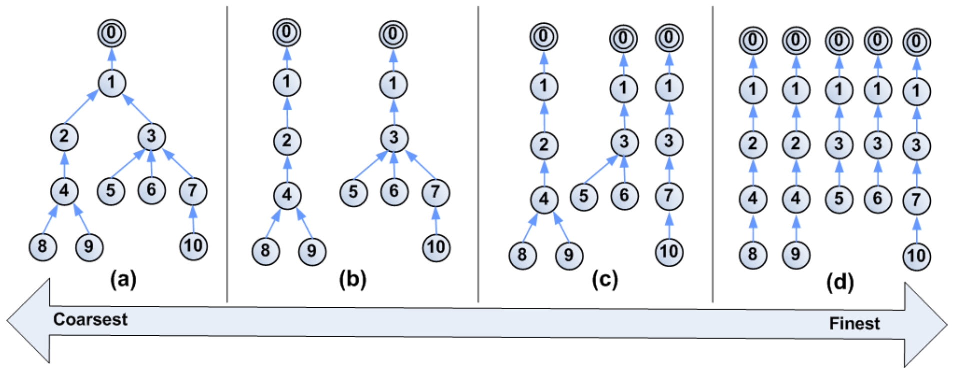


Figure 2. A junction tree in (a) can be decomposed into subtrees of various granularities shown in (b), (c) and (d). The arrows in the graph indicate evidence collection direction.

equal workload. By workload, we mean the overall execution time for processing the cliques in a subtree.

In Algorithm 1, we use the following notations. Weight $V_\mathcal{C}$ is the estimated execution time for updating a clique $\mathcal{C}$ in junction tree J. Given clique width $w_\mathcal{C}$, the number of children $d_\mathcal{C}$ and the number of states of random variables r, we have $V_\mathcal{C} = d_\mathcal{C} w_\mathcal{C}^2 r^{w_\mathcal{C}+1}$ [5]. The tolerance factor $\Delta \in [0,1]$ is a threshold that controls the load balance among the subtrees. Small Δ results in better load balance, but can lead to longer execution time for Algorithm 1. $ch(\mathcal{C})$ represents the children of $\mathcal{C}$. We use Path$(\tilde{r}, \mathcal{C})$ to denote the path from root $\tilde{r}$ to the parent of a clique $\mathcal{C}$, and Subtree$(\mathcal{C})$ the subtree rooted at $\mathcal{C}$. We use $R_\mathcal{C}$ to represent the overall weight of cliques in Path$(\tilde{r}, \mathcal{C})$ and $S_\mathcal{C}$ the overall weight of cliques in Subtree$(\mathcal{C})$. Thus, $(S_\mathcal{C} + R_\mathcal{C})$ gives the estimated execution time for updating a decomposed junction tree rooted at $\tilde{r}$. We assume there are P processors and the decomposed junction tree hosted by the i-th processor is J$_i$.

Algorithm 1 consists of three parts: The first part (Lines 2-8) populates $R_\mathcal{C}$ for each clique; the second part (Lines 9-15) populates $S_\mathcal{C}$. In the third part, we use a heuristic to decompose the input junction tree into P subtrees with approximately equal weight. Note that $(S_\mathcal{C} + R_\mathcal{C})$ is monotone nonincreasing from the root to leaf cliques. As the loop in Lines 17-26 iterates, more and more cliques are added to G in Line 19. In Lines 20-24, we assign the cliques to P groups using a heuristic, so that the overall weights of the decomposed junction trees are approximately equal. If the load balance is acceptable (Lines 25-26), the decomposed junction trees are generated using the cliques in Path$(\tilde{r}, \mathcal{C})$ and Subtree$(\mathcal{C})$. The complexity is $O(1)$ for Lines 1-2, and $O(N)$ for Lines 3-8 and 9-15. The time taken by Lines 16-26 depends on Δ and the input junction tree. In the worst case, each nonleaf clique is inserted into set G once. Thus, the complexity is also $O(N)$. Lines 27-28 take $O(P)$ time. Thus, the serial execution time of Algorithm 1 is $O(N)$.

C. Junction Tree Merging

A straightforward approach for junction tree merging is as follows: for each clique $\mathcal{C}$, access partially updated potential tables $\psi_\mathcal{C}$ from all the processors where $\mathcal{C}$ exists, and then combine them through a series of computations. Note that the potential table of cliques is generally much larger than the potential table of separators. Thus, although the above approach is doable, it requires intensive data transfer among the processors. This can adversely affect the performance.

An alternative approach for merging cliques is to utilize the separator potential tables, instead of the large clique potential tables. Consider the root clique in Figure 2(c) as an example. Note that evidence flows into the root through the separators between the root (i.e. Clique 0) and Clique 1 only. Thus, if we want to fully update $\psi_\mathcal{C}$, we simply transfer the potential tables of all the separators adjacent to $\mathcal{C}$. Note that the transferred data is much smaller than transferring $\psi_\mathcal{C}$. Let $\psi_\mathcal{C}^*$ denote the fully updated potential table for $\mathcal{C}$ and $\psi_\mathcal{C}^i$ the partially updated potential table in the i-th subtree. $\psi^i_{S_\mathcal{C}(j)}$ represents the j-th separator of $\mathcal{C}$ in the i-th subtree. Let $\tilde{\mathrm{S}}_\mathcal{C}^k$ denote all the separator potential tables required for obtaining $\psi_\mathcal{C}^*$ on the k-th processor:

$$\tilde{\mathrm{S}}_\mathcal{C}^k = \{\psi^i_{S_\mathcal{C}(j)}\},\ \ 0 \le i < P, i \ne k,\ \ 0 \le j < d_\mathcal{C}(i) \qquad (2)$$

where $d_\mathcal{C}(i)$ is the number of children of $\mathcal{C}$ in subtree J$_k$. Let $\hat{\psi}^i_{S_\mathcal{C}(j)}$ represent the potential table of separator $S_\mathcal{C}(j)$ obtained by marginalizing $\psi_\mathcal{C}$ before $\psi^i_{S_\mathcal{C}(j)}$ is used to update $\psi_\mathcal{C}$. We have the following formula $\mathcal{M}(\cdot)$ for merging any duplicated clique $\mathcal{C}$ in any subtree J$_k$:

$$\psi_\mathcal{C}^* = \mathcal{M}\ \left(\psi_\mathcal{C}^k, \tilde{\mathrm{S}}_\mathcal{C}^k\right) = \psi_\mathcal{C}^k \prod_{i,j} \frac{\psi^i_{S_\mathcal{C}(j)}}{\hat{\psi}^i_{S_\mathcal{C}(j)}} \qquad (3)$$

where $0 \le i < P, i \ne k,\ \ 0 \le j < d_\mathcal{C}(i)$.

During evidence collection, each processor checks if any local clique has been duplicated onto other processors. For each duplicated clique, we send the local separators corresponding to the clique to other processors, and also

Algorithm 1 Junction Tree Decomposition into Subtrees

```
Input: Junction tree J, clique weight $V_{\mathcal{C}}$, number of processors $P$, tolerance factor $\Delta$
Output: Subtrees $J_i$, $i = 0, 1, \ldots, P-1$
 1: let $R_{\mathcal{C}} = 0$, $S_{\mathcal{C}} = 0$, $\forall \mathcal{C} \in J$, $\tilde{r}$ be the root of J
    {Find weight of path from $\tilde{r}$ to parent of $S_{\mathcal{C}}$, $\forall \mathcal{C} \in J$}
 2: $Q = \{\tilde{r}\}$
 3: while $Q \neq \emptyset$ do
 4:   for all $\mathcal{C} \in Q$ do
 5:     $R_{\mathcal{C}'} = R_{\mathcal{C}} + V_{\mathcal{C}}$, $\forall \mathcal{C}' = ch(\mathcal{C})$
 6:   end for
 7:   $Q = \{\mathcal{C}' : \mathcal{C}' \in ch(\mathcal{C})$ and $\mathcal{C} \in Q\}$
 8: end while
    {Find weight of subtree rooted at $S_{\mathcal{C}}$, $\forall \mathcal{C} \in J$}
 9: $Q = \{$leaf cliques in J$\}$, $S_{\mathcal{C}} = V_{\mathcal{C}}, \forall \mathcal{C} \in Q$
10: while $Q \neq \emptyset$ do
11:   for all $\mathcal{C} \in Q$ do
12:     $S_{\mathcal{C}} = V_{\mathcal{C}} + \sum_{\mathcal{C}' \in ch(\mathcal{C})} S_{\mathcal{C}'}$
13:   end for
14:   $Q = \{\mathcal{C} : S_{\mathcal{C}} = 0$ and $S_{\mathcal{C}'} > 0, \forall \mathcal{C}' \in ch(\mathcal{C})\}$
15: end while
    {Decomposition}
16: $G = \{\tilde{r}\}$
17: repeat
18:   $\mathcal{C}_m = \max_{\mathcal{C} \in G}(R_{\mathcal{C}} + S_{\mathcal{C}})$ where $ch(\mathcal{C}) \neq \emptyset$
19:   $G = G \cup \{\mathcal{C}' : \mathcal{C}' \in ch(\mathcal{C}_m) \neq\} \backslash \{\mathcal{C}_m\}$
20:   let $G' = G$, $K_0 = K_1 = \cdots = K_{P-1} = \emptyset$
21:   while $G' \neq \emptyset$ do
22:     let $j = \arg\min_{i \in [0, P-1]} \left(\sum_{\mathcal{C}' \in K_i} (S_{\mathcal{C}'} + R_{\mathcal{C}'})\right)$
23:     $K_j = K_j \cup \{\max_{\mathcal{C} \in G'} \mathcal{C}\}$, $G' = G' \backslash \{\mathcal{C}\}$
24:   end while
25:   $K_{max} = \max_{i \in [0, P-1]} \left(\sum_{\mathcal{C}' \in K_i} (S_{\mathcal{C}'} + R_{\mathcal{C}'})\right)$,
      $K_{min} = \min_{i \in [0, P-1]} \left(\sum_{\mathcal{C}' \in K_i} (S_{\mathcal{C}'} + R_{\mathcal{C}'})\right)$
26: until $G = \{$leaf cliques in J$\}$ or
        $(K_{max} - K_{min})/(K_{max} + K_{min}) < \Delta$
27: for all $i = 0, 1, \cdots, P-1$ do
28:   $J_i = J \cap (\text{Path}(\tilde{r}, \mathcal{C}) \cup \text{Subtree}(\mathcal{C}))$, $\forall \mathcal{C} \in K_i$
29: end for
```

receive separators from other processors as well. Then, we use Eq. 3 to obtain the fully updated cliques. Note that we do not need to merge junction trees after evidence distribution.

D. Tradeoff between Startup Latency and Bandwidth Utilization Efficiency

During junction tree merging, the processors exchange locally updated separators by message passing. Assume the total amount of data transferred for junction tree merging is s and there are k communication steps executed in sequence. Given the aggregate network *bandwidth* B and the startup *latency* l for sending a message, an optimization approach for exchanging separators must minimize t_{comm} where,

$$t_{comm} = k \cdot l + \frac{s}{B} \tag{4}$$

Since l and B are constants for a given system, we must minimize k and s. We consider the following two cases for message communication.

(1) For each duplicated clique $\mathcal{C}$, a processor broadcasts the local separator potential tables of $\mathcal{C}$ to other processors containing $\mathcal{C}$, and receives the separator potential tables from other processors. Such an operation is known as *all-gather* in MPI [10]. In this case, all the transferred separators are necessary for junction tree merging. Thus, s is minimized. However, since a communication step is required for each duplicated clique, a decomposed junction tree with a large number of duplicated cliques can result in a large k in Eq. 4. For any junction tree with multiple leaf cliques, we always have $k > 1$.

(2) Each processor packages its local data to be sent into an array ordered by the target processor IDs, then in a single communication step, sends the data in such an array to their respective target processors. This operation is known as *all-to-all* in MPI [10], [11]. In this case, all the processors exchange data in a *single* communication step, i.e., $k = 1$. Thus, k is minimized. Note that each processor sends its local separators to all the other processors. In a specific receiver, the separators for merging cliques not hosted by the processor are not be used. Such separators are called *invalid data*. Therefore, due to transfer of invalid data, s in Eq. 4 is *not* minimized.

We develop a technique to explore the tradeoff between s and k. First, we represent the clique allocation and duplication using a *bitmap*, i.e., a boolean matrix where each row corresponds to a subtree and each column corresponds to a clique. We label the cliques using a breadth-first search (BFS) order starting from the root of the junction tree. Figure 3(a) illustrates a sample bitmap for the subtrees shown in Figure 2(c). Since we are only concerned with the duplicated cliques during junction tree merging, we remove the columns corresponding to the unduplicated cliques from the bitmap. The result is called *reduced bitmap* (see Figure 3(b)).

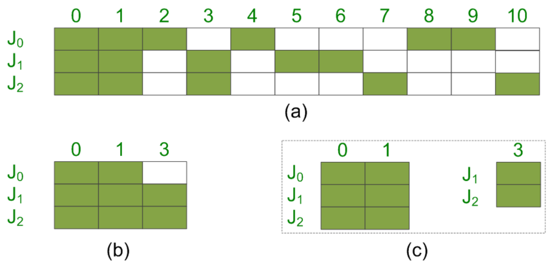


Figure 3. (a) A sample bitmap; (b) a reduced bitmap; (c) partitioned bitmaps.

Second, since the 0s in a bitmap correspond to invalid data transferred between processors, we partition a sparse bitmap into a set of smaller bitmaps using a heuristic, as shown in Figure 3(c). Each resulting bitmap is called a *sub-bitmap*. Any sub-bitmap with all entries equal to 0 can be eliminated. We define the *density* of a sub-bitmap as the percentage of the entries equal to 1. Given a sub-bitmap, the corresponding processors exchange data using an all-to-all communication step. The number of communication steps k is given by the number of sub-bitmaps. Denser sub-bitmaps lead to higher bandwidth utilization efficiency in all-to-all communication. Let d_j^i denote the clique degree of C_j in J_i. $|\psi_{S_j}^i|$ represents the size of a separator potential table in J_i. Assuming P_m processors are involved in the m-th sub-bitmap, the communication time is given by:

$$t_{comm} = k \cdot l + \frac{(\sum_{m=0}^{k-1} P_m - 1) \cdot \max_{0 \le i < P_m} \left(\sum_{j=0}^{|\mathrm{J}_i|} (d_j^i \cdot |\psi_{S_j}^i|) \right)}{B} \tag{5}$$

There are several ways to partition a bitmap. For example, we can construct a bipartite graph using the reduced bitmap and then identify the densely connected subgraphs by *graph clustering* [12]. A constraint for such clustering is that the duplicates of a clique must be assigned to the same sub-bitmap, so that we can obtain fully updated cliques. To the best of our knowledge, there is no known algorithm for graph clustering with such a constraint. We design a heuristic using the characteristics of the junction tree decomposition discussed in Section IV-B. The heuristic partitions a given reduced bitmap into a set of sub-bitmaps, while satisfying the above constraint.

We briefly discuss the proposed heuristic for bitmap partitioning. Since a bitmap represents subtrees decomposed from a given junction tree, a characteristic of junction tree decomposition is that the root of the junction tree is duplicated to all subtrees. Thus, the first column in the corresponding reduced bitmap must be *all* 1s. The cliques close to the root are likely to be duplicated to many subtrees. Therefore, we can start from the first column of a bitmap to find a dense sub-bitmap. Given the size of a separator potential table $|\psi_S|$ and the number of children of a clique d, we use the following metric to determine if a sub-bitmap is dense enough.

$$\frac{z \cdot d \cdot |\psi_S|}{\eta \cdot B} < l \tag{6}$$

where z is the number of entries equal to 0 and η, $\eta > 0$, is a constant called *tradeoff factor* with default value equal to 1. We use the default value. Users can reduce η to improve the bandwidth utilization efficiency, or increase it to reduce the overhead due to startup latency. Note that $(z \cdot d \cdot |\psi_S|)/B$ is an estimate of the time taken for transferring the invalid data. We compare this time with the latency l to determine if the sub-bitmap is dense or not.

Algorithm 2 Distributed Evidence Propagation

```
Input: Junction tree J, clique weight $V_{\mathcal{C}}, \forall \mathcal{C} \in \mathrm{J}$, number
    of processors $P$, tolerance factor $\Delta$, bandwidth $B$,
    startup latency $l$, tradeoff factor $\eta$
Output: Updated junction tree J
    {Initialization}
 1: $(\mathrm{J}_0, \cdots, \mathrm{J}_{P-1})$=Decompose$(\mathrm{J}, V_{\mathcal{C}}, P, \Delta)$ //Algorithm 1
 2: $\tilde{\Lambda}$ = reduced bitmap for $\{\mathrm{J}_0, \mathrm{J}_1, \cdots, \mathrm{J}_{P-1}\}$
 3: $\mathbb{M}$ = BitmapPartition$(\tilde{\Lambda}, B, l, \eta, \mathrm{J})$
 4: assign $\mathrm{J}_i$ and $\mathbb{M}$ to processor $p_i$, $\forall 0 \le i < P$
    {Parallel Evidence Propagation}
 5: for processor $p_i$, $i = 0, 1, \cdots, P-1$ pardo
 6:     EvidenceCollect$(\mathrm{J}_i)$
 7:     for all $\Lambda \in \mathbb{M}$ do
 8:         $\mathrm{S}^i = \emptyset$
 9:         for all clique $\mathcal{C}$ in both $\mathrm{J}_i$ and $\Lambda$ do
10:             $\mathrm{S}^i = \mathrm{S}^i \cup \{\psi^i_{S_{\mathcal{C}}(j)}\}$, $0 \le j < d_{\mathcal{C}}(i)$
11:         end for
12:         All-to-all send $\mathrm{S}^i$ to $p_j$ and receive $\hat{\mathrm{S}}^j$,
            $\forall 0 \le j < P-1, j \ne i$
13:         Obtain fully updated cliques using $\hat{\mathrm{S}}^j$ //Eq. (3)
14:     end for
15:     EvidenceDistribute$(\mathrm{J}_i)$
16: end for
```

Using Eq. 6, we identify a dense sub-bitmap starting from the first column of a reduced bitmap Λ. Assume the sub-bitmap consists of the first j columns in Λ. The rest of the bitmap is partitioned into two sub-bitmaps, one consisting of the rows where the j-th column is 1, the other consisting of the remaining rows. We eliminate the columns and rows with all 0s from the resulting sub-bitmaps. If any sub-bitmap is not dense enough according to Eq. 6, we recursively partition the sub-bitmaps, until all sub-bitmaps are dense.

E. Distributed Evidence Propagation Algorithm

Based on the techniques discussed in Sections IV-B, IV-C and IV-D, we present the complete algorithm for distributed evidence propagation in junction trees in Algorithm 2. The input includes an arbitrary junction tree J, clique weights (i.e., estimated execution time) $V_{\mathcal{C}}$, some system parameters such as bandwidth B and startup latency l and a user threshold $\Delta \in [0, 1]$ for controlling the junction tree decomposition. We let $\Delta = 0.1$ in our experiments. The output is the updated junction tree J, where the evidence originally on any clique has been propagated to the rest of the cliques.

Algorithm 2 consists of initialization and parallel evidence propagation. Initialization can be done offline. Thus, we only provide sequential algorithm for these steps. In Line 1, we invoke Algorithm 1 in Section IV-B to decompose J into P subtrees. The corresponding bitmap is created in Line 2 and partitioned in Line 3. Line 4 assigns each subtree to a separate processor and broadcasts $\mathbb{M}$. In Lines 5-16, all processors run in parallel to process the subtrees. In Lines 6 and 15, each processor performs evidence collection and distribution in its local subtree using the sequential algorithm proposed in [3]. In Lines 7-14, for each bitmap in Λ, we perform all-to-all communication among the processors corresponding to the bitmap. In Line 14, each processor collects the relevant separators in S^i. The received data is used to obtain fully updated cliques in Line 13.

V. Experiments

A. Facilities

We implemented the proposed method using the Message Passing Interface (MPI) on the High-Performance Computing and Communications (HPCC) Linux cluster at the University of Southern California (USC) [13]. The HPCC cluster employs a diverse mix of computing and data resources, including 256 Dual Quadcore AMD Opteron nodes running at 2.3 GHz with 16 GB memory. This machine uses a 10 GB low-latency Myrinet backbone to connect the compute nodes. The cluster runs USCLinux, a customized distribution of the RedHat Enterprise Advanced Server 3.0 (RHE3) Linux distribution. The Portable Batch System (PBS) is used to allocate nodes for a job.

B. Datasets

To evaluate the performance of our proposed method, we generated various junction trees by mimicking the junction trees converted from a real Bayesian network called the Quick Medical Reference decision theoretic version (QMR-DT) [14]. The number of cliques N in these junction trees was in the range 32768 to 65536. The clique degree d, i.e., the number of children of a clique, was in the range 1 to 8, except for the leaf cliques. The clique width W_C, i.e., the number of random variables per clique was varied from 8 to 20. The separator width W_S was from 2 to 5. The random variables r in the junction trees were either binary or ternary. When binary variables were used, the number of entries per clique potential table was between 256 and 1048576; while the number of entries per separator potential table was between 4 and 32. We used single precision floating point data for the potential tables. All the potential tables were aligned in the memory to improve the performance. We conducted experiments with 1 to 128 processors. For the data layout and the potential table organization, we followed the approach used in [5].

C. Baseline Methods

`Serial` is a sequential implementation of evidence propagation discussed in Section II-B. For a given junction tree, we start from the root to obtain the breadth-first search (BFS) order of the cliques. During evidence collection, we utilize Eq. (1) to update each clique according to the *reversed* BFS order. Then, during evidence distribution, the processor updates all the cliques again according to the BFS order.

`Clique allocate` is an asynchronous message driven parallel implementation of evidence propagation. We mapped the cliques at each level of the input junction tree onto distinct processors so as to maximize the number of cliques that can be updated in parallel. We started with leaf cliques during evidence collection. Once a clique obtained messages from all its children, the clique was updated by the host processor. Similarly, we started with the root and updated all the cliques again during evidence distribution.

`Data parallel` is a parallel baseline method exploring data parallelism in exact inference. We traversed the input junction tree according to breadth first search (BFS) order and updated the cliques one by one. For each clique, the potential table was partitioned and each processor updated a part of the potential table using parallel node level primitives proposed in [5]. The processors were synchronized after a clique was updated.

`Chains` is a parallel baseline method which decomposes a junction tree into a set of chains, each corresponding a path from a leaf clique to the root [8]. We manually assigned the chains to the processors, so that workload was evenly distributed. The processors first updated the assigned chains in parallel and then exchanged duplicated cliques in a global communication step. The junction tree was fully updated after the duplicated cliques were merged.

D. Experimental Results

In Figure 4, we illustrate the experimental results of the proposed distributed evidence propagation method and two baseline methods discussed in Section V-C with respect to a junction tree with the following parameters: $N = 65536, W_C = 15, r = 2, d = 4, W_S = 4$. We had consistent results with respect to other junction trees. We ran each experiment 10 times and calculated the average execution time and its standard deviation. As shown in Figure 4, the overhead due to parallelization was negligible for the proposed method, since the execution time with respect to a single processor was almost the same as the baseline called `serial`. In contrast, the baseline `chains` shows significant overhead. `Clique allocate` showed higher overhead than our proposed method, since it involves more coordination among the processors. Since the potential tables in the input junction tree are much smaller than those in [5], the `data parallel` baseline method showed limited scalability.

From Figure 4(a), the *improvement* of the proposed distributed evidence propagation method compared with all the baseline methods was more than 28% when 128 processors were used. In Figure 4(b), we illustrate the execution time when more than 32 processors were used, so that we can take a close look at the improvement of the proposed method.

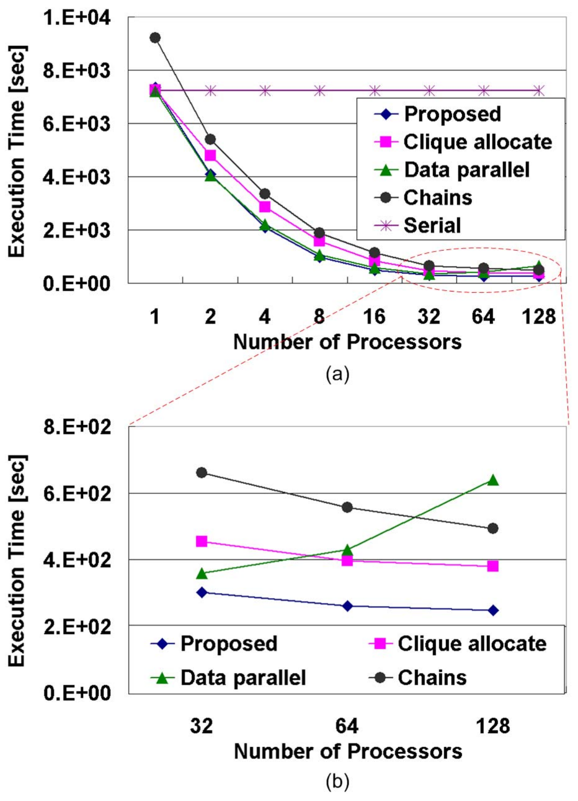
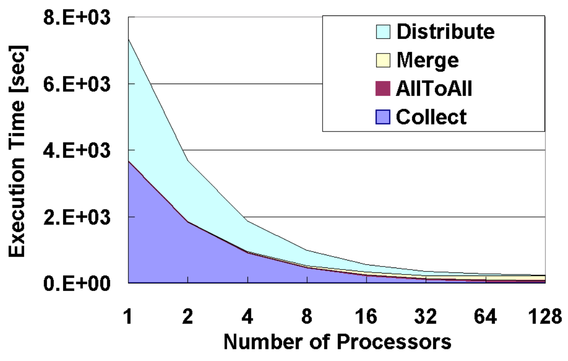
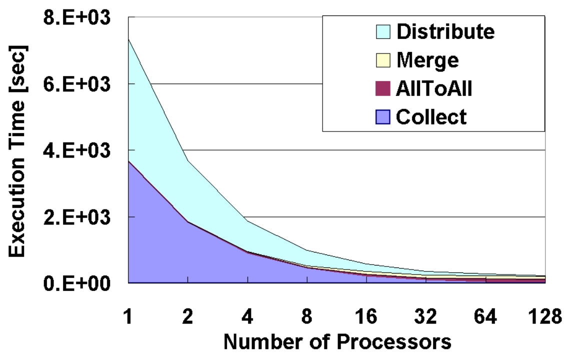


Figure 4. Scalability of the proposed technique compared with baseline methods.



(a) Evidence propagation with a single bitmap



(b) Evidence propagation with a set of sub-bitmaps

Figure 5. Execution time of various steps.

We illustrate the scalability for each stage, i.e., evidence collection (`collect`), all-to-all communication (`AllToAll`), junction tree merging (`Merge`) and evidence distribution (`Distribute`), of the proposed parallel evidence propagation algorithm (Lines 9-20 in Algorithm 2) in Figure 5. Figure 5(a) shows the results where we used a single bitmap. Thus, there was only one all-to-all communication. In Figure 5(b), we divided the bitmap into multiple bitmaps. Thus, we had multiple all-to-all communication steps, but the bandwidth utilization efficiency was improved. We controlled the number of bitmaps by altering the tradeoff factor η in Eq. 6

Figures 5(a) and (b) show that the impact of η was very limited when a small number of processors were used. As the number of processors increases, we observed the impact of η on the execution time for each stage. In Figure 6, we can observe that using a single all-to-all communication step results in less overhead due to startup latency. In Figure 6, we show the normalized execution time for the results in Figure 5(a) and (b) when 128 processors were used. When a single bitmap was used, the all-to-all communication took a small percentage of execution time, since we sent only one message. So, the overhead due to startup latency was minimized in this case. However, since invalid data were transferred in all-to-all communication, junction tree merging took additional time to identify useful separators from all received data, as shown in Figure 6. When multiple bitmaps were used, we improved the efficiency of junction tree merging, but the communication time was longer because of increased overhead due to startup latency.

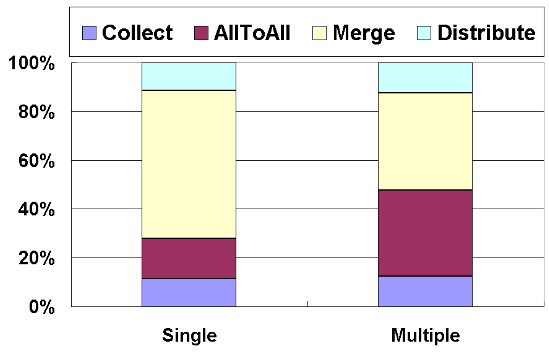


Figure 6. Normalized execution time of various steps.

Finally, we evaluated our proposed method using various junction trees. In Figure 7, the label *Original JTree* represents the junction tree having the following parameters:

$N = 65536, W_C = 15, r = 2, d = 4, W_S = 4$. We varied a parameter each time as shown in the labels to illustrate the impact of various parameters on the execution time of our method. When we reduced the number of cliques in the junction tree from 65536 to 32768, the execution time of the new junction tree exhibited similar scalability as the original tree. The execution time was almost half of that for the original junction tree. When we used the junction tree with $W_C = 10$, the execution time was much smaller than that for the original junction tree. For the original junction tree, we must update 2^{15} entries for each clique potential table. However, for the junction tree with $W_C = 10$, we only update 2^{10} entries, 1/32 of that for the original junction tree. Thus, it is reasonable that the execution time was much faster. When W_S was reduced from 4 to 2, the impact was negligible. When we increased d from 2 to 4, we observed in Figure 7 that the scalability was adversely affected. The execution time increased slightly as the number of processors increases. The reason is that large d results in more separators being transferred.

The experimental results in this section show that the proposed method scales well with respect to almost all the junction trees. In addition, compared with several baseline algorithms for evidence propagation, our method shows superior performance in terms of the total execution time.

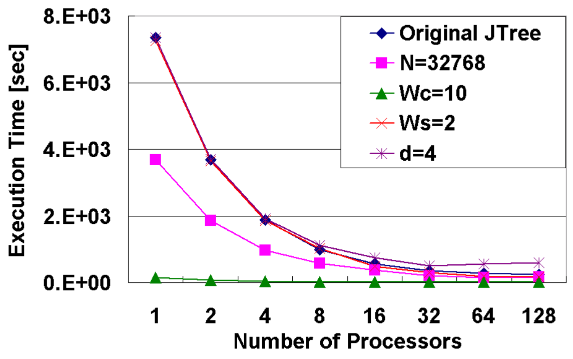


Figure 7. Impact of various parameters of junction trees on execution time.

VI. Conclusion

In this paper, we developed a novel parallel exact inference algorithm based on junction tree decomposition and merging. We used bitmaps to explore the tradeoff between startup latency and bandwidth utilization efficiency in communication. In the future, we plan to parallelize the evidence collection and distribution in each subtree at node level. In addition, we intend to develop a dynamic scheduling algorithm to improve the load balance across the processors.

Acknowledgment

This research was partially supported by the National Science Foundation under grant number CNS-0613376. NSF equipment grant CNS-0454407 is gratefully acknowledged.

References

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

[2] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2002.

[3] S. L. Lauritzen and D. J. Spiegelhalter, "Local computation with probabilities and graphical structures and their application to expert systems," *J. Royal Statistical Society B*, vol. 50, pp. 157–224, 1988.

[4] D. Pennock, "Logarithmic time parallel Bayesian inference," in *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 431–438.

[5] Y. Xia and V. K. Prasanna, "Scalable node-level computation kernels for parallel exact inference," *IEEE Trans. Comput.*, vol. 59, no. 1, pp. 103–115, 2010.

[6] A. V. Kozlov and J. P. Singh, "A parallel Lauritzen-Spiegelhalter algorithm for probabilistic inference," in *Supercomputing*, 1994, pp. 320–329.

[7] R. D. Shachter, S. K. Andersen, and P. Szolovits, "Global conditioning for probabilistic inference in belief networks," in *Proceedings of the Tenth Conference on Uncertainty in Articial Intelligence*, 1994, pp. 514–522.

[8] Y. Xia and V. K. Prasanna, "Junction tree decomposition for parallel exact inference," in *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2008.

[9] T. Ito, T. Uno, X. Zhou, and T. Nishizeki, "Partitioning a weighted tree to subtrees of almost uniform size," in *Proceedings of the 19th International Symposium on Algorithms and Computation*, 2008, pp. 196–207.

[10] "Message passing interface." [Online]. Available: http://www.mcs.anl.gov/mpi/

[11] W. Liu, C.-L. Wang, and V. K. Prasanna, "Portable and scalable algorithm for irregular all-to-all communication," *J. Parallel Distrib. Comput.*, vol. 62, no. 10, pp. 1493–1526, 2002.

[12] H. Zha, X. He, C. Ding, H. Simon, and M. Gu, "Bipartite graph partitioning and data clustering," in *Proceedings of the 10th international conference on Information and knowledge management*, 2001, pp. 25–32.

[13] USC center for High-Performance Computing and Communications, "http://www.usc.edu/hpcc/."

[14] B. Middleton, M. Shwe, D. Heckerman, H. Lehmann, and G. Cooper, "Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base," *Medicine*, vol. 30, pp. 241–255, 1991.