Computer Physics Communications 193 (2015) 38–48

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Parallel finite element technique using Gaussian belief propagation

Yousef El-Kurdi [a,*,1], Maryam Mehri Dehnavi [b,1], Warren J. Gross [a], Dennis Giannacopoulos [a]

[a] *McGill University, Department of Electrical and Computer Engineering, 3480 University Street, Montreal, Quebec, H3A 0E9, Canada*
[b] *Massachusetts Institute of Technology, G770, 32 Vasser Street, Cambridge, MA 02139, USA*

ARTICLE INFO

Article history:
Received 29 January 2015
Received in revised form
2 February 2015
Accepted 30 March 2015
Available online 6 April 2015

Keywords:
FEM
Gaussian belief propagation
Graphical models
Gaussian distributions
Parallel algorithms
GPU

ABSTRACT

The computational efficiency of Finite Element Methods (FEMs) on parallel architectures is severely limited by conventional sparse iterative solvers. Conventional solvers are based on a sequence of global algebraic operations that limits their parallel efficiency. Traditionally, sophisticated programming techniques tailored to specific CPU architectures are used to improve the poor performance of sparse algebraic kernels. The introduced FEM Multigrid Gaussian Belief Propagation (FMGaBP) algorithm is a novel technique that eliminates all global algebraic operations and sparse data-structures. The algorithm is based on reformulating the FEM into a distributed variational inference problem on graphical models. We present new formulations for FMGaBP, which enhance its computation and communication complexities. A Helmholtz problem is used to validate the FMGaBP formulation for 2D, 3D and higher FEM degrees. Implementation techniques for multicore architectures that exploit the parallel features of FMGaBP are presented showing speedups compared to open-source libraries, specifically deal.II and Trilinos. FMGaBP is also implemented on manycore architectures in this work; Speedups of 4.8X, 2.3X and 1.5X are achieved on an NVIDIA Tesla C2075 compared to the parallel CPU implementation of FMGaBP on dual-core, quad-core and 12-core CPUs respectively.

1. Introduction

Manycore architectural advances in High Performance Computing (HPC) have introduced difficult challenges to the FEM software design. The conventional FEM software relies on performing global and sparse algebraic operations that severely limits its parallel performance. Many attempts were made to improve the performance of conventional sparse computations at the expense of sophisticated programming techniques. Such techniques are tailored to specific CPU hardware architectures, such as cache access optimizations, data-structures and code transformations [1]. These optimizations are known to limit code portability and reusability. For example, implementations of Conjugate Gradient (CG) solvers for FEM problems [2], require global sparse operations which perform at a low fraction of the peak CPU computational throughput [3]. Also accelerating CG solvers on parallel architectures is communication-bound; recent attempts to improve the communication overhead of such solvers through reformulation, namely communication avoiding schemes, suffer from numerical instability and limited support for preconditioners [4,5]. This performance bottleneck is even more pronounced when high accuracy FEM analysis scales to large number of unknowns, in the order of millions or more, which prevents the FEM software users from productively utilizing their parallel HPC platforms.

While existing generic and optimized libraries such as deal.II [6], GetFEM++ [7], and Trilinos [8] can be used for sparse FEM computations; obtaining a sustained performance can be difficult due to the varying sparsity structure for different application areas. In addition, such libraries do not help with the costly stage of assembling the sparse matrix. However, recent work by Kronbichler et al. [9] use a matrix free (MF) approach to execute the sparse matrix–vector multiply (SMVM) kernel in the CG solver. While their approach shows promising speedups, it does not depart from the sequential global algebraic setup of the CG solver and is only efficient for high order elements. The present work is based on a novel distributed FEM reformulation using belief propagation (BP) that eliminates the dependency on any sparse data-structures or algebraic operations; hence, attacking the root-cause of the problem.

* Corresponding author.
E-mail addresses: yousef.el-kurdi@mail.mcgill.ca (Y. El-Kurdi), mmehri@mit.edu (M.M. Dehnavi), warren.gross@mcgill.ca (W.J. Gross), dennis.giannacopoulos@mcgill.ca (D. Giannacopoulos).
1 These authors contributed equally to this work.

http://dx.doi.org/10.1016/j.cpc.2015.03.019
0010-4655/© 2015 Elsevier B.V. All rights reserved.

The belief propagation algorithm, as proposed by Pearl in [10], is a recursive message passing algorithm on graphical models that efficiently computes the marginal distribution of each variable node by sharing intermediate results. If the graph is a tree, then BP is guaranteed to converge to exact marginals. However, if the graph contains cycles, as typically the case in many practical applications, then BP takes an iterative form, referred to as Loopy Belief Propagation (LBP), which can be used to obtain an approximation for the marginals [10–14]. BP recently showed excellent empirical results in certain applications, such as machine learning, channel decoding, and computer vision [15–25]. Shental et al. [26] introduced the Gaussian BP algorithm as a parallel solver for a linear system of equations by modeling it as a pairwise graphical model. While the solver showed great promise for highly parallel computations on diagonally dominant matrices [27], it does not scale for large FEM matrices. It also fails to converge for high order FEM problems [28,29]. In addition, such a solver would still require assembling a large sparse data-structure.

The introduced Finite Element Gaussian Belief Propagation (FGaBP) algorithm and its multigrid variant, the FMGaBP algorithm, presented in [28,30,31], are distributed reformulations of the FEM that results in highly efficient parallel implementations. The algorithms provide a highly parallel approach to processing the FEM problem, element-by-element, based on distributed message communication and localized computations. This provides an algorithm amicable to different parallel computing architectures such as multicore CPUs and manycore GPUs.

In this work, we introduce new formulations for the FGaBP algorithm that better exploit its distributed nature. The new algorithms provide more efficient memory bandwidth utilization and considerably lower computational complexity; that is, reducing the local computational complexity from $O\left(n^3\right)$ to $O\left(n^2\right)$, where n is the rank of the local (element) FEM matrix. We verify the numerical results of the new formulation using the definite Helmholtz equation with a known solution. We also compare the new formulations with state-of-the-art open-source libraries such as deal.II [6] and Trilinos [8] on modern multicore CPUs. Implementation details of FMGaBP on GPUs are also presented in this work and its performance is compared to multicore CPUs.

The paper is organized as follows. In Section 2, a background on the FGaBP and FMGaBP algorithms is provided, which illustrates the algorithms and their key parallel features. In Section 3, we present the new formulations that reduce computation and communication costs of FGaBP. Section 4 presents implementation details on multicore and manycore architectures. Finally in Section 5, we present and discuss speedup results and close with concluding remarks.

2. Preliminary

In this section, an overview of the FGaBP and FMGaBP algorithms is provided; illustrating their distributed attributes, which will later be used to develop more efficient variants of the algorithms.

2.1. The FGaBP algorithm

In the following, the FGaBP algorithm is presented in three main stages. First, the FEM problem is transformed into a probabilistic inference problem. Second, a factor graph model of the FEM problem is created to facilitate the execution of a computational inference algorithm such as BP. Finally, the FGaBP update rules and algorithm is presented.

2.1.1. FEM as a variational inference

The variational form of the Helmholtz equation as discretized by the FEM is generally represented as follows [32,33]:

$$\mathcal{F}(U) = \sum_{s\in\mathcal{S}} \mathcal{F}_s(U_s) \tag{1}$$

where $\mathcal{S}$ is the set of all finite elements (local functions); U_s are the field unknowns for element s; and $\mathcal{F}_s$ is the energy-like contribution of each finite element. The local function $\mathcal{F}_s$ takes a quadratic form that can be shown as:

$$\mathcal{F}_s(U_s) = \frac{1}{2}U_s^T M_s U_s - B_s^T U \tag{2}$$

in which M_s is the element characteristic matrix with dimensions $n \times n$ where n is the number of Local element Degrees of Freedom (LDOF), and B_s is the element source vector.

Conventionally, the FEM solution is obtained by setting $\frac{\partial \mathcal{F}}{\partial U} = 0$, which results in a large and sparse linear system of equations presented as:

$$Au = b \tag{3}$$

where A is a large sparse matrix with dimensions $N \times N$; N is the number of Global Degrees of Freedom (GDOF) of the linear system; and b is the Right-Hand Side (RHS) vector. The linear system is typically solved using iterative solvers such as the Preconditioned Conjugate Gradient (PCG) method when A is Symmetric Positive Definite (SPD).

The FGaBP algorithm takes a different approach by directly minimizing the energy functional (1) using the BP inference algorithm. A variational inference formulation of FEM is created by modifying (1) as follows:

$$\mathcal{P}(U) = \exp\left[-\mathcal{F}\right] \tag{4}$$

$$= \frac{1}{Z}\prod_{s\in\mathcal{S}} \Psi_s(U_s) \tag{5}$$

where Z is a normalizing constant, and $\Psi_s(U_s)$ are local factor functions expressed as:

$$\Psi_s(U_s) = \exp\left[-\frac{1}{2}U_s^T M_s U_s + B_s^T U_s\right]. \tag{6}$$

Considering applications where M_s is SPD, the function Ψ_s, as in (6), takes the form of an unnormalized multivariate Gaussian distribution. In addition, it can be shown using convex analysis [16,34] that $\mathcal{P}$ is a valid multivariate Gaussian distribution functional of the joint Gaussian random variables U. The solution point to the original problem, which is the stationary point of the functional $\mathcal{F}$, can be restated as:

$$\arg\min_U \mathcal{F} = \arg\max_U \mathcal{P}. \tag{7}$$

Since the Gaussian probability $\mathcal{P}$ is maximized when $U = \mu$, where μ is the marginal mean vector of $\mathcal{P}$, the FEM problem can alternatively be solved by employing computational inference for finding the marginal means of U under the distribution $\mathcal{P}$. Hence BP inference algorithms will be employed to efficiently compute the marginal means of the random variables U.

2.1.2. FEM factor graph

Because $\mathcal{P}$ is directly derived from the FEM variational form, it is conveniently represented in a factored form as shown in (5). As a result, we can define a graphical model to facilitate the derivation of the BP update rules. One widely used graphical model is a Factor Graph (FG) [15], which is a bipartite graphical model that directly represents the factorization of $\mathcal{P}$. In our setting, we refer to such a FG as the FEM-FG. The FEM-FG, as shown in Fig. 1(b), includes two types of nodes, a random variable node (u_i) representing each node in the unknown vector U, and a factor node representing the local factors Ψ_s. An edge is inserted between a variable node u_i and a factor node Ψ_s, if u_i is an argument of the local factor Ψ_s.

Inference on FEM-FG can perform more dense and localized computations, especially for higher order FEM, as opposed to inference on the pairwise models based on sparse matrices as in [26].

2.1.3. BP update rules

Using the FEM-FG, we can execute the general BP update rules [10] by passing two types of messages, Factor Node (FN) messages and Variable Node (VN) messages. A factor node message, $m_{a\to i}$, is sent from factor nodes a (FN$_a$) to the connected variable node i (VN$_i$); and a variable node message, $\eta_{i\to a}$, is sent back from VN$_i$ to FN$_a$. BP messages are basically probability distributions such that, a FN message $m_{a\to i}$ represents the distribution in terms of the continuous random variable u_i, or the most probable state of u_i, as observed from the FN Ψ_a. In return, the VN message $\eta_{i\to a}$ is a distribution in terms of u_i representing observations from other connected factors. For simplicity, we will drop the arrow from the notation and represent messages between nodes' indexes a and i as m_{ai} and η_{ia}. The general BP update rules are stated as follows:

$$m_{ai}^{(t)}(U_i) \propto \int_{U_{\mathcal{N}(a)\backslash i}} \Psi_a(U_a) \prod_{j\in\mathcal{N}(a)\backslash i} \eta_{ja}^{(t_\star)}(U_j)\, \mathrm{d}U_{\mathcal{N}(a)\backslash i} \tag{8}$$

$$\eta_{ia}^{(t)}(U_i) \propto \prod_{k\in\mathcal{N}(i)\backslash a} m_{ki}^{(t_\star)}(U_i) \tag{9}$$

$$b_i^{(t)}(U_i) \propto \prod_{k\in\mathcal{N}(i)} m_{ki}^{(t)}(U_i) \tag{10}$$

where t and $t_\star$ are iteration counts such that $t_\star \le t$; $\mathcal{N}(a)$ is the set of all nodes' indexes connected to node index a, referred to as the neighborhood set of a; $\mathcal{N}(a)\backslash i$ is the neighborhood set of a minus node i; $b_i(u_i)$ is referred to as the belief at node i. The integral in (8) is multidimensional; however, since we are using Gaussian distributions, the integral can be solved in a closed form.

2.1.4. FGaBP update rules

To facilitate the derivation of the BP update rules, we use the following Gaussian function parameterization:

$$\mathcal{G}(\alpha,\beta) \propto \exp\left[\frac{-1}{2}\alpha u^2 + \beta u\right] \tag{11}$$

where α is the reciprocal of the variance and $\frac{\beta}{\alpha}$ is the Gaussian mean. This parameterization results in simplified BP message update formulas that are only functions of parameters α and β. The following is the formulation of the FGaBP algorithm update rules:

1. Iterate: $t = 0$ initialize all messages $\beta = 0$ and $\alpha = 1$.
2. Iterate: $t = 1, 2, \ldots$ and $t_\star \le t$.
3. For each VN$_i$ process: compute messages (α_{ia}, β_{ia}) to each FN$_a$, such that $a \in \mathcal{N}(i)$, as follows:

$$\alpha_i^{(t_\star)} = \sum_{k\in\mathcal{N}(i)} \alpha_{ki}^{(t_\star)}, \qquad \alpha_{ia}^{(t)} = \alpha_i^{(t_\star)} - \alpha_{ai}^{(t_\star)} \tag{12}$$

$$\beta_i^{(t_\star)} = \sum_{k\in\mathcal{N}(i)} \beta_{ki}^{(t_\star)}, \qquad \beta_{ia}^{(t)} = \beta_i^{(t_\star)} - \beta_{ai}^{(t_\star)}. \tag{13}$$

4. For each FN$_a$ process:
   (a) Receive messages ($\alpha_{ia}^{(t_\star)}$, $\beta_{ia}^{(t_\star)}$), where $i \in \mathcal{N}(a)$.
   (b) Define $\mathcal{A}^{(t_\star)}$ and $\mathcal{B}^{(t_\star)}$ such that $\mathcal{A}^{(t_\star)}$ is a diagonal matrix of incoming $\alpha_{ia}^{(t_\star)}$ parameters, and $\mathcal{B}^{(t_\star)}$ is a vector of incoming $\beta_{ia}^{(t_\star)}$ parameters. Then, compute W and K as follows:

$$W^{(t_\star)} = M + \mathcal{A}^{(t_\star)} \tag{14}$$

$$K^{(t_\star)} = B + \mathcal{B}^{(t_\star)} \tag{15}$$

   where M and B are the element a characteristic matrix and source vector as defined in (6) with $s = a$.
   (c) Partition $W^{(t_\star)}$ and $K^{(t_\star)}$ as follows:

$$W^{(t_\star)} = \begin{bmatrix} W_{\mathcal{L}(i)}^{(t_\star)} & V^T \\ V & \bar{W}^{(t_\star)} \end{bmatrix}, \qquad K^{(t_\star)} = \begin{bmatrix} K_{\mathcal{L}(i)}^{(t_\star)} \\ \bar{K}^{(t_\star)} \end{bmatrix}$$

   where $\mathcal{L}(i)$ is the local index corresponding to the global variable node i.
   (d) Compute and send FN$_a$ messages ($\alpha_{ai}^{(t)}$, $\beta_{ai}^{(t)}$) to each VN$_i$ as follows:

$$\alpha_{ai}^{(t)} = M_{\mathcal{L}(i)} - V^T(\bar{W}^{(t_\star)})^{-1}V \tag{16}$$

$$\beta_{ai}^{(t)} = B_{\mathcal{L}(i)} - (\bar{K}^{(t_\star)})^T(\bar{W}^{(t_\star)})^{-1}V. \tag{17}$$

5. Messages can be communicated according to any specified schedule such that $t_\star \le t$.
6. At message convergence, the means of the VNs, or solutions, μ_i can be obtained by:

$$\bar{u}_i^{(t)} = \mu_i^{(t)} = \frac{\beta_i^{(t)}}{\alpha_i^{(t)}}. \tag{18}$$

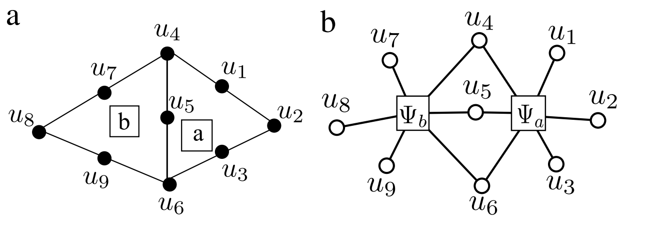

Fig. 1. (a) Sample FEM mesh of two-second order triangles. (b) The FEM-FG model.

The FGaBP update rules in (16) and (17) are based on distributed local computations performed using matrices of order $(n-1)$. Clearly, the FGaBP algorithm does not need to assemble a large global system of equations nor does it perform any operations on sparse matrices as required in conventional implementations of PCG. Also, the FGaBP update rules are generally applicable to any arbitrary element geometrical shape or interpolation order.

2.1.5. Boundary element treatment

Essential boundary conditions, such as Dirichlet conditions, are incorporated directly into Ψ_s in (6). Once boundary conditions are incorporated, the FGaBP communicates informative messages between variable nodes only. To setup boundary FNs, the VNs (U_s) of a boundary FN (Ψ_s) are first partitioned into $U_s = \left[U_{\{v\}}, U_{\{c\}}\right]^T$ where v is the set of indexes for interior VNs and c is the set of indexes of boundary VNs. Then, the boundary FN matrix M_s and vector B_s in (6) are modified as:

$$M_s = M_{vv} \tag{19}$$

$$B_s = B_v - M_{vc}U_c \tag{20}$$

where B_v, M_{vv}, and M_{vc} are the sub-vector and the sub-matrix partitions corresponding to the U_v and U_c partitions. After incorporating the boundary conditions, the FGaBP update rules are executed on boundary elements as normal.

2.2. The FMGaBP algorithm

As previously demonstrated, the FGaBP solves the FEM by passing locally computed messages on a graphical model comprised of factor and variable nodes. However, when the discretization

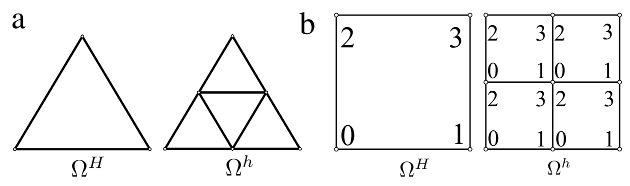

Fig. 2. (a) Mesh refinement by splitting each triangle in mesh Ω^H into four geometrically similar sub-triangles to produce a finer mesh Ω^h. (b) Local node numbering for each parent–child element set using quadrilaterals.

level becomes finer, or the number of elements and variables increase, the FGaBP convergence takes longer due to slower information propagation. While attempts to improve convergence through message relaxation schemes were proposed in [29], the number of iterations remains proportional to the number of problem variables. Multigrid (MG) acceleration schemes [35,36] provide optimal convergence speeds. As a result, the number of iterations are almost independent of the FEM mesh discretization level. The FGaBP algorithm should benefit greatly from MG schemes mainly because, BP communications on coarser levels can serve as bridges to communications between far away nodes on finer levels. This can considerably improve the overall information propagation in the FEM-FG model and thus enhance convergence. As shown later, the convergence rates in the multigrid FGaBP (FMGaBP) algorithm are independent of the domain's discretization level. In addition, FMGaBP retains the distributed nature of computations, which has important benefits for parallel implementation. Mainly, the FMGaBP level transfer operations are computationally decoupled and localized without requiring any global sparse algebraic operations. To illustrate the FMGaBP formulation, a multivariate distribution, associated with each FN$_a$, called the factor node belief b_a is defined. The belief b_a is in the form of a multivariate Gaussian distribution:

$$b_a^{(t)}(U_a) \propto \exp\left[-\frac{1}{2}U_a^T W_a^{(t)} U_a + (K_a^{(t)})^T U_a\right] \tag{21}$$

where U_a is a vector of random unknowns linked to the FN$_a$. The belief b_a, unlike the nodal belief b_i defined in (10), represents the marginal distribution as seen from FN$_a$ in a specific iteration t. The message update rules of the BP algorithm in (8)–(10) show that at message convergence the joint mean of the distribution b_a, given by $\bar{u}_a = W_a^{-1}K_a$, will be equal to the marginal means of each of the random unknowns U_a as computed from a global perspective by (18). In other words, the means computed from local beliefs will agree with the marginal means computed from the global perspective at message convergence. Using this observation, a quantity referred to as the belief residual r_a is formulated via:

$$r_a^{(t)} = K_a^{(t)} - W_a^{(t)}\bar{u}_a^{(t)}. \tag{22}$$

Using multiple grids with refinement by splitting and a consistent local node numbering between each set of parent–child elements as shown in Fig. 2, the belief residuals of each group of child elements can be locally restricted to the parent element:

$$K_a^H = \mathcal{R}_l r_a^h \tag{23}$$

where K_a^H is the source vector of the parent element, r_a^h is the accumulated local residual of child elements, and $\mathcal{R}_l$ is the child–parent local restriction operator. The iteration count (t) is dropped because both sides of the equation are operated in the same iteration. Similarly, we can use local operations for interpolation to apply the corrections from the coarse elements as follows:

$$\bar{u}_a^h \leftarrow \bar{u}_a^h + \mathcal{I}_l \bar{u}_a^H. \tag{24}$$

Using the level updated $\bar{u}_a^h$ we can reinitialize the corresponding level local messages with again using (22) but for $r_a \rightarrow 0$. We would then solve for K_a and the incoming messages. In this setup, the α_{ai} messages of each corresponding level is kept without modifications and only the β_{ai} messages are reinitialized. For self-adjoint problems, $\mathcal{R}_l$ is typically the transpose of $\mathcal{I}_l$. Since convergence is achieved when the level restricted local belief residuals approach zero, the resulting multigrid FMGaBP algorithm becomes a fixed-point algorithm for the true stationary solution point.

3. FGaBP reformulations

As can be seen from the FGaBP update rules in (12), (13), (16) and (17), the FGaBP computational complexity per iteration is dominated by the FN processes. Because of the matrix inversion required by each FN, the total FGaBP complexity per iteration, when using the Cholesky algorithm, is $O\big(N_f n(n-1)^3/3\big) \approx O\big(N_f n^4/3\big)$ where N_f is the number of FNs. For FEM problems, N_f is typically proportional to N. In addition, we have $n \ll N_f$, e.g. for triangular meshes $n = 3$ and tetrahedral meshes $n = 4$. More importantly, n is independent of N_f which implies that the FGaBP algorithm can offer high parallel scalability for a good choice of message scheduling, as will be shown later. However, due to the high FN computational complexity, the local computational cost may dominate as n increases when supporting higher degree FEM basis functions, or for example $n \geq 5$.

In this section, we present reformulations of FGaBP that reduces the FN complexity to $O\big(n^2\big)$. We also present schemes such as, element merging and color-based message scheduling, that enhance the parallel efficiency of the FGaBP algorithm by better exploiting its distributed computations.

3.1. Lower complexity FGaBP

We can reformulate the FN update rules using the partitioned matrix inversion identity as stated in the following:

$$Z = \begin{bmatrix} P & Q \\ R & S \end{bmatrix}, \qquad Z^{-1} = \begin{bmatrix} \tilde{P} & \tilde{Q} \\ \tilde{R} & \tilde{S} \end{bmatrix} \tag{25}$$

where:

$$\tilde{P} = N$$
$$\tilde{Q} = -NQS^{-1}$$
$$\tilde{R} = -S^{-1}RN$$
$$\tilde{S} = S^{-1} + S^{-1}RNQS^{-1}$$
$$N = \big(P - QS^{-1}R\big)^{-1}.$$

In our algorithm, P is a single element matrix and R is equal to Q^T with dimensions $(n-1)\times 1$. The FN update rules can alternatively be obtained by directly computing the inverse of W and partitioning it as follows:

$$(W^{(t_\star)})^{-1} = \begin{bmatrix} \tilde{W}_{\mathcal{L}(i)} & \tilde{C}^T \\ \tilde{C} & \tilde{D} \end{bmatrix}. \tag{26}$$

The resulting FN updates will be:

$$\alpha_{ai}^{(t)} = \frac{1}{\tilde{W}_{\mathcal{L}(i)}} - \alpha_{ia}^{(t_\star)} \tag{27}$$

$$\beta_{ai}^{(t)} = B_{\mathcal{L}(i)} + \frac{1}{\tilde{W}_{\mathcal{L}(i)}}(\bar{K}^{(t_\star)})^T\tilde{C}. \tag{28}$$

Using an in-place Cholesky inversion algorithm to compute $(W^{(t_\star)})^{-1}$, the FN complexity can be reduced to $O\big(n^3/3\big)$. Since W is small and dense, optimized linear algebra libraries can be used to compute its inverse efficiently, e.g. Eigen [37], Gmm++ [38], and Lapack [39].

For cases where n is relatively large, e.g. $n \geq 5$, computing the inverse can be costly. As shown in (7), the FEM solution requires only finding the marginal means which are the ratios of the nodal parameters β_i and α_i as shown in (18). From substituting (13) with (28) and rearranging terms, we obtain:

$$\bar{u}_a^{(t)} = (W^{(t_\star)})^{-1} K^{(t_\star)} \tag{29}$$

which can be seen as a local element approximate solution obtained from the FN_a for the $VN_i i \in \mathcal{N}(a)$ and for fixed α messages. If we partition W as $W = D - E - F$ such that D is the main diagonal of W while E and F are the lower and upper off-diagonals of W correspondingly, a local stationary, fixed-point, iterative process can be created within each FN. For example, the following will constitute a Gauss–Seidel (GS) iteration:

$$\bar{u}_a^{(t)} = (D - E)^{-1} F \bar{u}_a^{(t_\star)} + (D - E)^{-1} K^{(t_\star)}. \tag{30}$$

Other fixed-point iterative methods such as Jacobi or successive overrelaxation (SOR) can also be configured.

The α messages can be fixed after allowing the FGaBP algorithm to execute normally for a couple of iterations, or until the α messages converge to a very high tolerance such as 10^{-1}. This should replace the initial values in the α messages with better estimates. Then a number of iterations is executed using (29) to obtain a better estimate for $\bar{u}_a$. In all of our experiments, one or two iterations of GS was practically sufficient. Finally, the new $\beta_{ai}^{(t)}$ messages are obtained from the $\bar{u}_a^{(t_\star)}$ estimates as follows:

$$\beta_{ai}^{(t)} = \bar{u}_i^{(t_\star)} \alpha_i^{(t_o)} - \beta_i^{(t_\star)} + \beta_{ai}^{(t_\star)}, \tag{31}$$

where t_o is the iteration in which the α messages are fixed. Clearly using this approach, the FN process complexity is reduced to approximately $O(n^2)$. It can be shown that the fixed-point solutions, or marginal means, of the original FGaBP update rules are equal to the fixed-point solutions of the new FGaBP update rules. However, the final message parameters, α and β, may be different between the two algorithms. We refer to this reduced complexity algorithm as the approximated update (AU) FGaBP algorithm.

3.2. Element merging

In this section, variations on the graphical structure of the FGaBP algorithm are proposed in order to increase the parallel efficiency of the FGaBP execution on multicore architectures with suitable cache capacities. The FEM-FG factorization structure allows us to easily redefine new factors in (5) by joining other factors as follows:

$$\Psi_{\hat{s}}(U_{\hat{s}}) = \Psi_{s_1}(U_{s_1}) \Psi_{s_2}(U_{s_2}), \tag{32}$$

where s_1 and s_2 are joint into factor $\hat{s}$, which is a function of the variable set $U_{\hat{s}} = U_{s_1} \cup U_{s_2}$. Once suitable elements are identified for merging, the FGaBP algorithm can be executed normally after locally assembling the merged element matrices and source vectors. When the factors are adjacent, the cardinality of $U_{\hat{s}}$ is $|U_{\hat{s}}| = |U_{s_1}| + |U_{s_2}| - |U_{s_1} \cap U_{s_2}|$. Element Merging (EM) can be particularly advantageous if elements sharing edges in 2D meshes or surfaces in 3D meshes are merged. As a result, the memory bandwidth utilization can improve because of considerable edge reduction in the FEM-FG graph. By merging adjacent elements, the high CPU computational throughput can be better utilized in exchange for the slower memory bandwidth and latency.

EM can be mainly useful for 2-D triangular and 3-D tetrahedral meshes with first order FEM elements. To help illustrate the effect of merging, we define a quantity referred to as the percentage of Merge Reduction Ratio (MRR) which is obtained by dividing the amount of reduced memory due to the effect of merging by the original amount of memory before the merge. In general, the MRR is computed as:

$$\text{MRR} = \frac{\sum_{a \in \mathcal{D}} M_a - \sum_{a \in \mathcal{C}} M_a}{\sum_{a \in \mathcal{D}} M_a} \cdot 100\%, \tag{33}$$

where $\mathcal{D}$ is the set of all factors before any merge, $\mathcal{C}$ is the set of merged factors, and M_a is the amount of memory required for FN_a. Considering the particular implementation detailed later in Section 4, the memory complexity per factor node can be obtained by $M_i \propto O(4n_a + n_a^2)$ where n_a is the number of VN_a edges, or the rank of M_a. If we consider structured meshes, for illustration purposes only, the resulting MRR is 23.8% when merging every two adjacent triangles into a quadrilateral, or 46.4% when merging every four fully connected triangles into a quadrilateral. Fig. 3 illustrates the different FEM-FG graphs from merging every two or every four adjacent triangles in a structured triangular mesh.

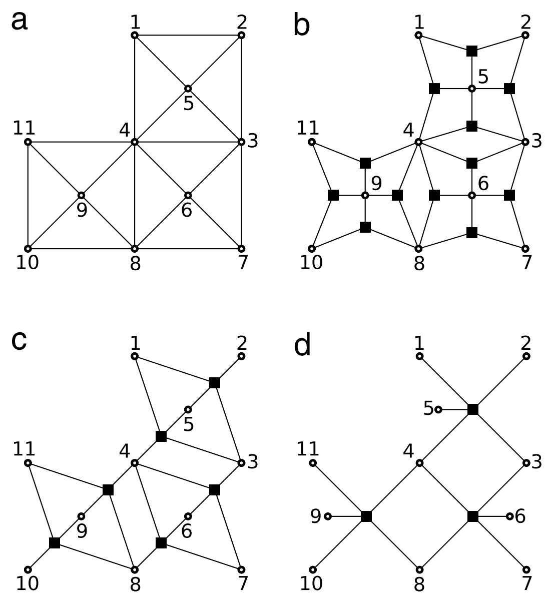

Fig. 3. Triangular element merging example. (a) Original triangular mesh. (b) The initial FEM-FG using single element factorization. (c) Merging two adjacent triangles. (d) Merging adjacent four triangles.

For irregular triangular meshes with a large number of triangles, Euler's formula [40, p. 28] states that each vertex will be surrounded by six triangles on average. Thus, when merging mostly six triangle groups into hexagons, the MRR increases to 38.9%, or to 42.9% when merging five triangles. Merging triangles does not have to be uniform. We may decide to merge patches of 2, 3, 4, 5, 6, or more as long as the triangles share edges and form connected, or preferably, convex regions. Similarly for 3D tetrahedral meshes, merging tetrahedrons sharing faces can also result in significant memory storage reductions. If two tetrahedrons of first order are merged, the MRR is 29.7%, and 53.1% when merging three tetrahedrons sharing faces and are enclosed by five vertexes.

While merging elements based on a structured mesh is a trivial operation, we can still efficiently merge certain element configurations in unstructured meshes by using partitioning algorithms [41–44]. Specifically, the work in [45] presents algorithms to create macro-elements by joining adjacent elements in unstructured grids. Partitioning may add some overhead in the preprocessing stage; however, since in practice the number of factors is much greater than the number of CPU cores, a lower partition quality can be used to lower the partitioning overhead time [42] without having much impact on the overall parallel efficiency.

The element merging does not affect the underlying FEM mesh discretization properties, it does however affect the FGaBP numerical complexity as a solver. Our results in Section 5.2 reveal that the overall computational complexity of the merged FEM-FG can be higher than that of the original, un-merged one. However, the FGaBP demonstrates considerable speedups for the merged FEM-FG structure, because of better utilization of available memory bandwidth and cache resources resulting from improved computational locality. These observations are illustrated in Section 5.2. To conclude, we propose to use the merge feature to control trade-offs between CPU resources such as memory bandwidth, cache utilization and CPU cycles, thus facilitate fine-tuned implementations on manycore architecture.

3.3. *Message scheduling*

Message communication in FGaBP can be performed subject to a particular schedule which can be sequential, parallel, or in any order. One of the key empirical observations of the FGaBP algorithm is the flexibility in message communication, which enables implementations that efficiently trade off computation with communication on various parallel architectures without compromising the numerical stability of the algorithm. However, message scheduling can considerably affect the convergence rate of the algorithm; therefore, a good message schedule exposes parallelism by exploiting the underlying connectivity structure of the problem [27].

There are two basic scheduling schemes for general BP messages, sequential (asynchronous) and parallel (synchronous). In sequential scheduling, all the FNs are sequentially traversed based on a particular order while their messages are communicated and synchronized one FN at a time. Each FN computes its messages based on the most recent nodal messages available within the iteration, which is $t_\star = t$. This message schedule results in the least number of FGaBP iterations; but, it does not offer much parallelism. In parallel message scheduling, all the FN are processed in parallel while using the α_i and β_i values computed at a previous iteration, $t_\star = t - 1$. An additional loop is needed to traverse all the VNs in parallel to compute new α_i and β_i values from updated α_{ai} and β_{ai}. Such scheduling offers a high degree of parallelism; however, it requires considerably higher number of iterations due to slower information propagation. To address shared memory architectures, we propose an element-based coloring schedule that exploits parallelism inherent in the FEM-FG graphical model while not significantly increasing the number of FGaBP iterations.

3.3.1. *Color-parallel scheduling*

To implement a color-parallel message schedule (CPS), an element coloring algorithm needs to be used. The mesh elements are colored so that no two adjacent elements have the same color symbol. Elements are deemed adjacent if they share at least a node. A simple mesh coloring diagram is illustrated in Fig. 4 using two types of meshes, a quadrilateral mesh and a triangular mesh. FN messages in each color group are computed and communicated in parallel.

To facilitate a CPS scheme, the FGaBP message updates are modified as follows:

$$\alpha_i^{(t)} = \alpha_i^{(t_\star)} + (\alpha_{ai}^{(t)} - \alpha_{ai}^{(t_\star)}) \tag{34}$$

$$\beta_i^{(t)} = \beta_i^{(t_\star)} + (\beta_{ai}^{(t)} - \beta_{ai}^{(t_\star)}). \tag{35}$$

In other words, a running sum of the α_i and β_i parameters are kept in each VN, initialized to zero at $t = 1$, while differences on the FN edge messages are only communicated by FN processes. In this scheme, there is no need for an additional loop to traverse and synchronize the VNs.

The FN processes are synchronized before starting each color group. This scheme is particularly efficient for multi-threaded

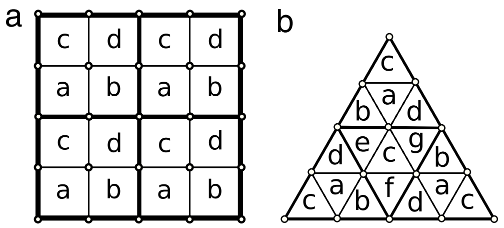

Fig. 4. Illustration of mesh element coloring showing two types of meshes. (a) Structured quadrilateral mesh containing a total of four colors. (b) Triangular mesh containing a total of six colors.

implementations on multicore CPUs or manycore GPUs, since thread-safety is automatically guaranteed by the CPS. A typical coloring algorithm would aim to produce the least number of colors; since, this will reduce the number of thread synchronizations needed at the end of each color group. However, because FEM meshes contain a very large number of elements, producing a reasonable number of colors using a low complexity algorithm can be sufficient as long as each color contains a nearly balanced number of elements. Our numerical tests, shown later in Section 5, indicate that FMGaBP with CPS results in competitive iteration counts compared to PCG with both Geometric MG (GMG) and Algebraic MG (AMG) preconditioning.

4. Implementation techniques

In this section we describe techniques used for the implementation of the AU-FGaBP and FMGaBP algorithms on manycore and multicore architectures using the CPS scheme.

4.1. *Data-structures*

Using the CPS scheme of FGaBP and assuming all FNs are of the same type, the overall storage requirement of FGaBP is $O\big(2N + N_f(n^2 + 4n)\big)$ in 64-bit words. This includes two vectors of size N for the VNs' α_i and β_i, and a data-structure of size N_f containing the collection of FN data-structures where each FN data-structure contain dense matrices M_a, vectors B_a, α_{ai}, and β_{ai}, and an integer vector storing local to global index associations. Also, this setup assumes that all indexes are stored in 64-bit integers and all real numbers are stored in 64-bit double-precision floating-points, which is essential for very large problems. Since usually $O(N_f) \approx O(N)$, the overall FGaBP memory complexity is $O(N)$, typical for sparse problems. However, unlike conventional sparse data-structures such as the compressed row storage (CRS), all the FGaBP data is dense. Hence, the FGaBP data-structure adds minimal overhead by eliminating the need to store complicated sparsity patterns. More importantly, iterative solvers in comparison, such as the PCG, require considerable memory not only to store the sparse matrix, but also the preconditioner.

The FMGaBP data-structures are mostly based on the FGaBP data-structures with the addition of another dense matrix per multigrid level. The added matrix stores the index associations of parent–child FNs for each hierarchical pair of coarse–fine levels. The total size of the FMGaBP memory can be obtained by:

$$\text{FMGaBP Mem.} \approx O\left[(Z + cN_f)\sum_{l=0}^{L-1}\frac{1}{c^l} - cN_f\right] \tag{36}$$

$$= O\left[(Z + cN_f)\frac{1-(1/c)^L}{1-(1/c)} - cN_f\right] \tag{37}$$

where l is the level index; L is the total number of levels; $Z = 2N + N_f(n^2 + 4n)$ which is the FGaBP memory on the finest level;

and c is the number of children, e.g. $c = 4$ for 2D quadrilateral meshes or $c = 8$ for 3D hexahedral meshes. Clearly, the overall memory complexity is linear in N as $L \rightarrow \infty$.

4.2. Multicore CPU implementation

Both the FGaBP and FMGaBP were programmed using C++ Object Oriented Programming (OOP) [46]. FGaBP and FMGaBP are parallelized on multicore using OpenMP [47].

4.3. Manycore GPU implementation

In the past decade, architectures with many (tens or hundreds of) integrated cores have been introduced and used in the scientific computing community. GPUs are a popular class of manycore processors offering greater opportunities of parallelism on a single chip than multicore CPUs. It is expected that adeptly parallel algorithms such as the FMGaBP to benefit from the increased parallelism of GPU architectures. In this section we will detail the implementation techniques used to evaluate the FMGaBP performance on GPU architectures.

4.3.1. Background

Attempts to port parts or all of the FEM computations to manycore architectures have been presented in previous works. Most of these works are aimed at accelerating either the global sparse matrix assembly or the global system solve stage. Constrained by the special characteristics of their applications, the works in [48,49] present simple assembly strategies suited for manycore architectures. Their proposed techniques are not applicable for general FEM applications but, rather, are mostly suited to the sparsity structure of their specific applications. Graph coloring schemes in [50–52] are used to partition the FEM elements to non-overlapping sets, in order to enable a parallel thread-safe execution of the assembly routines for the global sparse matrix.

Assembling a global matrix, to be solved in a separate stage, based on graph coloring and partitioning can be costly. In the case of FGaBP, coloring adds little overhead since the global sparse matrix is never assembled and the colored graph is used to process the FEM solution in parallel. Other work [53–55] propose strategies and novel sparse storage formats to reduce the memory footprint of the assembled sparse matrix. Since, FGaBP eliminates the need for assembling a large sparse matrix, many of the optimizations proposed in previous work are, thus, not applicable to our work.

Significant research has been aimed at improving the solve stage of the sparse system using GPUs. Most of these work focus on accelerating the execution of the compute intensive kernels in the Krylov solvers. As shown in [4,56] such efforts are mainly communication bound and are limited by the maximum performance achieved from parallelizing the compute intensive kernels. The assembled matrix is usually large and sparse and in many cases does not fit in the small and fast access memory levels of CPU and the co-processor memory.

FGaBP avoids assembling a global sparse matrix, thus, is a promising candidate for manycore architectures. Instead of solving a large sparse matrix, each FGaBP iteration needs to compute the incoming and outgoing messages for each factor node. The compute intensive kernel in a FGaBP iteration involves computing the inverses of many small dense matrices; an operation that is embarrassingly parallel and well suited for manycore architectures. Coloring the FEM-FG graphs improves the underlying parallelism significantly and allows FNs of the same color to be processed in parallel by hundreds of threads without producing any synchronization or memory collisions when accessing VN data. To demonstrate the embarrassingly parallel nature of both of the FGaBP and FMGaBP algorithms, we have implemented them on the NVIDIA Tesla C2075 GPU.

4.3.2. The GPU architecture

The NVIDIA Tesla C2075 GPU, which belongs to the Fermi generation, is used to illustrate the performance of the FMGaBP implementation on manycore architectures. The Tesla C2075 has a 6 GB DRAM, 448 CUDA cores, 48 KB of shared memory, and a memory bandwidth of 144 GB/s.

Current NVIDIA GPUs consist of streaming multiprocessors (SMs) each containing a few scalar processor cores (SPs), an on-chip shared memory, a data cache and a register file. Threads executing on each SM have access to the same shared memory and register file, which enables communication and data synchronization with little overhead. Threads executing on different SMs can only synchronize through the GPU of-chip global DRAM which incurs high latency of several hundred clock cycles. To parallelize an algorithm on GPUs, all of these architectural features have to be taken into account to efficiently use the available GPU resources.

The most popular APIs used to implement algorithms on GPUs are the Compute Unified Device Architecture (CUDA) and the Open Computing Language (OpenCL). CUDA 5.0 was used along with the library CUBLAS 5.0 [57] to implement the FMGaBP algorithm on the GPU. Initially, data has to be explicitly transferred from the CPU memory to GPU and then a collection of kernels are instantiated from the CPU to execute the parallel segments of the program on the GPU. Threads are grouped into blocks that are scheduled for execution on the GPU's SM. Groups of 32 threads in a block, called warps, are scheduled to execute the same instruction at the same time. Threads in the same block can communicate via on-chip shared memory with little latency while threads in different blocks have to go through GPU global memory for any kind of data synchronization [57].

4.3.3. GPU implementation details

FMGaBP is fully implemented on the NVIDIA Tesla C2075. The FNs, VNs, and level matrix data are transferred to the GPU once thus no GPU–CPU memory transfers are required during the algorithm's execution. The following details the GPU implementation of FMGaBP:

Multigrid restrict and prolong kernels: The restriction and prolongation stages are implemented in two different kernels. The parent–child mapping in the FMGaBP is loaded into shared memory to reduce global memory references. The compute intensive operation in the multigrid computations is the dense matrix–vector multiply for each parent FN in the coarser grid. The number of parent nodes assigned to each warp is computed by dividing the number of threads per warp (32) by the number of children for each parent. For example, in a 2D problem using quadrilateral meshes, each warp in the interpolation kernel applies corrections from eight FNs in the coarse grid to their children; thus allocating four threads to each FN in the coarse grid to parallelize the compute intensive kernels involved in the restrict operations.

Batch of inverses on GPUs: Computing the inverse of local matrices in the smoother iterations is the most time consuming operation in the FGaBP algorithm. Depending on the problem size, the number of matrices to be inverted can be very large. Various heuristics could be used to compute a batch of inverses on the GPU. Depending on the size of the local matrices, each inverse could be computed via one thread block, one warp or even one thread. For example, if the rank of each matrix is 256 then allocating one thread block (with 256 threads) to each matrix inverse would be efficient.

A batch of inverses can be computed using the NVIDIA CUBLAS library [57] for matrices up to rank 32. An in-place LU decomposition should first be performed and then the cublasDgetriBatched kernel is called to compute an out-of-place batch inversion. Since each warp computes the inverse of one matrix, the aforementioned kernel does not perform well for the low rank matrices in the

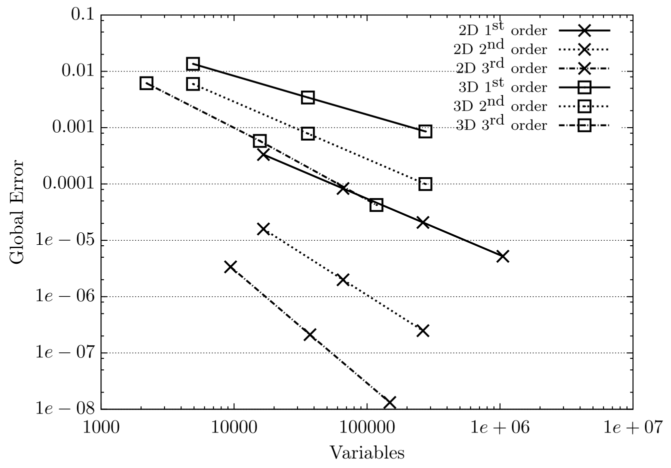

Fig. 5. The global error of the AU-FGaBP obtained from element-based l^2-norm error relative to the exact solution.

FMGaBP kernel. For 2D problems using quadrilateral meshes or 3D problems using tetrahedral meshes, our matrices are only of rank 4, thus when using the CUBLAS matrix inversion, the GPU resource will be underutilized and threads in a warp will not have enough work. Our matrix inversion kernel is customized to the matrix's dimension. The number of inverses computed via one warp is obtained through dividing the number of threads per warp (32) by the rank of the local dense matrices. For example, for a 2D problem with rank 4 local matrices, each warp computes 8 matrix inversions. We outperform the CUBLAS matrix inversion kernel by 2X when inverting a batch of 10 million rank 4 matrices. Another major advantage of our matrix inverse kernel is that it performs the inverse in-place on shared memory. As a result, the computed inverses do not have to be stored in global memory and the outgoing messages can be computed in the same kernel. Not storing the matrix inverses in the global memory enables the GPU to solve larger problems more readily.

Kernel fusion in FGaBP: The FGaBP iterations involve computing the incoming and outgoing messages and updating the VN's running sums. Instead of calling three separate GPU kernels one for each stage, we fuse these kernels and only call one GPU kernel for each iteration. Key advantages resulting from the fusion process are: First, data can be loaded completely into shared memory in order to be used by a single FGaBP kernel call reducing communication within the GPU's memory hierarchy. Second, the local matrix inverses can be generated on the fly and used to compute the running sum without the need to be stored in global memory. Lastly, kernel call overheads are also reduced by only calling one kernel for each FGaBP iteration.

5. Results and discussions

In this section, we present numerical and performance results of the new FGaBP and FMGaBP algorithms. The definite Helmholtz problem is first used to verify the numerical results of the AU-FGaBP algorithm. Then, the performance results of the enhanced FMGaBP algorithm on multicore and manycore architectures are presented. In all our experiments, unless otherwise stated, the iterations are terminated when the relative message error l^2-norm drops below 10^{-12} on the finest level.

5.1. FGaBP verification

The OOP based design of the FGaBP software facilitates its integration with existing frameworks of open-source software such as deal.II [6] and GetFEM++ [7]. The basic FGaBP formulation was previously demonstrated in [28,30] for 2D Laplace problems using both triangular and quadrilateral FEM elements as provided by the libraries GetFEM++ and deal.II. In this work, we verify the numerical results of the new AU-FGaBP formulation using the definite Helmholtz problem with a known solution for 2D and 3D domains as well as higher order FEM elements. The Helmholtz problem setup is provided by deal.II. The definite Helmholtz is formulated as follows:

$$-\nabla \cdot \nabla u + u = g, \quad \text{on } \Omega \tag{38}$$

$$u = f_1, \quad \text{on } \partial D \tag{39}$$

$$\mathbf{n} \cdot \nabla u = f_2, \quad \text{on } \partial N \tag{40}$$

where ∂D and ∂N are the Dirichlet and Neumann boundaries such that $\partial D \cup \partial N = \partial \Omega$, and Ω is the square or cubic domain bounded by $[-1, 1]$. Eq. (40) constitutes the non-homogeneous Neumann boundary condition. The right-hand-side functions are set, such that the exact solution is:

$$u(p) = \sum_{i}^{3} \exp\left(-\frac{|p - p_i|^2}{\sigma^2}\right) \tag{41}$$

where p is a spatial variable in (x, y, z), p_i are exponential center points chosen arbitrarily, and $\sigma = 0.125$. The library deal.II creates the mesh, and provides the FGaBP class with the elements' M, B, and local to global index data. The AU-FGaBP processes the FEM problem element-by-element in parallel to compute the solution. Afterwards, certain functions from the library deal.II are used to compute the final error relative to the exact solution of the Helmholtz problem on each individual element.

The AU-FGaBP uses α message tolerance of 10^{-1} and two GS iterations. The test cases are obtained by varying the FEM element order from the 1st to the 3rd order for both 2D quadrilaterals and 3D hexahedrals. Fig. 5 shows the global error plots for each test case. The global error plots are obtained by summing up the squares of the l^2-norm of the error on each element and then taking the square root of the result. As shown the AU-FGaBP obtains the expected error trends for the FEM; accuracy increases on all test cases with increasing number of elements as well as increasing FEM order.

5.2. Element merging performance

Element merging in the FGaBP algorithm is demonstrated using a structured triangular mesh on a unit square domain. The Laplace equation is solved in the domain using zero Dirichlet on the boundary. The unit square is subdivided into equally spaced sub-squares where each square is further divided into two right triangles. We perform two level merges by merging each two adjacent triangles, and then each four adjacent triangles. The original mesh has $N_f = 200\,000$ triangular element FN. Relaxation, as in [29], was not used in order to isolate the effect of merging on performance and iterations. The algorithm iterations were terminated when the message relative error l^2-norm reached $< 10^{-9}$. Table 1 shows the speedup results from merging. The experiments were performed on an Intel Core2 Quad CPU with clock frequency of 2.83 GHz.

The merge results in increasing speedups. The results illustrate that the execution time improves with increased merging which was mainly due to the improved locality of the algorithm. The complexity of the merged algorithms can approximately be stated as $O\big(TN_f(n^3 + n^2)\big)$, where T is the total number of iterations. The results also show reductions in iterations with increased merging, whereas the computational complexity increases due to increased local complexities in the factor nodes. These reductions in iterations may be attributed to the better numerical properties of the merged algorithms by correlating more local information in each factor belief. However the improved locality of the

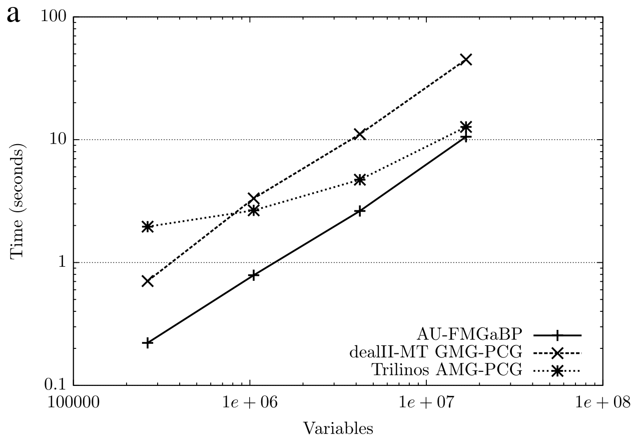

Fig. 6. Execution times on a 16-core node. (a) 2D execution times. (b) 3D execution times.

Table 1
AU-FGaBP with element merge speedups.

Merge	N_f	LDOF	Iteration ratio[a]	Complexity ratio[b]	Speedup
Un-merged	200 000	3	1.0	1.0	1.0
2-triangle	100 000	4	1.08	0.972	1.34
4-triangle	50 000	6	1.35	0.771	2.89

[a] Iterations ratio = iterations of un-merged/merged.
[b] Complexity ratio = complexity of un-merged/merged.

merge algorithms predominate the increase in overall complexity resulting in higher speedups. Mainly, improved locality results in better trade-offs of cache and memory bandwidth for cheaper CPU flops. Nonetheless, increased merging is expected to result in reduced performance; however, such high merge configurations would not be practical in most FEM problems.

5.3. Multicore performance

We compare our multicore implementation to two optimized parallel open-source implementations which are deal.II-MT (Multithreaded) and Trilinos [8]. The execution time for all runs are obtained on a SciNet Sandybride cluster node [58]. The node contains 2 × 8-core Xeon 2.0 GHz CPUs with 64 GB DRAM. The implementation from deal.II uses geometric MG (GMG) as a preconditioner with multithreaded parallelism, while the implementation from Trilinos uses Algebraic MG (AMG) as a preconditioner with MPI [59] parallelism. Since our focus is to show the efficiency of the parallel computations of the FEM solvers, we have used simple domains with well balanced partitions.

The AU-FMGaBP required 6 iterations for all 2D level runs; and 8 iterations for all 3D level runs. This illustrates that the FMGaBP, typical of multigrid schemes, results in optimal convergence. deal.II implementations show similar iteration results. Trilinos execution resulted in up to 14 iterations for 2D and 16 iterations for 3D since AMG typically requires more iterations than GMG.

The AU-FMGaBP algorithm was used with the CPS scheme. Fig. 6 shows the execution results for all implementations parallelized on 16-cores. Problem sizes ranging from 35K to 16.7M unknowns were used. The AU-FMGaBP demonstrated faster execution time in all runs while showing linear scalability with the number of unknowns. As the problem size increases, the overhead due to Trilinos's MPI calls reduces resulting in improved efficiency for larger problems. In contrast, the AU-FMGaBP demonstrated higher efficiency for all problem sizes. The single node ran out of memory for larger problems. However these results indicate that to efficiently address larger problems, the AU-FMGaBP needs to employ hybrid parallelism with multithreading and MPI on multiple nodes. Such

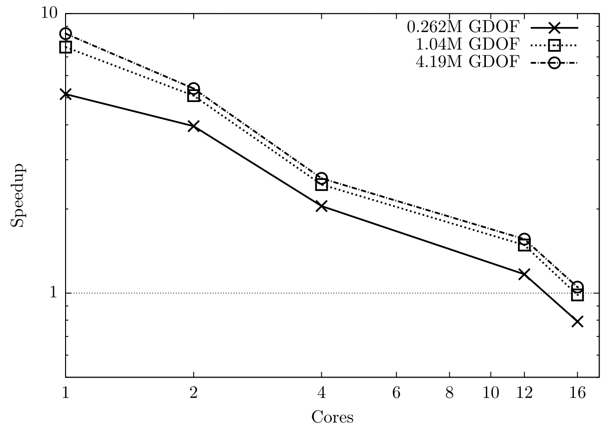

Fig. 7. The speedup achieved from accelerating FMGaBP on NVIDIA Tesla C2075 compared to the parallel implementation of the method on 1–16 CPU cores.

implementation requires specialized message scheduling algorithms and mesh partitioning schemes which support the element merging features. This implementation is planned for future work.

5.4. Manycore performance

The FMGaBP is implemented on NVIDIA Tesla C2075 for the 2D Helmholtz problem with number of unknowns ranging from 26K to 4.1M. Larger problems should be executed on a cluster of GPUs because of the GPU global memory size limits. Fig. 7 shows the speedup achieved by implementing FMGaBP on a single GPU versus the proposed parallel CPU implementation of the method on the SciNet 2 × 8-core Xeon processor. The speedup scalability is also presented in the figure by altering the number of threads for the CPU runs. As shown in the figure, the Tesla C2075 outperforms the CPU with up to 12 cores for all problem sizes. Larger problems are able to utilize the GPU resources more efficiently thus the GPU is faster than the 16-core CPU node for the largest problem with 4.1M unknowns. The only case where the GPU did not demonstrate speedups was for the smaller problem sizes (26K and 1M unknowns). The average (speedup over all problem sizes) achieved from the GPU implementation compared to the dual-core, quad-core and 12-core CPU settings are 4.8X, 2.3X and 1.5X respectively.

6. Conclusions

Enhanced distributed FEM algorithms based on probabilistic inference are presented showing high degree of efficiency on parallel architectures. Multicore implementations of our algorithms showed considerable speedups on most experiments against highly optimized open-source libraries; while, manycore implementations showed speedups on various problem sizes, outperforming 12-core CPUs. Future work will explore extending the FGaBP to solve nonlinear FEM applications along with support for local element assembly in order to further increase parallelism. In addition, hybrid multithreaded and MPI implementation based on distributed message scheduling on cluster architectures will be investigated.

Acknowledgments

The authors would like to acknowledge financial support from NSERC (The Natural Sciences and Engineering Research Council of Canada), ReSMiQ (Le Regroupement Stratégique en Microsystèmes du Québec) and SYTACom (FQRNT Center for Advanced Systems and Technologies in Communications). Some computations were performed on the Sandy supercomputer at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; Ontario Research Fund—Research Excellence; and the University of Toronto.

References

[1] Richard Vuduc, James W. Demmel, Katherine A. Yelick, OSKI: A library of automatically tuned sparse matrix kernels, in: Proceedings of SciDAC 2005, San Francisco, CA, USA, June, J. Phys. Conf. Ser. (2005) Institute of Physics Publishing.
[2] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, Philadelphia, PA, 2003.
[3] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, Optimization of sparse matrix–vector multiplication on emerging multicore platforms, in: Proc. ACM/IEEE Conf. Supercomputing, SC'07, 2007, pp. 1–12. http://dx.doi.org/10.1145/1362622.1362674.
[4] Mark Frederick Hoemmen, Communication-avoiding Krylov subspace methods (Ph.D. thesis), EECS Department, University of California, Berkeley, 2010, April. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-37.html.
[5] A.T. Chronopoulos, C.W. Gear, s-step iterative methods for symmetric linear systems, J. Comput. Appl. Math. 25 (2) (1989) 153–168. http://dx.doi.org/10.1016/0377-0427(89)90045-9. URL http://www.sciencedirect.com/science/article/pii/0377042789900459.
[6] W. Bangerth, R. Hartmann, G. Kanschat, deal.II—a general purpose object oriented finite element library, ACM Trans. Math. Software 33 (4) (2007) 24/1–24/27.
[7] Yves Renard, Julien Pommier, GetFEM++—an open-source finite element library. http://download.gna.org/getfem/html/homepage/.
[8] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, Kendall S. Stanley, An overview of the trilinos project, ACM Trans. Math. Software (ISSN: 0098-3500) 31 (3) (2005) 397–423. http://doi.acm.org/10.1145/1089014.1089021 URL http://trilinos.sandia.gov.
[9] Martin Kronbichler, Katharina Kormann, A generic interface for parallel cell-based finite element operator application, Comput. & Fluids (ISSN: 0045-7930) 63 (0) (2012) 135–147. http://dx.doi.org/10.1016/j.compfluid.2012.04.012. URL http://www.sciencedirect.com/science/article/pii/S0045793012001429.
[10] Judea Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.
[11] Kevin P. Murphy, Yair Weiss, Michael I. Jordan, Loopy belief propagation for approximate inference: An empirical study, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI'99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN: 1-55860-614-9, 1999, pp. 467–475.
[12] Jonathan S. Yedidia, William T. Freeman, Yair Weiss, Constructing free energy approximations and generalized belief propagation algorithms, IEEE Trans. Inform. Theory 51 (2004) 2282–2312.
[13] Yair Weiss, William T. Freeman, Correctness of belief propagation in Gaussian graphical models of arbitrary topology, Neural Comput. (ISSN: 0899-7667) 13 (10) (2001) 2173–2200.
[14] Martin J. Wainwright, Tommi Jaakkola, Alan S. Willsky, Tree-based reparameterization framework for analysis of sum–product and related algorithms, IEEE Trans. Inform. Theory 49 (5) (2003) 1120–1146.
[15] F.R. Kschischang, B.J. Frey, H.A. Loeliger, Factor graphs and the sum–product algorithm, IEEE Trans. Inf. Theory (ISSN: 0018-9448) 47 (2) (2001) 498–519.
[16] Martin J. Wainwright, Michael I. Jordan, Graphical models, exponential families, and variational inference, Found. Trends Mach. Learn. (ISSN: 1935-8237) 1 (2008) 1–305. http://dx.doi.org/10.1561/2200000001.
[17] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, D.A. Spielman, Improved low-density parity-check codes using irregular graphs, IEEE Trans. Inform. Theory (ISSN: 0018-9448) 47 (2) (2001) 585–598. http://dx.doi.org/10.1109/18.910576.
[18] Robert J. Mceliece, David J.C. Mackay, Jung fu Cheng, Turbo decoding as an instance of pearls belief propagation algorithm, IEEE J. Sel. Areas Commun. 16 (1998) 140–152.
[19] T.J. Richardson, R.L. Urbanke, The capacity of low-density parity-check codes under message-passing decoding, IEEE Trans. Inform. Theory (ISSN: 0018-9448) 47 (2) (2001) 599–618. http://dx.doi.org/10.1109/18.910577.
[20] W.T. Freeman, E.C. Pasztor, Learning low-level vision, in: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, Vol. 2, 1999, pp. 1182–1189. http://dx.doi.org/10.1109/ICCV.1999.790414.
[21] Jian Sun, Nan-Ning Zheng, Heung-Yeung Shum, Stereo matching using belief propagation, IEEE Trans. Pattern Anal. Mach. Intell. (ISSN: 0162-8828) 25 (7) (2003) 787–800. http://dx.doi.org/10.1109/TPAMI.2003.1206509. URL http://dx.doi.org/10.1109/TPAMI.2003.1206509.
[22] Brendan J. Frey, Ralf Koetter, Nemanja Petrovic, Very loopy belief propagation for unwrapping phase images, in: Advances in Neural Information Processing Systems, vol. 14, 2001, pp. 737–743. URL http://media.nips.cc/nipsbooks/nipspapers/paper_files/nips14/AA39.pdf.
[23] V. Kolmogorov, Convergent tree-reweighted message passing for energy minimization, IEEE Trans. Pattern Anal. Mach. Intell. (ISSN: 0162-8828) 28 (10) (2006) 1568–1583. http://dx.doi.org/10.1109/TPAMI.2006.200.
[24] Talya Meltzer, Chen Yanover, Yair Weiss, Globally optimal solutions for energy minimization in stereo vision using reweighted belief propagation, in: International Conference on Computer Vision, 2005, pp. 428–435.
[25] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, Carsten Rother, A comparative study of energy minimization methods for markov random fields with smoothness-based priors, IEEE Trans. Pattern Anal. Mach. Intell. (ISSN: 0162-8828) 30 (6) (2008) 1068–1080. http://doi.ieeecomputersociety.org/10.1109/TPAMI.2007.70844.
[26] O. Shental, P.H. Siegel, J.K. Wolf, D. Bickson, D. Dolev, Gaussian belief propagation solver for systems of linear equations, in: IEEE Int. Symp. on Inform. Theory (ISIT), 6-11 2008, pp. 1863–1867.
[27] Y. El-Kurdi, W.J. Gross, D. Giannacopoulos, Efficient implementation of Gaussian belief propagation solver for large sparse diagonally dominant linear systems, IEEE Trans. Magn. (ISSN: 0018-9464) 48 (2) (2012) 471–474. http://dx.doi.org/10.1109/TMAG.2011.2176318.
[28] Y. El-Kurdi, W.J. Gross, D. Giannacopoulos, Parallel solution of the finite element method using Gaussian belief propagation, in: The 15th Biennial IEEE Conference on Electromagnetic Field Computation, 2012, p. 141.
[29] Y. El-Kurdi, D. Giannacopoulos, W.J. Gross, Relaxed Gaussian belief propagation, in: IEEE Int. Symp. on Inform. Theory, ISIT, 2012, pp. 2002–2006. http://dx.doi.org/10.1109/ISIT.2012.6283652. July.
[30] Y. El-Kurdi, W.J. Gross, D. Giannacopoulos, Parallel multigrid acceleration for the finite element Gaussian belief propagation algorithm, IEEE Trans. Magn. (ISSN: 0018-9464) 50 (2) (2014) 581–584. http://dx.doi.org/10.1109/TMAG.2013.2284483.
[31] Yousef El-Kurdi, Parallel Finite Element Processing Using Gaussian Belief Propagation Inference on Probabilistic Graphical Models (Ph.D. thesis), ECE Department, McGill University, Montreal, Canada, 2014.
[32] Peter P. Silvester, Ronald L. Ferrari, Finite Elements For Electrical Engineers, third ed., Cambridge University Press, New York, USA, ISBN: 0-521-44505-1, 1996.
[33] Jianming Jin, The Finite Element Method in Electromagnetics, second ed., Wiley–IEEE Press, New York, USA, 2002.
[34] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, in: Adaptive Computation and Machine Learning, MIT Press, ISBN: 9780262013192, 2009.
[35] U. Trottenberg, C.C.W. Oosterlee, A. Schüller, Multigrid, Academic Press, ISBN: 9780127010700, 2001.
[36] William L. Briggs, Van Emden Henson, Steve F. McCormick, A Multigrid Tutorial, second ed., SIAM, Philadelphia, PA, USA, ISBN: 0-89871-462-1, 2000.
[37] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. 2010. http://eigen.tuxfamily.org.
[38] Gmm++: a generic template matrix c++library. http://download.gna.org/getfem/html/homepage/gmm.html. retrieved 2013.
[39] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, third ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, ISBN: 0-89871-447-8, 1999, paperback.
[40] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars, Computational Geometry: Algorithms and Applications, third ed., Springer-Verlag TELOS, Santa Clara, CA, USA, ISBN: 978-3-540-77973-5, 2008.
[41] Dominique Lasalle, George Karypis, Multi-threaded graph partitioning, Int. Parallel Distrib. Process. Symp., 0 (ISSN: 1530-2075) (2013) 225–236. http://doi.ieeecomputersociety.org/10.1109/IPDPS.2013.50.
[42] George Karypis, Kirk Schloegel, PARMETIS—Parallel Graph Partitioning and Sparse Matrix Ordering Library Version 4.0. University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN 55455, March 2013.

[43] George Karypis, Vipin Kumar, MeTis: nstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0, 2009. http://www.cs.umn.edu/~metis.

[44] George Karypis, Vipin Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs, in: Proceedings of the 1996 ACM/IEEE Conference on Supercomputing, Supercomputing'96, Washington, DC, USA, IEEE Computer Society, ISBN: 0-89791-854-1, 1996. http://dx.doi.org/10.1145/369028.369103. URL http://dx.doi.org/10.1145/369028.369103.

[45] T.F. Chan, J. Xu, L. Zikatanov, Eberly College of Science Department of Mathematics, An Agglomeration Multigrid Method for Unstructured Grids, in: PSU Applied Mathematics Report Series, Penn State Department of Mathematics, 1998.

[46] B. Stroustrup, The C++ Programming Language, Pearson Education, ISBN: 9780133522853, 2013.

[47] OpenMP architecture review board. OpenMP application program interface, 2013. URL http://openmp.org/wp/openmp-specifications/.

[48] Jeff Bolz, Ian Farmer, Eitan Grinspun, Peter Schröoder, Sparse matrix solvers on the gpu: Conjugate gradients and multigrid, ACM Trans. Graph. (ISSN: 0730-0301) 22 (3) (2003) 917–924. http://dx.doi.org/10.1145/882262.882364. URL http://doi.acm.org/10.1145/882262.882364.

[49] Javier Rodríguez-Navarro, Antonio Susín, Non structured meshes for cloth gpu simulation using fem, in: VRIPHYS, 2006, pp. 1–7.

[50] Dimitri Komatitsch, David Michéa, Gordon Erlebacher, Porting a high-order finite-element earthquake modeling application to nvidia graphics cards using cuda, J. Parallel Distrib. Comput. (ISSN: 0743-7315) 69 (5) (2009) 451–460. http://dx.doi.org/10.1016/j.jpdc.2009.01.006. URL http://dx.doi.org/10.1016/j.jpdc.2009.01.006.

[51] Jan S. Hesthaven, Tim Warburton, Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications, first ed., Incorporated Springer Publishing Company, 2007, ISBN 0387720650, 9780387720654.

[52] Cris Cecka, Adrian J. Lew, E. Darve, Assembly of finite element methods on graphics processors, Internat. J. Numer. Methods Engrg. (ISSN: 10970207) 85 (5) (2011) 640–669. http://dx.doi.org/10.1002/nme.2989. URL http://dx.doi.org/10.1002/nme.2989.

[53] A. Dziekonski, P. Sypek, A. Lamecki, M. Mrozowski, Generation of large finite-element matrices on multiple graphics processors, Internat. J. Numer. Methods Engrg. 94 (2) (2013) 204–220.

[54] Adam Dziekonski, Piotr Sypek, Adam Lamecki, Michal Mrozowski, Finite element matrix generation on a GPU, Progr. Electromagn. Res. 128 (2012).

[55] Zhisong Fu, T. James Lewis, Robert M. Kirby, Ross T. Whitaker, Architecting the finite element method pipeline for the gpu, J. Comput. Appl. Math. 257 (2014) 195–211.

[56] Maryam MehriDehnavi, Yousef El-Kurdi, James Demmel, Dennis Giannacopoulos, Communication-avoiding Krylov techniques on graphic processing units, IEE Trans. Magn. 49 (5) (2013) 1749–1752.

[57] NVIDIA Corporation. NVIDIA CUDA C Programming Guide, 2013.

[58] Chris Loken, Daniel Gruner, Leslie Groer, Richard Peltier, Neil Bunn, Michael Craig, Teresa Henriques, Jillian Dempsey, Ching-Hsing Yu, Joseph Chen, L. Jonathan Dursi, Jason Chong, Scott Northrup, Jaime Pinto, Neil Knecht, Ramses Van Zon, SciNet: Lessons learned from building a power-efficient Top-20 system and data centre, J. Phys. Conf. Ser. (ISSN: 1742-6596) 256 (1) (2010). http://dx.doi.org/10.1088/1742-6596/256/1/012026.

[59] MPI Forum. Message Passing Interface (MPI) Forum Home Page. http://www.mpi-forum.org/ (Dec. 2009).