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
3480 University Street, Montreal (QC), H3A 2A7, Canada
yousef.el-kurdi@mail.mcgill.ca, warren.gross@mcgill.ca, dennis.giannacopoulos@mcgill.ca

***Abstract*—We present a fast and parallelizable variant of the recently developed Gaussian Belief Propagation solver that demonstrates 17x speedups over basic implementations. Compared to the diagonally-preconditioned conjugate gradient, our algorithm demonstrates empirical improvements up to 6x in iteration count and speedups of up to 1.8x in execution time. Also we present variant of the algorithm that is aimed for enhanced implementation on parallel architectures.**

I. Introduction

Belief propagation (BP) algorithms, first invented by Pearl [1], are probabilistic inferencing algorithms based on recursive message updates that in general exhibit low complexity and are inherently parallel; both of which properties can be greatly exploited in processing large sparse linear systems. It is shown in [2] that BP over Gaussian graphical models (GaBP) can be used as a solver for linear systems of equations $Ax = b$.

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

$$\max_x \exp(-\frac{1}{2}x^T Ax + b^T x) \tag{1}$$

The exponential expression in (1) resembles a Gaussian multivariate probability distribution $p(\boldsymbol{x})$ where $\boldsymbol{x}$ is the nodal variables vector and a covariance matrix A^{-1}. Through some algebraic manipulation, it can be shown that the solution to the linear system x^* is actually the means 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 transferred to a probabilistic inference problem of finding the means of the variables in the multivariate distribution $p(\boldsymbol{x})$.

The matrix A_{ij} with nodal variables represented by $\boldsymbol{x}$ can be viewed as an undirected graphical model, also referred to as Markov random field, where non-zeros ($A_{ij} \neq 0$) represents undirected edges between variable nodes i and j. By factoring the graph's distribution $p(\boldsymbol{x})$ into self potential functions $\phi_i(x_i) \triangleq \exp(-\frac{1}{2}A_{ii}x_i^2 + b_i x_i)$ and edge potentials $\psi_{i,j}(x_i, x_j) \triangleq \exp(-\frac{1}{2}x_i A_{ij} x_j)$, 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 belief 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 scheduling. Since the underlying distribution is Gaussian, the belief updates, shown in (2) and (3), will be based on propagating only two variables: the estimated nodal means μ and variances P. For detailed derivation of GaBP, the reader is encouraged to refer to [2] and [3].

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

$$\mu_{i\rightarrow j} = -A_{ij}(A_{ii} + \sum_{k\in\mathcal{N}(i)\setminus j} P_{k\rightarrow i})^{-1}(b_i + \sum_{k\in\mathcal{N}(i)\setminus j} \mu_{k\rightarrow i}) \tag{3}$$

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

In this paper, we will present an efficient implementation of GaBP solver ideal for processing large sparse systems and that is suitable for implementation on both sequential as well as parallel environments. In addition, we will demonstrate the parallelism capabilities by simulating a scheduling-efficient parallel variant of our implementation that efficiently limits the iteration increase due to fully parallel scheduling. Also, since previous work did not present detailed comparison analysis of GaBP with Conjugate Gradient method, we will present empirical results of GaBP convergence performance compared to the diagonally-Preconditioned Conjugate Gradients (PCG) method, and we will show that our implementation efficient GaBP can demonstrate both a reduction in iteration count as well as a speedup in execution time for large sparse diagonally dominant systems.

II. Efficient Implementation GaBP Algorithm

In practical implementations of GaBP, the actual performance will depend on certain factors mainly, the used sparse data-structure, and the message transfer medium, e.g. memory bandwidth. For general purpose CPU implementation, the choice of sparse data-structure will have the critical impact on performance due to message access time, data-locality and vectorization of loops. For example, hash-maps (unordered-maps) can be used to provide constant-time access of data, however they exhibit poor data locality.

The paper's main contribution is the introduction of a fast and efficient implementation algorithm of GaBP suitable for execution on both sequential as well as parallel architectures. Since our algorithm employs constant-time access data-structures such as queues or stacks, it achieves optimum message processing performance of $O(nnz)$ per iteration which is comparable to PCG. In addition, the new algorithm exhibits more data locality and can readily exploit instruction level parallelism (ILP) which makes its performance competitive with well known iterative methods such as the Jacobi, the Gauss-Seidel, the Successive Overrelaxation, and the CG method.

In addition to its demonstrated efficiency for sequential implementations, the GaBP algorithm is inherently an embarrassingly parallel algorithm. Typically it is expected that the number of iterations for fully parallel scheduled GaBP to be increased, however we are presenting here a new hybrid sequential-parallel scheduling algorithm that should optimally limit this increase in iterations. Hence our hybrid implementation will efficiently exploit parallelism in both ILP, and course-grained multi-processing.

III. Results

The graphs in Fig. 1 (a) show the time speedups that can be achieved using our proposed GaBP implementation algorithm compared with the basic algorithm using both ordered-maps and hash-maps as data-structures to store messages. All executions were running sequential scheduling GaBP on a single CPU core (Intel Core2 Quad @ 2.8GHz) running Linux. In all test cases of random sparse matrices, our implementation has demonstrated speedups of up to 17x with an increasing overall trend as the number of non-zeros increases. This speedup was mainly due to the exploitation of ILP parallelism facilitated by the enhanced data-locality of the new algorithm.

The PCG method was used for the performance comparison with GaBP. The PCG was executed on the same CPU and was obtained from the GMM++ library [5], which is widely used for FEM applications. Iterations were stopped until the residue reached $\epsilon = 10^{-9}$ using double-precision computation. The test matrices are obtained from [6], with the exception of one matrix which was randomly generated by Matlab. The matrices were made diagonally dominant by loading the diagonals with a uniformly distributed positive random number having a standard deviation $\sigma \in [10^{-2}, 10^{2}]$. The plots in Fig. 1 (b and c) show the performance results against PCG. Iteration count reductions, up to 6x, are obtained by GaBP. Also our GaBP implementation was able to achieve time speedups for most cases reaching up to 1.8x.

The parallel behaviour of our hybrid scheduling GaBP algorithm was simulated on the same CPU. Fig. 1 (d) shows the number of iterations (scaled by the maximum number of iterations) increase due to parallelism as the number of CPU partitions increases from 1 to 4096. For this experiment a simple partitioning scheme was used by assigning nodes sequentially based on the order of their matrix index. It can be seen that we obtain slowest and limited increase in iterations

TABLE I
Test Matrices

Category	ecology1	G3_circuit	thermal2	random
N	1,000,000	1,585,478	1,228,045	1,000,000
Non-zeros	4,996,000	7,660,826	8,580,313	8,999,976

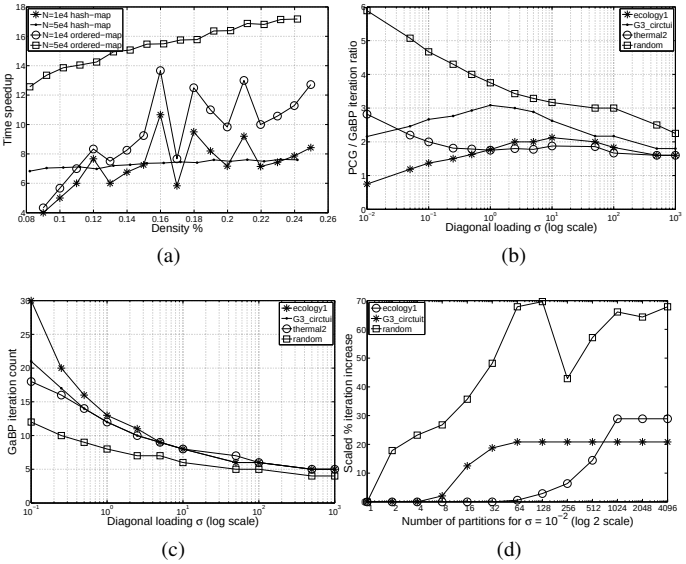

Fig. 1. GaBP implementation speedup (a) New FIFO-based implementation speedup over map or hash-map basic implementations, (b) Iteration speedup over PCG, (c) GaBP iterations, (d) Hybrid GaBP iteration increase

for matrices with banded sparsity structure when compared to "random" since our hybrid-scheduling algorithm can better exploit the local node connectivities for such matrices.

In conclusion, a new efficient GaBP implementation algorithm is presented demonstrating speedups of up to 17x over basic implementations. Also, both improvements in time speedup and reduction in iteration count over PCG was demonstrated using our modified algorithm. Finally, a parallel-sequential hybrid scheduling GaBP variant was simulated to demonstrate ability to achieve promising parallel performance. The long version of the paper will present a detailed analysis of the new algorithm for a wider class of matrices. In addition, we will demonstrate the implementation and the performance of the hybrid-scheduled GaBP on parallel architectures such as GPUs.

References

[1] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
[2] O. Shental and D. Bickson et. al., in *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, 6-11 2008, pp. 1863–1867.
[3] Y. Weiss et. al., *Neural Comp.*, vol. 13, no. 10, pp. 2173–2200, 2001.
[4] J. K. Johnson et. al., in *in Advances in Neural Information Processing Systems 18*. MIT Press, 2006, pp. 579–586.
[5] Gmm++: a generic template matrix c++library. [Online]. Available: http://download.gna.org/getfem/html/homepage/gmm.html
[6] The University of Florida Sparse Matrix Collection. [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices