Efficient Implementation of Gaussian Belief Propagation Solver for Large Sparse Diagonally Dominant Linear Systems

Yousef El-Kurdi, Warren J. Gross, and Dennis Giannacopoulos

Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 2A7, Canada

We present an implementation-oriented algorithm for the recently developed Gaussian Belief Propagation solver that demonstrates $17\times$ speedup over the prior algorithm for diagonally dominant matrices generated by typical Finite Elements applications. Compared to the diagonally-preconditioned conjugate gradient method, our algorithm demonstrates empirical improvements up to $6\times$ in iteration count and speedups up to $1.8\times$ in execution time. Also we present a new flexible scheduling scheme of the algorithm that is aimed for implementation on parallel architectures by reducing the iteration count of parallel GaBP and achieving better hardware parallelism.

***Index Terms*—Acceleration, Gaussian belief propagation, parallel algorithms, sparse matrices.**

I. Introduction

BELIEF PROPAGATION (BP) algorithms, first presented by Pearl [1], are probabilistic inferencing algorithms based on recursive message updates that in general exhibit low complexity and high parallelism; both of which properties can be greatly exploited in processing large sparse linear systems. Linear systems of equations are algebraically formulated as $Ax = b$, where the solution to the system is $x^* = A^{-1}b$. Iterative methods, e.g., the Preconditioned Conjugate Gradient (PCG), are traditionally used to to solve such large sparse systems, since they exhibit lower computational complexity and memory requirement than directly computing the inverse of A. A newly introduced iterative method, shown in [2], uses BP over Gaussian graphical models (GaBP) as a solver for linear system of equations. In this paper, we present implementation-oriented algorithms of GaBP that demonstrate potential speedups for both sequential CPU execution as well as parallel implementation on emerging many-core architectures.

It can be seen that the solution of the linear system $x^* = A^{-1}b$ can be found by solving the optimization problem:

$$\max_x \left[\exp\left(-\frac{1}{2}x^T Ax + b^T x\right)\right]. \tag{1}$$

The exponential expression in (1) resembles a multivariate Gaussian probability distribution $p(\boldsymbol{x})$ where $\boldsymbol{x}$ is the nodal variables vector and A^{-1} is the covariance matrix. By algebraically solving the maximization problem, it can be shown that the solution vector x^* to the linear system is actually the mean vector of the nodal variables $\boldsymbol{x}$ of the probability distribution $p(\boldsymbol{x})$ defined as $\boldsymbol{\mu} \triangleq A^{-1}b$. Hence the solution to the linear system is transformed to the probabilistic inference problem of finding the means of the variables in the multivariate Gaussian distribution $p(\boldsymbol{x})$.

The matrix A_{ij} with nodal variables represented by $\boldsymbol{x}$ can be viewed as an undirected graph, also referred to as Markov random field, where each non-zero ($A_{ij} \neq 0$) represents an undirected edge between variable node n_i and variable node n_j. By factoring the graph's distribution $p(\boldsymbol{x})$ into the nodal functions $\phi_i(x_i) \triangleq \exp(-(1/2)A_{ii}x_i^2 + b_i x_i)$ and edge functions $\psi_{i,j}(x_i, x_j) \triangleq \exp(-(1/2)x_i A_{ij} x_j)$, the continuous formulation of belief propagation algorithm can then be applied to infer the means of the nodal variables $\overline{x_i}$. In belief propagation, each node n_i computes a new message towards node n_j on a particular edge $(i \rightarrow j)$ using all messages received from nodes in the neighbourhood $\mathcal{N}(i)$ of node n_i excluding the message received from n_j, with message updates from each node performed either sequentially or concurrently subject to a specific schedule. Since the underlying distribution is Gaussian, the belief updates will be based on propagating only two variables: the estimated nodal means μ and variances P, as shown in (2) and (3). For detailed derivation of GaBP, the reader is encouraged to refer to [2] and [3]:

$$P_{ij} = -A_{ij}^2 \left(A_{ii} + \sum_{k \in \mathcal{N}(i)\backslash j} P_{ki}\right)^{-1} \tag{2}$$

$$\mu_{ij} = -A_{ij}\left(A_{ii} + \sum_{k \in \mathcal{N}(i)\backslash j} P_{ki}\right)^{-1}\left(b_i + \sum_{k \in \mathcal{N}(i)\backslash j} \mu_{ki}\right). \tag{3}$$

GaBP was shown in [4] to converge for a particular class of matrices referred to as walk-summable models. The walk-summability condition states that the spectral radius of the normalized off-diagonals of A in the absolute sense should be $\rho(|I - D^{-(1/2)}AD^{-(1/2)}|) < 1$, where D is the diagonal elements of A. Such class of matrices includes the symmetric positive-definite diagonally dominant systems that arise in many key applications such as the finite element method (FEM).

The computational speed of GaBP implementation will depend on certain factors such as: the data-structure used to represent the nodes and their connectivities which are typically sparse in memory, and the message transfer medium such as the memory bandwidth for shared-memory architectures or the network bandwidth for CPU-clusters. GaBP nodes process messages in three designated stages. First, the node receives

Manuscript received July 06, 2011; revised October 01, 2011; accepted November 03, 2011. Date of current version January 25, 2012. Corresponding author: Y. El-Kurdi (e-mail: yousef.elkurdi@mail.mcgill.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMAG.2011.2176318

messages from sparse connections. Second, the node retrieves the messages locally and performs computations. Last, the node responds with new messages on the same sparse connections. For CPU implementation, the choice of sparse data-structure required to store the messages will have the critical impact on performance due to message access time, data-locality and vectorization of computational loops. As will be demonstrated later, sparse matrices arising from typical FEM applications can exhibit banded sparsity structure, when proper reordering algorithms are used, which reduces connection sparsity and increases the parallel GaBP performance.

GaBP message updates are performed subject to a particular schedule. Empirical results show that the choice of the belief propagation schedule can greatly impact the number of iterations required for convergence [5]. Two basic scheduling schemes common in belief propagation are, Sequential Update (SU) scheduling, and Parallel Update (PU) scheduling also known as flooding. In SU scheduling, nodes are processed in sequence according to a particular ordering; while messages are propagated sequentially to subsequent nodes in the same iteration. Hence, nodes can only be processed in sequence providing little opportunity for parallel node processing. PU scheduling on the other hand, facilitates fully concurrent execution of nodes; however the catch is that PU scheduling requires considerably larger number of iterations for the belief propagation to converge when compared to SU scheduling.

In this paper, we introduce an implementation-oriented algorithm of GaBP suitable for execution on both sequential as well as parallel hardware architectures. The new algorithm achieves optimum message retrieval time by embedding nodal connection information in each message, therefore eliminating the need to use sparse data-structures, and facilitating the use of constant-time access data-structures such as queues or stacks. In addition, such data-structures exhibit contiguous data locality allowing the algorithm to exploit instruction level parallelism (ILP) or (loop vectorization) making the algorithm's execution-time more competitive with traditional iterative methods implementations such as the PCG algorithm. We also present empirical results of SU-GaBP computational performance compared with PCG showing considerable reduction in iteration count for large sparse diagonally-dominant linear systems. Finally, we present a Hybrid Update (HU) scheduling version of our algorithm which is expected to benefit from features of both sequential and parallel updates resulting in improved partially-parallel implementations while considerably reducing the PU-GaBP iterations.

1: *Initialize:*
2: Define node ordering: η
3: **for** each edge $A(i,j)$ **do**
4: **if** $n_i <_\eta n_j$ **then**
5: $n_i.\,\mathrm{push}(n_j.\,\mathrm{pointer}, A_{ij}, P_{ji}=0, \mu_{ji}=0)$
6: **end if**
7: **end for**
8: *Compute:*
9: **repeat** {iterations}
10: **for** each node $n_i \in \eta$ **do**
11: $P_{agg} = A_{ii} + \sum_k P_{ki}$
12: $\mu_{agg} = b_i + \sum_k \mu_{ki}$
13: **for** each received message $n_j \to n_i$ **do**
14: $\mu_{ij} = -A_{ij}(P_{agg} - P_{ji})^{-1}(\mu_{agg} - \mu_{ji})$
15: $P_{ij} = -A_{ij}^2(P_{agg} - P_{ji})^{-1}$
16: n_i.pop_message
17: $n_j.\,\mathrm{push}(n_i.\,\mathrm{pointer}, A_{ij}, P_{ij}, \mu_{ij})$
18: **end for**
19: $\overline{x_i} = \mu_{agg} P_{agg}^{-1}$
20: **end for**
21: **until** *convergence:* all $\overline{x_i}$ converged
22: *Output:* $\overline{x_i}$

Fig. 1. Sequential update GaBP implementation-oriented algorithm.

1: *Initialize:*
2: **for** each edge $A(i,j)$ **do**
3: $n_i.\,\mathrm{push}(n_j.\,\mathrm{pointer}, A_{ij}, P_{ji}=0, \mu_{ji}=0)$
4: **end for**
5: *Compute:*
6: **repeat** {iterations}
7: **for** each node n_i **do** {concurrent execution}
8: wait for all n_i messages to be received
9: $n_i.\,\mathrm{swap\_pointers}(\mathrm{currMessStack}, \mathrm{newMessStack})$
10: $P_{agg} = A_{ii} + \sum_k P_{ki}$
11: $\mu_{agg} = b_i + \sum_k \mu_{ki}$
12: **for** each received message $n_j \to n_i$ **do**
13: $\mu_{ij} = -A_{ij}(P_{agg} - P_{ji})^{-1}(\mu_{agg} - \mu_{ji})$
14: $P_{ij} = -A_{ij}^2(P_{agg} - P_{ji})^{-1}$
15: $n_i.\,\mathrm{pop\_message}$
16: $n_j.\,\mathrm{push}(n_i.\,\mathrm{pointer}, A_{ij}, P_{ij}, \mu_{ij})$
17: **end for**
18: $\overline{x_i} = \mu_{agg} P_{agg}^{-1}$
19: **end for**
20: **until** *convergence:* all $\overline{x_i}$ converged
21: *Output:* $\overline{x_i}$

Fig. 2. Parallel update GaBP implementation-oriented algorithm.

II. Implementation-Oriented GaBP Algorithm

The sequential update version of our algorithm is shown in Fig. 1. By adding a source node pointer (n_i) and the edge variance parameter $(A_{i,j})$, simple contiguous data-structures such as stacks and queues are only required to store the nodes' messages. This optimization resulted in constant-time per message for retrieval and processing, reducing the algorithm's overall message processing complexity to $O(\mathrm{nnz})$, where nnz is the number of non-zeros in A. Also since the data-structure used is contiguous in memory, the computational loops in lines 11 and 12 can be vectorized by the compiler resulting in considerable speedups due to ILP. Stacks are used in all our algorithms in Figs. 1, 2, and 3; however, queues could be used instead without any impact on the algorithm's performance. It is important to note that this message modification resulted in a fixed increase in memory requirement on a per message basis. Since this fixed increase is independent of the problem size, the algorithm's overall message memory requirement scalability is not impacted e.g., $O(c \times \mathrm{nnz}) = O(\mathrm{nnz})$.

The initialization stage is critical for the correct operation of the algorithm. As shown in lines 2 to 6, a unique nodal ordering

Initialize:
Define node partitioning ζ
Define node ordering η in each ζ
for each edge $A(i,j)$ **do**
  if $n_i =_\zeta n_j$ **then** {nodes in the same partition}
    if $n_i <_\eta n_j$ **then**
      $n_i.\text{push}(n_j.\text{pointer}, A_{ij}, P_{ji}=0, \mu_{ji}=0)$
    end if
  else {nodes NOT in the same partition}
    $n_i.\text{push}(n_j.\text{pointer}, A_{ij}, P_{ji}=0, \mu_{ji}=0)$
  end if
end for
Compute:
repeat {iterations}
  for each partition in ζ **do** {concurrent execution}
    for each node $n_i \in \eta$ **do** {sequential execution}
      wait for all n_i messages to be received
      $n_i.\text{swap\_pointers}(\text{currMessStack}, \text{newMessStack})$
      $P_{agg} = A_{ii} + \sum_k P_{ki}$
      $\mu_{agg} = b_i + \sum_k \mu_{ki}$
      for each received message $n_j \rightarrow n_i$ **do**
        $\mu_{ij} = -A_{ij}(P_{agg} - P_{ji})^{-1}(\mu_{agg} - \mu_{ji})$
        $P_{ij} = -A_{ij}^2(P_{agg} - P_{ji})^{-1}$
        $n_i.\text{pop\_message}$
        $n_j.\text{push}(n_i.\text{pointer}, A_{ij}, P_{ij}, \mu_{ij})$
      end for
      $\overline{x_i} = \mu_{agg} P_{agg}^{-1}$
    end for
  end for
until *convergence:* all $\overline{x_i}$ converged
Output: $\overline{x_i}$

Fig. 3. Hybrid update GaBP implementation-oriented algorithm.

(η) needs to be defined, where $<_\eta$ is an ordering relationship used to initialize messages according to η order. The nodes need to be processed according to the same order, so to avoid deadlock in SU-GaBP.

The parallel update version of our algorithm is shown in Fig. 2. The PU-GaBP algorithm facilitates concurrent node execution by using two message buffers, a current-message buffer to store messages from the previous iteration, and a new-message buffer to receive messages in the current iteration. At the beginning of each node execution the buffers are swapped. Also, there is no need to define any node ordering as the case in SU-GaBP. Since the number of processing elements in typical computing environments is much less than the number of nodes, the PU-GaBP algorithm may not be practical for implementation of large systems; nonetheless, the algorithm has conceptual importance as we will see later.

A. Hybrid Update GaBP Algorithm Implementation

In typical parallel computing environments, the number of processing elements is limited; hence a degree of sequential processing will be required for large systems. Our implementation-oriented Hybrid Update (HU) algorithm, shown in Fig. 3, take advantage of this sequentiality to propagate faster message updates which reduces the PU-GaBP iteration count while exploiting parallelism in both nodal multi-processing and sequential ILP.

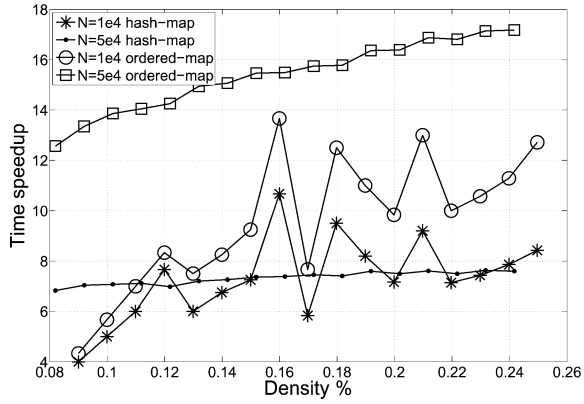


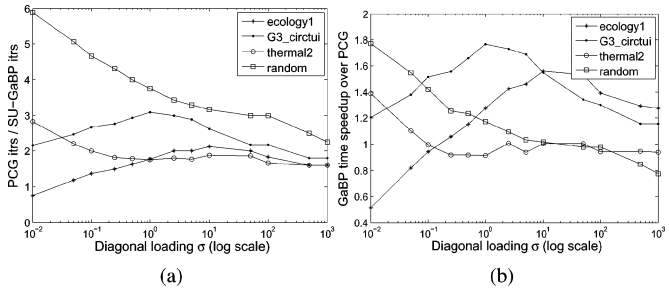
Fig. 4. Implementation-oriented SU-GaBP algorithm speedup over ordered-map and hash-map based implementations.

By creating a partitioning scheme of nodes, where each partition is processed in parallel by a processing element, updates between nodes in different partitions will be done using the PU algorithm, while updates between nodes in the same partition will be done using the SU algorithm. This flexibility allow the HU algorithm to easily implement different sequential-parallel scheduling variations by simply changing the partitioning and the node ordering within each partition. That is, by choosing a partitioning and an ordering scheme that exploits node locality in terms of connectivity, the number of iteration penalty due to parallel GaBP can be considerably reduced, as will be demonstrated by our results. Typically, FEM matrices have banded sparsity structures making them more suitable for parallel processing using our HU algorithm.

III. Results

Fig. 4 shows the speedups that can be achieved using our proposed implementation-oriented SU-GaBP algorithm compared with implementations using both ordered-maps and hash-maps as data-structures to store messages. All executions were running on a single CPU core (Intel Core2 Quad @ 2.8 GHz) using double-precision computations. Randomly generated sparse matrices (nodes with random connectivities) are used for this performance analysis. Our implementation has demonstrated speedups of up to $17\times$ with an increasing overall trend as the number of non-zeros increases. This speedup trend was mainly due to the exploitation of ILP parallelism facilitated by the enhanced data-locality of our new algorithm. While hash-maps can provide constant-time access for messages, they exhibit poor data locality limiting ILP parallelism. The vertical trend in the graphs is mainly due to the speedups from the improved message access time-complexity. While our algorithm and hash-map based implementation provides constant-time access for messages, the ordered-map based implementation provides $O(m)$ access time-complexity where m is the total messages per node. It is important to note that ordered-map is typically the preferred choice of data-structure to implement compressed row sparse matrices in sparse computational libraries that require flexible read/write operations on the sparse matrix such as the case in GaBP.

In order to assess the computational speed of SU-GaBP, we compare it with the Diagonally-Preconditioned Conjugate Gradient (D-PCG). The D-PCG code was executed on the same



(a) (b)

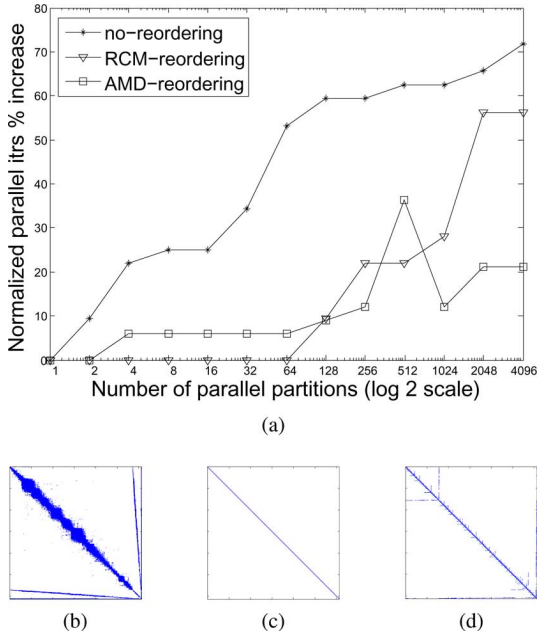
Fig. 5. Implementation-oriented SU-GaBP speedup. (a) Iteration improvement of GaBP over PCG, (b) SU-GaBP time speedup over PCG.

TABLE I
TEST MATRICES [7]

Category	ecology1	G3_circuit	thermal2	random
N (nodes)	1,000,000	1,585,478	1,228,045	1,000,000
nnz	4,996,000	7,660,826	8,580,313	8,999,976

CPU and it was obtained from the GMM++ library [6] which is widely used for FEM applications. Iterations were stopped when the Frobenius norm of the residual reached $\epsilon = 10^{-9}$ using double-precision computation. The test matrices, shown in Table I, are obtained from [7], with the exception of the matrix "Random". The matrix "thermal2" results from unstructured steady-state FEM application. The matrices were made diagonally dominant by loading the diagonals with a uniformly distributed positive random number having a standard deviation $\sigma \in [10^{-2}, 10^{3}]$, in order to make them conform with the walk-summability criteria [4] required for GaBP convergence. The plots in Fig. 5(a) and (b) show the performance results against D-PCG. Iteration count reductions, up to 6x, are obtained by SU-GaBP. Also our SU-GaBP implementation was able to achieve time speedups for many cases reaching up to $1.8\times$. It is worth noting the discrepancy between the time speedup gains and the iteration reduction, which could be mainly due to the memory bandwidth requirements of SU-GaBP which is not currently optimized in our code.

The parallel behavior of our HU-GaBP algorithm was simulated on a single-core CPU. In order to simulate different partitioning and node ordering, matrix reordering techniques were used. Two common reordering techniques used here are Reverse Cuthill-McKee (RCM) [8] which reduces the matrix bandwidth, and Approximate Minimum-Degree (AMD) which produces large blocks of zeros [9]. Fig. 6 shows the parallel simulation results of our HU-GaBP implementation algorithm on the matrix "thermal2" as the number of parallel partitions increases from 1 to 4096. HU-GaBP demonstrates gradual rate of increase in iterations, while RCM and AMD showed considerably lower rate of increase compared to the original matrix. These results demonstrate HU-GaBP potential for parallelization speedups and its flexibility in exploiting the problem's connectivity structure. Our future work objective is to implement HU-GaBP on CPU-clusters and many-core architectures and compare its performance to leading parallel implementations of other iterative methods.



(a)

(b) (c) (d)

Fig. 6. HU-GaBP algorithm results for thermal2. (a) HU-GaBP parallel iteration increase rate, (b) Sparsity structure of original thermal2, (c) RCM reordered, and (d) AMD reordered.

IV. CONCLUSION

Implementation-oriented algorithms of GaBP were presented which demonstrates speedups of up to $17\times$. Also, both improvements in execution time and reduction in iteration count over D-PCG were demonstrated using our modified algorithm. Finally, a hybrid parallel-sequential scheduling variant of our algorithm was simulated to demonstrate considerable reductions in parallel iterations and promising more potential for parallel performance gains from implementations on CPU-cluster and multi-core hardware architectures.

REFERENCES

[1] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. New York: Morgan Kaufmann, 1988.
[2] O. Shental, P. Siegel, J. Wolf, D. Bickson, and D. Dolev, "Gaussian belief propagation solver for systems of linear equations," in *IEEE Int. Symp. Information Theory (ISIT)*, 6–11, 2008, pp. 1863–1867.
[3] Y. Weiss and W. T. Freeman, "Correctness of belief propagation in gaussian graphical models of arbitrary topology," *Neural Comput.*, vol. 13, no. 10, pp. 2173–2200, 2001.
[4] J. K. Johnson, D. M. Malioutov, and A. S. Willsky, "Walk-sum interpretation and analysis of Gaussian belief propagation," in *Advances in Neural Information Processing Systems 18*. Cambridge, MA: MIT Press, 2006, pp. 579–586.
[5] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," in *Proc. 22nd Conf. Uncertainty in AI (UAI)*, Boston, MA, Jul. 2006.
[6] Gmm++: a generic template matrix c++library. Retrieved 2010. [Online]. Available: http://download.gna.org/getfem/html/homepage/gmm.html
[7] The University of Florida Sparse Matrix Collection. Submitted to ACM Transactions on Mathematical Software. [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices
[8] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proc. 1969 24th National Conf. ACM*, New York, 1969, pp. 157–172, ACM.
[9] A. George and W. H. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Rev.*, vol. 31, pp. 1–19, Mar. 1989.